Formalisation du comportement des objets gérés dans le cadre du modele OSI

Olivier Festor

Université Henri Poincaré
Nancy I

Centre de Recherche en
Informatique de Nancy

[UFR STMIA]
École Doctorale IAE + M
Département de Formation Doctorale en Informatique

Formalisation du comportement des objets gérés dans le cadre du modèle OSI

THÈSE

présentée et soutenue publiquement le 1er Octobre 1994

en vue de l'obtention du titre de

Docteur de l'Université Henri Poincaré - Nancy I

(Spécialité Informatique)

par

Olivier FESTOR

Composition du jury

MM.	Jean-Pierre THOMESSE	*Président, Rapporteur*
	Jean-Pierre FINANCE	*Examinateur*
	Jacques LABETOULLE	*Examinateur, Rapporteur*
	Jean-Jacques PANSIOT	*Examinateur, Rapporteur*
	André SCHAFF	*Examinateur*
	Georg ZÖRNTLEIN	*Examinateur*

Remerciements

De nombreuses personnes ont contribué à la réussite de cette thèse et m'ont aidé tout au long de ces trois dernières années. Très touché par leurs encouragements, je tiens à les remercier tous.

Je tiens tout d'abord à remercier Jean-Pierre Thomesse, Professeur à l'ENSEM, de me faire l'honneur de présider ce jury et d'avoir accepté, malgré les charges qui sont les siennes, d'être également rapporteur de cette thèse.

Je remercie Jean-Pierre Finance, Président de l'Université Henri Poincaré - Nancy I, d'avoir accepté de juger le résultat des travaux qu'il m'a permis d'entreprendre en acceptant d'en être le Directeur et en m'accueillant au sein de l'équipe PROGRAIS du Centre de Recherche en Informatique de Nancy.

En acceptant d'écrire un rapport sur cette thèse,Jacques Labetoulle, Professeur à l'Institut EURECOM, et Jean-Jacques Pansiot, Professeur à l'Université Louis Pasteur de Strasbourg, m'ont fait un grand plaisir. La rigueur et la minutie de leur lecture ainsi que les commentaires constructifs qu'ils ont émis m'ont été très bénéfiques.

Ma gratitude va également à André Schaff, Maître de Conférences à l'Université Henri Poincaré - Nancy I, qui, dès le début de ma thèse, a dirigé et encadré mon travail. André Schaff s'est énormément investi dans cette tâche et si cette thèse a pu aboutir aujourd'hui, c'est en grande partie grâce à lui.

Georg Zörntlein était responsable du suivi de mon travail au sein de l'European Networking Center d'IBM à Heidelberg. Sa grande disponibilité et sa grande maîtrise scientifique m'ont permis d'avancer plus rapidement dans mes recherches en m'évitant bien souvent de m'égarer sur des sentiers par trop hasardeux. Qu'il en soit ici remercié.

Je suis très reconnaissant à Lothar Mackert, Directeur de l'IBM European Networking Center, et Gautam Kar, Chef du Département "Systems and Network Management", d' avoir cru en mes idées et de m'avoir permis de faire cette thèse au sein de l'IBM ENC à Heidelberg.

Je tiens également à remercier tout particulièrement Juergen Schneider. Tout au long de ces trois années de thèse, il a toujours soutenu mon travail, n'hésitant pas à prendre sur son temps personnel pour m'écouter, me conseiller et m'encourager dans mon effort.

Je remercie très sincèrement mes collègues de travail, grâce auxquels ma culture scientifique a sensiblement évolué durant les trois années passées à l'ENC; en particulier Brigitte Bär pour les nombreux échanges scientifiques que nous avons eus, Alexander Clemm pour l'excellente coopération scientifique et pour avoir su me faire partager sa passion pour la Californie et les "roller-coaster", Wilko Eschebach qui m'a beaucoup aidé dans le développement des outils, mon collègue Jean Monnery qui m'a permis de découvrir un autre domaine majeur des réseaux qui est celui des messageries, et "last but not least" mon collègue et ami Stefan Peuser avec qui j'ai partagé le bureau pendant trois ans et avec qui la coopération, basée sur une entente parfaite, a été très fructueuse.

Je remercie également l'ensemble des élèves de l'Ecole Supérieure d'Informatique et Applications de Lorraine qui ont travaillé dans les différents projets industriels que j'ai proposés et supervisés, ainsi que les stagiaires que j'ai encadrés à l'ENC et plus spécialement mon collègue et ami David Orain qui s'est beaucoup investi dans le développement des outils présentés dans cette thèse.

Je dédie cette thèse à mes parents ainsi qu'à Pascale, ma future épouse. Tout au long de ces huit années d'études, ils m'ont constamment témoigné leur soutien et affection sans faille. La réussite de ce travail est également la leur.

Merci à tous.

Table des matières

Liste des Figures

Introduction générale

La croissance des réseaux

Les réseaux d'ordinateurs, vitaux au développement de la communication, ont évolué ces dernières années de façon exponentielle. Cette progression s'est traduite dans un premier temps par une augmentation considérable du nombre de systèmes hôtes (terminaux connectés à un réseau) puis, par l'interconnexion de plus en plus fréquente de différents réseaux, connexions entre réseaux locaux d'entreprises (LAN[1]), réseaux métropolitains (MAN[2]) et grands réseaux internationaux (WAN[3]).

Parallèlement à cette croissance en taille, les systèmes de communication ont dû s'adapter à de nouvelles exigences formulées par leurs utilisateurs. Il s'agit principalement d'une demande de services de plus en plus importante ainsi que de critères sévères concernant la puissance et la fiabilité de ces services.

La réalisation de systèmes de communication pouvant fournir de tels services a été rendue possible par l'utilisation de nouvelles technologies matérielles et logicielles. Citons, à titre d'exemple, la fibre optique comme support de transmission. La conception de nouveaux protocoles adaptés à de telles technologies permet d'assurer, d'une part une qualité de services de très haut niveau, et d'autre part, avec l'interconnexion de nombreux réseaux, la multiplication du volume des services offerts.

Les nouveaux réseaux puissants et fiables qui proposent de nombreux services de différentes natures tels que le transfert de fichier, le traitement transactionnel ou les messageries multi-médias sont aujourd'hui une réalité. La croissance de l'utilisation de ces réseaux entraîne de nouveaux besoins plus spécifiques envers leur administration.

La gestion des réseaux

Pour maîtriser ces nouveaux systèmes et satisfaire les exigences dues aux utilisateurs et aux nouvelles applications distribuées, le besoin de suivi et de manipulation des ressources des réseaux est apparu comme une activité vitale à leur pérénité. Ces différentes activités de suivi et de manipulation sont regroupées au sein de la gestion des réseaux.

Les ressources d'un réseau sont fréquemment issues de constructeurs différents, alors que les systèmes de gestion actuellement offerts sur le marché sont encore trop souvent des systèmes propriétaires ne pouvant gérer que les composants d'un seul constructeur. Or, l'interconnexion et l'hétérogénéité des différents réseaux rendent ces systèmes de gestion inopérants car ils sont limités à un sous-réseau. Ils deviennent inutilisables dans un environnement hétérogène. En effet, chaque ressource présente des caractéristiques différentes tant au point de vue de son architecture que de ses fonctionnalités de gestion.

1. . Local Area Network
2. . Metropolitan Area Network
3. . Wide Area Network

Afin de permettre une gestion intégrée de l'ensemble de ces ressources, il a fallu définir une série de textes qui normalisent les différents composants et les fonctions de l'activité de gestion. Ces normes doivent à la fois contenir un modèle décrivant la répartition des rôles de gestion au sein d'un système de communication et définir les protocoles de communication utilisés. Ces derniers ont pour but de permettre l'échange d'informations de gestion entre les différents intervenants. De plus, les normes doivent mettre à disposition des concepteurs de systèmes de communication des moyens de modélisation des ressources et de description de l'information échangée entre intervenants de gestion.

Si de nombreux systèmes spécifiques de gestion sont aujourd'hui proposés, seuls deux modèles sont à ce jour standardisés (resp. normalisés) ou en phase de l'être. Ces deux modèles sont, d'une part l'approche SNMP[4] standard de l'IAB[5] pour les systèmes de communication basés sur les protocoles TCP/IP[6], et d'autre part l'approche OSI[7] normalisée dans le cadre de l'ISO[8].

Conscient de la nécessité de gérer les ressources d'un réseau de communication, l'ISO a entrepris la normalisation des activités de gestion dès la définition du modèle de référence. Cependant, c'est au cours des deux dernières années que les travaux réalisés dans ce domaine ont conduit à la définition de normes stables pouvant servir de base au développement réel de systèmes de gestion de réseaux hétérogènes.

Le besoin de formaliser les modèles

Chaque approche propose un modèle spécifique pour la description des ressources de gestion. Ce modèle, basé sur une approche formelle, comprend notamment un langage de spécification des ressources gérées.

Dans l'approche SNMP, ce langage de spécification est basé sur une description par des variables simples des ressources de gestion. Cette description ne prend pas en compte la spécification des opérations possibles sur les interfaces des ressources gérées. Une telle approche a l'avantage d'être simple (toute ressource est modélisée par un ensemble de variables simples). L'inconvénient est qu'elle ne permet ni la définition d'opérations complexes sur les ressources, ni la description formelle du comportement associé.

L'approche de gestion OSI intègre également dans sa version actuelle un langage de spécification des ressources de gestion. Ce langage est basé sur les concepts des méthodes de développement orientées-objets. Il permet de modéliser les ressources à gérer à partir d'un ensemble d'objets caractérisés par des données et des opérations normalisées sur ces données.

Si ce langage permet effectivement de spécifier de manière formelle les données associées aux objets et les signatures des opérations de gestion permises, il ne permet ni la spécification formelle du comportement d'un tel objet c'est à dire le comportement observable aux interfaces de celui-ci, ni la spécification des interfaces autres que celle de gestion.

L'absence de formalisme pour la description du comportement ne favorise en rien l'interconnexion des systèmes ouverts et laisse libre cours aux spécifieurs de systèmes. Dans la plupart des spécifications d'objets gérés existantes, la description du comportement n'est pas fournie ou contient des informations qui ne permettent pas d'identifier le comportement exact du système spécifié.

Afin d'obtenir une spécification complète, précise, et non-ambigue des systèmes de gestion de réseau, il est nécessaire de disposer d'une technique de description formelle permettant, d'une part la spécification des données de gestion, et d'autre part la spécification formelle du comportement des objets gérés. Or,

4. . Simple Network Management Protocol
5. . Internet Activity Board
6. . Transmission Control Protocol/ Internet Protocol
7. . Open Systems Interconnection
8. . International Standard Organization

le langage de spécification des données de gestion normalisé à ce jour, ne comporte aucun formalisme pour la description du comportement.

Le cadre de travail et la motivation du sujet

Les travaux entrepris dans cette thèse et présentés dans ce mémoire ont été réalisés dans le cadre d'une coopération entre le Centre de Recherche en Informatique de Nancy (CRIN) et l'European Networking Center d'IBM (ENC) à Heidelberg en Allemagne. Au sein du département "Systems and Network Management" de l'ENC, dans lequel cette thèse a été réalisée, de nombreux chercheurs travaillent au développement d'outils d'applications de gestion de réseaux.

Dans le cadre des travaux en cours sur le développement d'outils d'aide à l'élaboration d'applications de gestion basées sur les normes OSI en vigueur, le besoin de disposer d'un environnement formel pour la spécification du comportement des objets de gestion est rapidement apparu. Les nombreuses discussions avec les personnes qui utilisent l'approche OSI pour modéliser les ressources gérées au sein d'un réseau ont permis d'identifier l'inadéquation entre le modèle de description du comportement des ressources proposé par la norme et les besoins réels des spécifieurs et des outils de spécification.

Dans le cadre des normes OSI, l'ISO impose un langage de spécification pour la modélisation des ressources de gestion de réseaux. Ce langage, appelé GDMO[9], est reconnu et largement utilisé à travers le monde. De plus, de nombreux catalogues d'objets de gestion décrits à l'aide de ce formalisme sont actuellement disponibles. On compte pour le moment plus d'une dizaine de catalogues comprenant plusieurs centaines d'objects.

Mais ce langage ne traite absolument pas les aspects relatifs à la description du comportement des ressources à modéliser. Vu les besoins en formalismes pour le comportement et les normes incomplètes mais reconnues, le problème posé se résume de manière suivante:

«Il faut des mécanismes formels pour spécifier le comportement des objets gérés. Ces mécanismes doivent parfaitement s'intégrer dans les normes actuelles afin de garder le maximum de compatibilité avec l'existant. Ces nouveaux mécanismes de spécification doivent faciliter le travail des spécifieurs, être facile d'emploi, et être intégrés dans un environnement de développement afin d'en accroître les fonctionnalités».

Ces besoins exprimés par des concepteurs de systèmes de gestion de réseaux basés sur l'approche OSI forment le cahier des charges de base des recherches entreprises dans cette thèse.

L'approche proposée

Afin de formaliser le comportement des ressources gérées, plusieurs approches sont possibles. De nombreuses techniques de description formelle ont été appliquées avec succès dans le cadre général de la spécification du comportement des objets, ou dans le cadre plus particulier de la spécification de systèmes distribués. Le cas de la spécification du comportement des objets gérés dans l'approche OSI et plus particulièrement vis-à-vis des besoins exprimés, restreint cependant le nombre des techniques candidates. En effet, il est nécessaire dans le contexte OSI, que celles-ci, non seulement supportent les concepts de l'approche orientée-objets, mais encore s'intègrent parfaitement aux formalismes normalisés pour la spécification des ressources de gestion dans l'approche OSI.

Aussi, la recherche effectuée dans cette thèse se propose d'étendre le langage de description des objets gérés normalisé par l'ISO afin que ce langage supporte, d'une part la description formelle du

9. . Guidelines for the Definition of Managed Objects

comportement associé à cet objet de gestion, et supporte d'autre part la spécification formelle des interfaces associées à tout objet géré. Dans ce cadre, le langage GDMO va nous servir de référence. Le but fixé est d'étendre celui-ci afin qu'il supporte toutes les fonctionnalités requises énumérées ci-dessus.

Le choix d'une technique de description formelle pour permettre la spécification du comportement des ressources de gestion s'est rapidemment porté sur le langage CRS[10]. Les raisons de ce choix sont multiples. La principale est l'adéquation des mécanismes de spécification de la technique avec les normes OSI et le mode de pensée humain. En effet, la technique CRS initialement conçue pour la spécification des services et protocoles de communication, est basée sur le langage de description des données ASN.1 normalisé par l'ISO et utilise un mécanisme de règles pour la description du comportement, ce qui est proche du raisonnement humain.

Pour permettre l'intégration des concepts de CRS dans le langage de description des ressources de gestion, il a fallu dans un premier temps étendre la technique CRS avec des concepts objet et par la suite, intégrer ces mécanismes de spécification dans le langage normalisé par l'ISO avec une contrainte de compatibilité maximale.

Mettre en place un mécanisme de description formelle du comportement n'est pas une finalité en soi. Il faut que cette formalisation soit accompagnée d'un ensemble d'outils de développement qui en facilitent l'application et qui en exploitent les avantages. Aussi, allons nous dans ce mémoire étudier les différentes applications pouvant tirer profit de la spécification formelle du comportement. Dans ce cadre, nous allons principalement envisager une application basée sur une approche simulative des spécifications. Nous allons également entreprendre à ce niveau, la conception et la réalisation d'un environnement de développement intégrant les besoins des développeurs d'applications de gestion. Environnement qui intègrera les extensions proposées à la norme.

L'organisation du mémoire

Ce mémoire est organisé en trois parties distinctes: une étude du cadre général et des normes existantes, une proposition d'extension de ces normes ainsi que la présentation des applications qui peuvent tirer profit de ces extensions.

Première partie

Elle présente les différentes activités, standards et normes en vigueur dans le domaine de la gestion de réseaux. Le premier chapitre introduit les principaux concepts nécessaires à la compréhension du domaine d'activité. Le chapitre 2 est consacré à la présentation d'une approche simple (SNMP) et de son extension en cours de standardisation. Le chapitre 3 introduit les activités de normalisation au sein du modèle OSI. Le chapitre 4 contient une présentation détaillée du langage GDMO. Ce langage servira de base aux extensions proposées dans la seconde partie du document.

Seconde partie

Elle est consacrée aux extensions que nous proposons pour permettre une formalisation du comportement dans le modèle de description des ressources de gestion proposé par l'ISO. Elle est divisée en cinq chapitres distincts. Le premier concerne la définition du comportement des objets gérés. Il fixe l'ensemble des besoins de formalisation, comporte une étude détaillée des aspects liés au comportement dans GDMO et fixe le cahier des charges pour une formalisation éventuelle. Le chapitre 2 contient une

10. . Communicating Rule Systems

étude des différentes techniques de description formelle candidates à la formalisation du comportement des objets gérés dans le second chapitre. Le chapitre 3 comporte une présentation de la technique CRS ainsi qu'une description des motivations qui nous ont amenés à la sélectionner. Le chapitre 4 est consacré à l'extension de la technique CRS par des mécanismes d'héritage. Le chapitre 5 présente les extensions proposées au langage GDMO pour intégrer le formalisme basé sur CRS. Il présente notamment la manière dont le comportement formalisé en CRS étendu peut être inclu dans toute spécification GDMO en restant compatible avec la norme actuelle.

Troisième partie

Elle présente les applications basées sur l'extension du langage de description d'informations de gestion. Le premier chapitre est consacré à l'étude des différents environnements de développement d'applications de gestion existantes. Il présente également les réalisations entreprises sur notre environnement. Le second chapitre contient une description de l'approche simulative que nous avons mise en oeuvre pour une exploitation des spécifications de comportement à l'aide de l'environnement CRUSADE.

Conclusion générale

Les travaux présentés dans ce mémoire n'ont pas la prétention d'apporter la solution à tous les problèmes relatifs à la spécification des ressources gérées dans l'approche OSI. Aussi, présenterons nous dans la conclusion, un résumé des résultats obtenus, les limites de l'approche qui ont été identifiées à ce jour ainsi que les premières expériences acquises sur la mise en oeuvre de la technique. Les limites et les premières expériences serviront de base à l'identification de nouveaux axes de recherche à envisager dans ce domaine.

PARTIE I

Une introduction aux activités de gestion de réseaux

«Celui qui ne sait pas d'où il vient, ne sait pas où il va, car il ne sait pas où il est.»

Otto de Habsbourg

Cette partie a pour but de familiariser le lecteur avec les activités de gestion de réseaux et de faire le point sur l'état actuel des travaux de normalisation et de standardisation dans ce domaine.
Elle comporte une description générale du domaine, des besoins et des concepts, ainsi qu'une présentation des principales normes et standards en vigueur que sont les standards SNMP de l'IAB pour les réseaux TCP/IP et les normes OSI de l'ISO. Dans cette partie, une attention particulière sera portée à la description des travaux de normalisation de l'ISO car ils serviront de base aux extensions proposées dans la seconde et la troisième partie de ce document.

Chapitres

1. **Les modèles de gestion normalisés**
2. **Les approches SNMP et SNMPv2**
3. **La gestion des réseaux au sein du modèle OSI**
4. **Le langage GDMO**

1
Les modèles de gestion

Ce premier chapitre présente les concepts généraux relatifs à la gestion des réseaux. Après une brève introduction au problème, nous allons définir les différents champs d'application de la gestion ainsi que les différents modèles qui permettent la mise en place d'une gestion intégrée des réseaux hétérogènes.

1.1 Introduction

La majorité des systèmes de communication actuels ont une topologie relativement complexe. Ils sont le plus souvent composés de réseaux interconnectés (voir figure 1.1) qui utilisent des composants spécifiques. Actuellement, chaque constructeur propose une solution de gestion spécifique adaptée à son réseau. Ce type de système de gestion est le plus souvent incompatible avec d'autres sous-réseaux. Ceci entraîne bien souvent la nécessité de gérer chaque sous-réseau de manière indépendante.

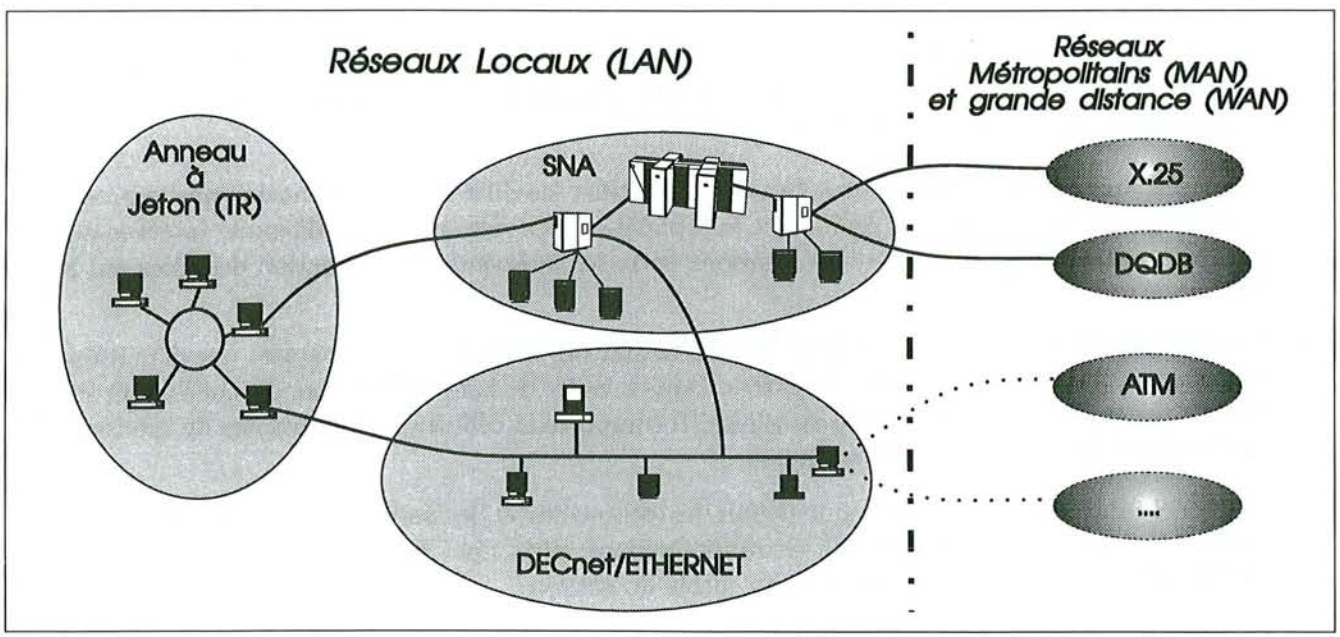

FIG. 1.1 - . *Exemple d'architecture d'un réseau hétérogène*

Un réseau de communication comporte à la fois des **ressources physiques** (systèmes hôtes, supports physiques de communication, répéteurs, routeurs, ponts, passerelles) ainsi qu'un certain nombre de **ressources logiques** (connexions entre systèmes hôtes, entités de protocole, etc ...).

Gérer un réseau revient à **observer** son activité (ex. collecte de statistiques sur le débit réel, calcul de taux d'erreurs), à **contrôler** les opérations en cours (ex. contrôle des accès, statuts des connexions) et à **agir** sur l'ensemble de ses ressources de communication (ex. activer, initialiser une station, un routeur).

La gestion des ressources est une activité critique vis-à-vis de la sûreté de fonctionnement d'un réseau. Elle doit être résistante aux pannes et rester opérationelle aussi longtemps que le système le supporte. De plus, le trafic engendré par l'échange d'informations de gestion sur un réseau doit rester minimal pour ne pas altérer les performances de ce dernier.

Une gestion intégrée d'un réseau hétérogène peut être définie comme la mise à disposition pour un opérateur d'un système unique permettant de manière uniforme de voir, d'accèder et d'influencer toute ressource du réseau qu'il gère. Ces ressources peuvent être de constructeurs différents. Le réseau géré peut résulter d'un ensemble de sous-réseaux hétérogènes basés sur des technologies différentes.

1.2 Les besoins de normalisation

Permettre une gestion intégrée d'un réseau hétérogène nécessite la normalisation des activités de gestion. Ces normes doivent fournir un modèle de gestion de réseau comprenant la définition du domaine d'application, la spécification des communications entre intervenants ainsi que la description formelle des informations échangées. La normalisation de cette activité permet au travers de produits de gestion conformes aux normes, d'une part une vision homogène de l'ensemble des ressources d'un réseau hétérogène, et d'autre part d'assurer la disponibilité d'un service unique d'accès à toute information de gestion indépendamment de la ressource qui la met à disposition.

L'activité de gestion de réseaux comporte quatre modèles complémentaires qui représentent les différentes parties intervenant dans sa réalisation [Hegering 93]. Ces modèles normalisés dans l'approche ISO [ISO-7498.4 89] sont:

1. le modèle **fonctionnel** dont le but est de définir les activités de base qui doivent être supportées par tout système de gestion ainsi que les fonctions qui permettront leur réalisation (ex. la fonction de gestion de charge sur un calculateur).

2. le modèle **organisationnel** dont le but est d'identifier les différents intervenants dans le processus de gestion. Ce modèle définit également la répartition des rôles entre les différents intervenants et spécifie la stratégie d'échange d'informations entre les intervenants en fonction des rôles qui leurs sont assignés.

3. le modèle de l'**information** qui a pour but de définir des mécanismes permettant une définition homogène des interfaces des ressources gérées. Dans le cadre de la normalisation, ce modèle fournit des méthodes de définition des ressources gérées. Il comporte la définition d'un langage de spécification de l'information associée à une ressource.

4. le modèle de **communication** qui définit les protocoles et les services spécifiques utilisés pour l'échange d'informations de gestion entre intervenants ainsi que l'architecture et les services sous-jacents nécessaires à la réalisation du protocole de gestion.

Ces différents modèles sont bien sûr liés entre eux. Le modèle de communication dépend d'abord des exigences du modèle organisationnel en terme de stratégie de collecte. Il dépend aussi du modèle de l'information pour la définition des paramètres du service ainsi que des fonctions d'accès aux ressources. La définition des interfaces pour les fonctions de gestion définies dans le modèle fonctionnel dépend du modèle de l'information car car c'est à partir des formalismes définis dans ce dernier que les fonctions sont modélisées. Nous allons décrire de manière plus détaillée ces différents modèles.

1.3 Le modèle fonctionnel

Afin de couvrir l'ensemble des besoins en gestion des réseaux, cinq domaines d'activités ont été identifiés et regroupés au sein de l'approche normalisée par l'ISO [ISO-7498.4 89]:

1. **Gestion des fautes**: les fautes se manifestent au sein d'un environnement de communication par des événements spécifiques (ex. erreurs de communication). La gestion des fautes comporte les activités de détection, d'isolation, d'identification et de recouvrement des erreurs susceptibles d'apparaître sur un réseau.

2. **Configuration des réseaux**: la gestion de la configuration d'un ensemble de réseaux gérés regroupe les activités de collecte et mise à disposition de l'information nécessaire à la gestion des interconnexions entre systèmes et réseaux. On peut donner à titre d'exemple l'identification de l'ensemble des stations actives sur un réseau de type Token-Ring, le contrôle du statut d'une connexion entre un réseau local de type ETHERNET et un réseau métropolitain de type DQDB [1].

3. **Tarification**: le besoin de gérer les coûts est apparu, d'une part avec les interconnexions de réseaux locaux au travers de réseaux publics (MAN, WAN), et d'autre part avec le besoin croissant d'optimiser l'utilisation des ressources au sein d'un même réseau local. Les fonctions associées à cette activité mettent à la disposition des exploitants de réseaux (mais aussi des utilisateurs), des systèmes d'information, de planification et de tarification des coûts d'un service. Ces fonctions facilitent l'optimisation de l'utilisation d'un réseau en autorisant par exemple des réservations de liens pour une qualité de service et une durée donnée.

4. **Performance**: les applications distribuées qui nécessitent des débits importants et une bonne qualité de service (ex. vidéo-conférences, transfert de fichiers) font appel aux dernières technologies de communication et requièrent le respect de nouveaux critères en termes de qualité de service. La garantie d'un débit minimal (pour le transfert d'images) ainsi que la fiabilité du transport (pour le transfert de fichiers) sont des critères de base qui doivent être pris en compte. Pouvoir assurer un débit constant et un taux d'erreur minimal à un utilisateur ou une application, nécessite la possibilité d'influencer directement les ressources de communication concernées. Les fonctions de collecte d'informations relatives à la performance ainsi que celles qui permettent d'influencer certains paramètres de façon à garantir une qualité de service à une application sont regroupées au sein de l'activité de gestion de la performance. Par exemple, le positionnement du paramètre *token-holding-time* et l'affectation de priorités pour un certain nombre de stations sur un réseau de type Token-Ring peuvent fixer une largeur de bande requise.

5. **Sécurité**: l'interconnexion de plus en plus fréquente de réseaux privés et publics a rendu nécessaire la mise en place d'une politique de contrôle très stricte garantissant la sécurité des ressources et des données accessibles. Cette activité comprend des fonctions de création, destruction et contrôle de mécanismes de sécurité (ex. gestion des comptes utilisateurs) ainsi que des fonctions de détection et de notification d'événements relatifs à la sécurité (ex. tentative de connexion d'une machine non répertoriée).

Si la liste des domaines d'applications semble aujourd'hui complète, celle de leurs fonctions n'est pas exhaustive. En effet, il incombera à chaque concepteur de système de gestion d'implanter, d'une part les fonctions génériques, et d'autre part de définir un certain nombre de fonctions spécifiques qui permettront de répondre aux besoins précis d'un utilisateur donné. Toute approche de la gestion de

1. . Dual Queue Dual Bus

réseau à pour but de fournir des fonctions qui permettent de couvrir les domaines décrits ci-dessus. Aussi, nous ne détaillerons pas dans les chapitres suivant, le modèle fonctionnel pour chaque approche. Nous présenterons brièvement en fin de chaque chapitre, les fonctions génériques disponibles dans l'approche.

1.4 Le modèle organisationnel

Pour mettre en place une stratégie de gestion d'un réseau hétérogène, il faut successivement identifier les différents intervenants, définir leurs rôles respectifs et définir les mécanismes de collecte d'informations de gestion. Ces différents aspects forment le modèle organisationnel.

1.4.1 Le modèle Gestionnaire-Agent

Les travaux entrepris sur l'identification des intervenants et sur la répartion des rôles ont abouti au modèle "Gestionnaire-Agent" utilisé dans toutes les approches existantes (figure 1.2.). Ce modèle définit les rôles pris par deux systèmes de communication au sein d'une activité de gestion. Un **Gestionnaire** est par définition "une partie d'une application distribuée qui est responsable d'une ou de plusieurs activités de gestion et qui, pour ce faire, initialise des opérations de gestion et reçoit des notifications". Un **Agent** est "la partie d'une application distribuée qui gère les objets gérés dans l'environnement de son système local. Il effectue des opérations de gestion sur les objets gérés à la suite d'opérations de gestion émanant d'un gestionnaire et émét des notifications envoyées par des objets gérés vers des gestionnaire" [CCITT.X.701 92].

Les systèmes de communication impliqués dans une activité de gestion peuvent être des stations de travail mais aussi tout autre type d'entité physique d'un réseau susceptible de supporter une fonction de gestion (ex. routeurs, PABX[2], concentrateurs). En effet, seule une entité physique peut supporter une fonction de gestion car toute fonction nécessite la présence d'un sous-système de communication et d'un processus de gestion. Le sous-système de communication représente les fonctionalités offertes par les ressources de communication à l'activité de gestion.

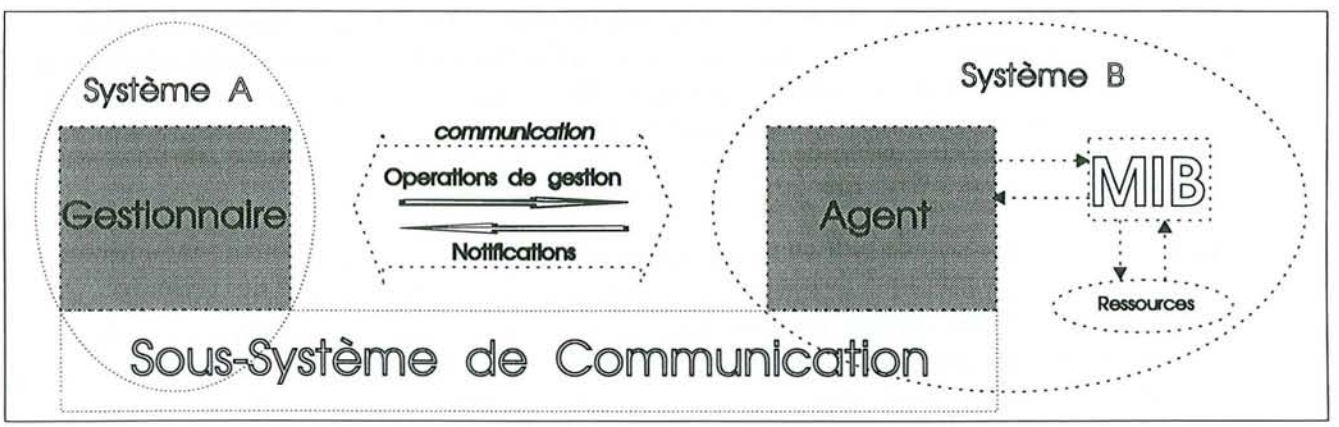

FIG. 1.2 - . *Illustration du modèle "Gestionnaire-Agent"*

L'exécution d'une activité de gestion est typiquement une activité de type client-serveur et implique donc toujours au moins deux utilisateurs de service situés dans des systèmes qui supportent un sous-système de communication complet (jusqu'au niveau applicatif). Un système prend le rôle de gestionnaire (client) et l'autre celui d'agent d'éxécution (serveur). Ces deux intervenants sont localisés, soit sur un

2. . Private Automatic Branch eXchange

même site, soit sur deux sites différents. Ils communiquent toujours au travers d'un sous-système de communication en utilisant un service de communication spécifique à l'activité de gestion. Seul l'agent dispose d'un accès direct aux ressources dont il a la responsabilité pour en extraire l'information de gestion. Le gestionnaire collecte l'information au travers de l'ensemble des agents du réseau et met cette information à disposition de l'ingénieur système ou d'éventuelles applications de gestion.

Deux stratégies de collecte d'informations de gestion existent entre le gestionnaire et l'agent de gestion. La première est basée sur un mécanisme de **polling**. Elle consiste pour le gestionnaire à demander à l'ensemble des agents du réseau, des informations sur l'état de leurs ressources de manière souvent cyclique. La seconde stratégie est basée sur un mécanisme d'**événements**. Dans ce cas, chaque agent envoie des informations au gestionnaire dès qu'un fait mérite d'être signalé au gestionnaire. Chaque méthode de collecte a ses avantages et ses inconvénients (voir Chapitre 2 2.1.1 et Chapitre 3 3.2). Le choix des méthodes est spécifique à une approche globale de la gestion. Elles seront détaillées dans les chapitres suivants.

1.4.2 Les domaines de gestion

L'administration de réseaux interconnectés implique bien souvent le besoin de partager les responsabilités de la gestion entre plusieurs organismes et plusieurs systèmes de gestion. Par exemple, un système de gestion qui gère deux réseaux locaux interconnectés par une ligne du réseau public ne pourra pas gérer le réseau public administré par un autre système et sous l'autorité d'un autre organisme.

Pour permettre la répartition des responsabilités de gestion sur un ou plusieurs réseaux, le concept de domaine de gestion a été défini. Un domaine représente une répartition du réseau en termes de systèmes gérés ou activités de gestion. La première répartition est la plus fréquente. Elle consiste à partager le réseau en domaines gérés chacun par un système de gestion spécifique. La seconde répartition est moins évidente mais fort intéressante. En effet, elle permet d'avoir plusieurs systèmes de gestion sur un réseau, chacun ayant une activité de gestion bien définie (comme par exemple la tarification pour l'un et la configuration pour l'autre). Dans ce cas, chaque système a une vue bien spécifique sur le réseau, vue restreinte aux informations significatives pour son activité. Le réseau est alors virtuellement partagé en domaines d'activités.

La présence de plusieurs systèmes de gestion sur un réseau donné nécessite la mise en place d'un mécanisme de coordination entre ces systèmes ainsi que la disponibilité d'un service de communication. Les schémas de coopération entre systèmes de gestion forment actuellement l'un des principaux axes de recherche dans l'administration des réseaux hétérogènes.

1.4.3 La gestion-système et la gestion de couche

Au sein d'un système de communication, deux types de gestion des ressources sont envisageables (voir figure 1.3). Un système peut être géré de manière globale par un processus utilisateur appelé SMAP[3]. Ce processus a une vue sur l'ensemble des ressources du système sur lequel il est implanté. Il peut avoir un rôle de gestionnaire ou d'agent et communique avec d'autres gestionnaires de systèmes au travers d'une entité de protocole de la couche application du sous-système de communication SMAE[4].

La **gestion-système** (SM[5]) permet la mise à disposition de l'ensemble des fonctions de gestion pour un système de communication donné. En contrepartie, une application de gestion de système nécessite la présence au sein du système de l'ensemble des facilités de communication offertes par une pile de

3. . System Management Application Process
4. . System Management Application Entity
5. . System Management

protocoles. Ceci concerne notamment le service d'échange d'informations de gestion qui est un service offert par la couche application du système de communication sous-jacent.

La deuxième façon de gérer un système de communication est de le faire au travers de **gestionnaires de couches** (LME[6]). Ces gestionnaires ont une vue limitée des ressources du système. Un gestionnaire de couche gère les ressources de sa couche (ex. entités de protocoles de transport). Deux gestionnaires de couches communiquent entre eux, soit au travers de protocoles de gestion spécifiques à leur couche, soit en utilisant les protocoles standards de la couche qu'ils gèrent (LE[7]).

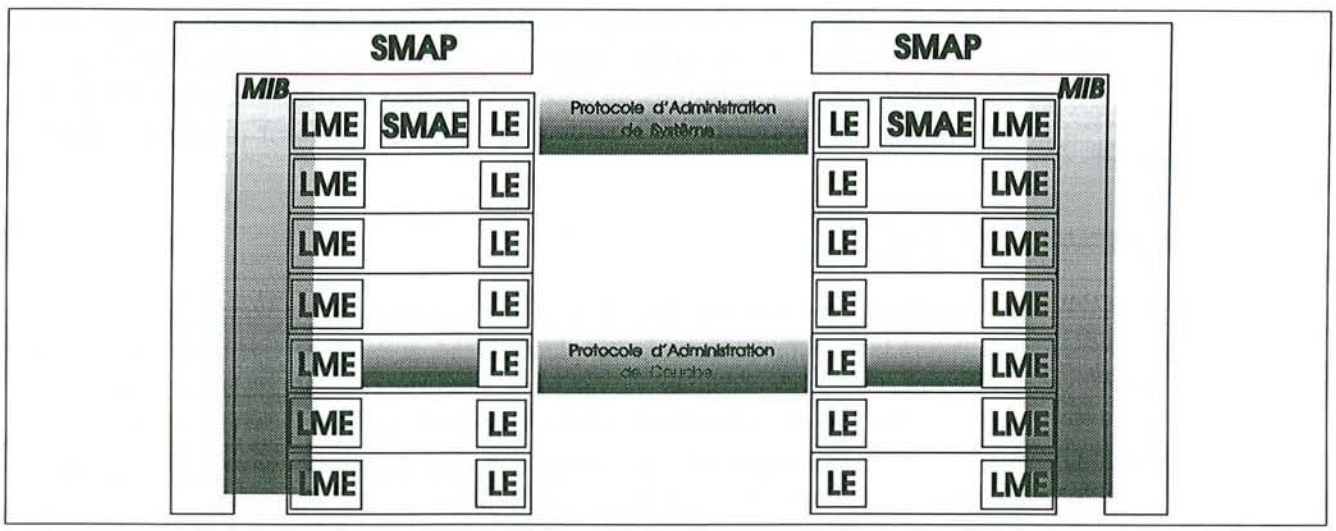

FIG. 1.3 - . *Gestionnaire de système et gestionnaire de couche*

La possibilité de gérer des systèmes à l'aide de gestionnaires de couches est particulièrement intéressante pour la gestion de systèmes tels que des routeurs ou des ponts. En effet, la plupart du temps ces éléments ne contiennent pas la totalité d'une pile de protocoles. Dans ces conditions ils ne peuvent pas supporter un processus de gestion-système.

1.5 Le modèle de l'information

Au sein d'un système de communication, chaque ressource est modélisée par un ensemble de données qui en reflète l'état. Cette modélisation, si elle est basée sur une approche unique, permet de représenter de manière homogène l'état de toute ressource du réseau. La représentation homogène facilite la définition d'un service de communication générique permettant d'accéder à cette information. Pour que les ressources soient modélisées de façon homogène, il faut mettre à la disposition des fournisseurs de ressources de communication un cadre formel pour la spécification des informations de gestion relatives à leurs ressources.

Le but du modèle de l'information est justement de fournir un tel cadre. Avec le modèle de communication, il constitue l'un des points les plus importants des normes relatives à une approche de la gestion de réseaux. Il définit la structure générale de toute information de gestion ainsi que la méthode de description de l'état d'une ressource. Pour cela, les mécanismes de description formelle doivent être utilisés.

6. . Layer Management Entity
7. . Layer Entities

1.5.1 La base d'informations de gestion

Tout agent de gestion maintient au sein d'une base d'informations de gestion (MIB[8]) l'ensemble des données nécessaires aux activités de gestion (figure 1.4). Cette base est définie comme un ensemble structuré d'objets gérés (MO[9]).

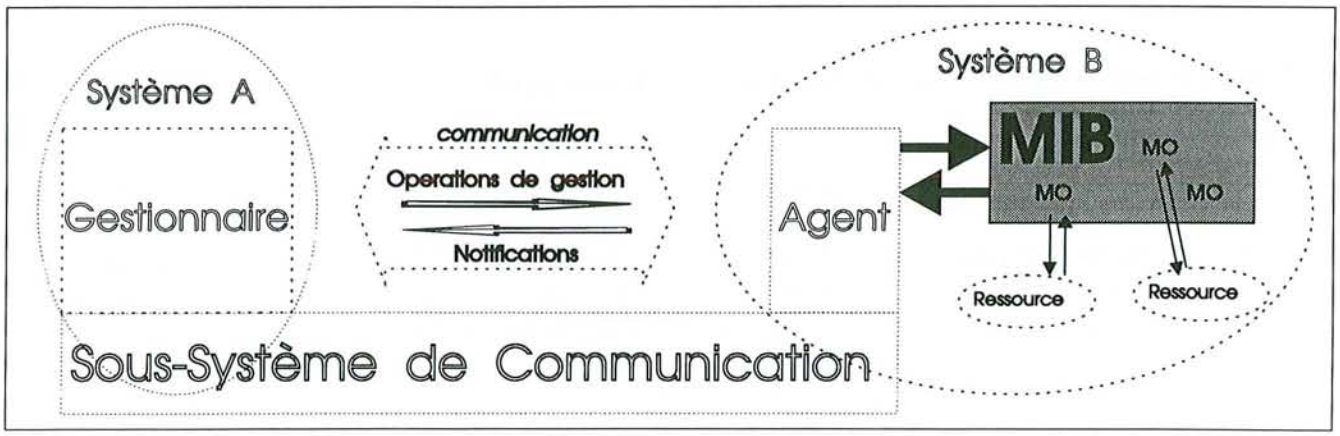

FIG. 1.4 - . *Localisation de l'information dans l'architecture d'un système de gestion de réseau*

Un objet géré est vu comme l'abstraction d'une ressource pour les besoins de sa gestion. En fait, un objet géré définit une interface qui fournit au système de gestion une vue normalisée de la ressource. Cette vue se définit comme un certain nombre de données qui représentent à tout moment l'état de la ressource et un certain nombre d'opérations autorisées au travers de cette interface. Ces opérations spécifient les actions autorisées sur la ressource en vue de sa gestion.

Un système de gestion n'agit donc pas directement sur une ressource mais sur les valeurs des données associées à celle-ci dans la base d'informations de gestion. L'affectation de ces valeurs par le gestionnaire aura des répercussions sur l'état réel de la ressource modélisée. De même, un changement d'état d'une ressource réelle au sein d'un système se traduit toujours par un changement de valeurs des données correspondantes au sein de la MIB.

1.5.2 La spécification des objets gérés

Afin de permettre à tout système de gestion d'accéder à toute ressource du réseau de manière uniforme, il est nécessaire que l'objet géré qui la modélise soit connu du système et que les données ainsi que les fonctions d'accès à l'information contenue dans l'objet soient normalisées. Dans ce but, il est nécessaire que toute approche fournisse un modèle générique pour la description des ressources de gestion. Ce modèle générique doit être complété par un langage formel de spécification des objets gérés.

Les modèles normalisés dans ce domaine convergent sur les points suivants:

- tout objet géré modélise une ressource au travers d'un ou plusieurs attributs. Ces attributs reflètent à tout moment l'état de la ressource associée.
- les attributs d'un objet géré ne sont accessibles que par des opérations de gestion normalisées.
- toute opération de gestion sur une ressource gérée se fait obligatoirement au travers des attributs de l'objet géré qui la modélise.

8. . Management Information Base
9. . Managed Object

Les éléments du langage de description des objets gérés sont spécifiques aux concepts retenus dans les différentes approches existantes. Le langage de description peut être très simple dans le cas où les objets gérés sont de type primitif (entiers, booléens) et les opérations limitées aux opérations de base (lecture, affectation). Le langage peut également être plus compliqué si le modèle de description des ressources intègre des concepts plus puissants. Ceux-ci devraient permettre la définition d'objets structurés qui encapsulent les attributs ainsi que la spécification d'opérations complexes sur la ressource.

1.5.3 Le nommage et l'identification des objets gérés

Réaliser une opération de gestion sur un système nécessite la connaissance des informations existantes dans la MIB du système géré mais aussi l'autorisation d'accès aux informations qui modélisent ses ressources. Afin de permettre une gestion intégrée des systèmes hétérogènes, il a été défini un mécanisme d'enregistrement unique permettant l'identification de chaque classe d'objet géré présent dans une MIB. Chaque approche fournit son propre mode d'identification des ressources. Les différents modes seront présentés en même temps que les approches normalisées existantes dans les chapitres 2 et 3 de cette partie.

1.6 Le modèle de communication

La gestion de réseaux est une application distribuée par définition. Aussi, celle-ci nécessite un support de communication qui permet l'échange d'informations de gestion entre acteurs de gestion (voir figure 1.5). Ce support de communication doit fournir un ensemble de services normalisés aux acteurs qui sont en l'occurence le ou les gestionnaires, ainsi que les agents de gestion. Cet ensemble de services doit correspondre aux besoins de ces systèmes.

Le gestionnaire dispose d'un ensemble d'opérations de gestion pour interagir avec un ou plusieurs de ses agents. Ces opérations lui permettent, d'une part d'accéder aux données d'une ressource, de les lire, de les changer, et d'autre part de créer ou détruire des données relatives à une ressource gérée chez un agent de gestion.

L'agent, quant à lui, ne dispose que de deux fonctions lui permettant de communiquer avec son gestionnaire. Il peut soit répondre aux opérations invoquées par le gestionnaire, soit émettre des notifications pour le prévenir de l'occurence d'événements spéciaux au sein de son système.

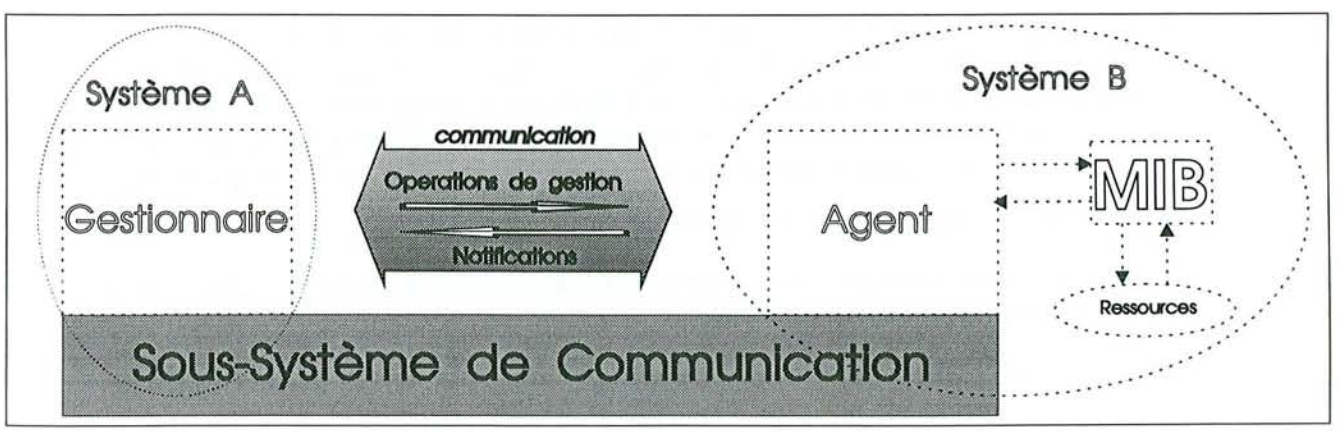

FIG. 1.5 - . *Le sous-système de communication*

Le support de communication qui fournit les services nécessaires aux activités de gestion doit donc

impérativement proposer les deux services suivants:

- accès et manipulation des données de gestion dans une base d'informations. Ce service permet au gestionnaire d'effectuer par le réseau des opérations de lecture d'attributs, d'affectation, de création ou de destruction d'objets gérés, de déclenchement d'action dans une base d'informations de gestion distante. Il permet également de collecter des événements émis par une base.
- délivrance de notifications émises par un agent de gestion vers un gestionnaire pour l'informer d'un changement relatif à une ou plusieurs des ressources supervisées par cet agent.

Un service de communication qui permet l'échange d'informations de gestion entre gestionnaires est également requis dans toute approche de gestion de réseaux. Ce type de service est nécessaire si plusieurs gestionnaires, dont les activités doivent être coordonnées dans leurs domaines respectifs, sont présents sur le réseau. La gestion étant une activité sensible, le service doit également prévoir des mécanismes de sécurité tels que l'identification du gestionnaire et le cryptage des données transmises.

L'activité de gestion a pour but de permettre une utilisation optimale du réseau géré. Aussi, le trafic engendré par celle-ci ne doit pas nuire aux autres transferts sur le réseau. Le sous-système de communication fournissant un service de transfert d'informations de gestion se voit donc imposer la contrainte d'être peu gourmand en ressources de communication (taille et nombre de transferts).

Le service de communication qui fournit l'ensemble de ces fonctionnalités est spécifique à l'approche générale retenue pour la gestion et fortement dépendant de la modélisation des ressources et des opérations sur celle-ci. Nous détaillerons donc ces services dans les chapitres suivants en présentant les approches existantes.

1.7 Résumé

La gestion de réseaux de plus en plus grands, complexes et hétérogènes nécessite des systèmes intégrés et évolutifs. Or, le préalable au développement de systèmes de gestion supportant une telle hétérogénéité est la normalisation des activités de gestion.

Normaliser ces activités implique la définition de quatre modèles. Il s'agit du modèle modèle fonctionnel qui identifie les activités génériques et qui définit pour chaque activité, les fonctions nécessaires à sa réalisation, du organisationnel qui définit les rôles des différents intervenants du réseau, du modèle de l'information qui permet une description homogène des ressources gérées et finalement du modèle de communication qui spécifie les services de communication offerts pour l'échange d'informations de gestion entre les intervenants.

Ces modèles, bien que différents par les aspects traités, sont étroitement liés. Aussi, toute approche de normalisation de la gestion de réseau se doit de fournir une solution globale comportant une définition homogène de chacun de ces modèles.

Les deux approches normalisées à ce jour (OSI, IAB) intègrent à plus ou moins grande échelle ces différents concepts. Nous allons dans les chapitres suivants décrire chacune d'entre elles de manière détaillée, en présentant leurs implantations dans les différents modèles introduits dans ce chapitre.

2
Les approches SNMP et SNMPv2

Première approche standardisée présentée dans le cadre des approches existantes, SNMP définit les différents modèles pour la gestion de réseaux hétérogènes au sein de la communauté Internet. Ce chapitre présente les principales caratéristiques de l'approche SNMP et de sa nouvelle évolution appelée SNMPv2 ou SMP [1].

La première partie de ce chapitre est consacrée à la présentation de la version initiale de l'approche SNMP tandis que les extensions fournies par SNMPv2 figurent dans la seconde partie. Un résumé des concepts ainsi qu'une présentation de ses limites serviront de conclusion à ce chapitre.

2.1 L'approche SNMP

Standardisée en 1988 par l'IAB, l'approche SNMP [Case 88, Case 90] propose une solution intégrée au problème de la gestion des réseaux hétérogènes. Cette approche est basée sur un ensemble de recommandations qui définissent le modèle organisationnel, la structure de l'information de gestion, l'architecture des bases d'informations de gestion et le protocole de communication utilisé pour le transfert d'informations.

Nous allons dans cette section présenter les concepts retenus dans la version initiale de l'approche SNMP pour la représentation des quatre modèles nécessaires à la mise en place d'un environnement de gestion intégrée.

2.1.1 Le modèle organisationnel

Le modèle organisationnel [Davin 92] a pour but d'identifier les différents acteurs impliqués dans la gestion du réseau et de leur affecter des rôles particuliers. Il définit également les stratégies de collecte d'informations.

La répartition des rôles

L'approche SNMP a retenu le modèle **"Gestionnaire-Agent"** tel qu'il a été présenté dans le chapitre précédent. Un agent est ici composé d'une interface de service SNMP, d'un processus de gestion et d'une base d'informations de gestion dont le but est de refléter à tout moment l'état des ressources gérées.

Le modèle prévoit également la possibilité de gérer des réseaux d'architecture différente de celle de TCP/IP. Ceci est possible grâce au concept d'agent de proximité (Proxy-Agent) [Case 90]. La base

1 . Simple Management Protocol

d'informations de cet agent reflète l'état du réseau externe. Son processus applicatif a pour tâche de faire le lien entre les activités de gestion SNMP et les activités spécifiques au réseau externe.

A tout système de gestion est associé un identificateur de communauté qui permet la mise en place de domaines de gestion (permet un partage du réseau entre différents gestionnaires). Cependant, aucun mécanisme n'est prévu dans l'approche SNMP pour la mise en place de structures hiérarchiques de systèmes de gestion.

Les mécanismes d'accès à l'information

Nous avons vu dans le chapitre précédent que deux mécanismes de collecte étaient envisageables pour accéder à l'information de gestion: le polling et la notification d'événements.

En théorie, l'approche SNMP intègre les deux. En effet, elle comporte des mécanismes de requête d'informations du gestionnaire vers l'agent et un service de notification d'événements de l'agent vers le gestionnaire. Le problème lié à l'utilisation des notifications est qu'il n'y a aucune garantie de délivrance de celles-ci. Il faut donc utiliser ce mécanisme avec une certaine prudence.

Cette approche donne à l'agent un rôle passif vis-à-vis du gestionnaire qui supporte toute l'activité. L'agent ne fait que répondre aux requêtes du gestionnaire qui doit, pour s'informer de l'état de ses ressources, interroger régulièrement ses agents de gestion. Cette approche a l'avantage d'être relativement simple à implanter. Elle consomme peu de ressources de calcul, mais génère beaucoup de trafic de gestion sur le réseau, ce qui est un désavantage important.

2.1.2 Le modèle de l'information

Le modèle de l'information a pour but de définir la structure de l'information de gestion et de fournir des mécanismes d'identification des objets gérés. La définition de la structure de l'information de gestion comprend le choix d'une approche pour la description des données et la spécification d'un langage de description des objets gérés.

La description des objets gérés

Dans l'approche SNMP, chaque ressource d'un système est modélisée par un objet géré [Rose 90], représenté par une variable dans la MIB de l'agent. Cette variable est soit de type simple, soit structurée. Le type de la variable est exprimé à partir d'un sous-ensemble de la notation ASN.1[2] ([ISO-8824 90]). Ce choix de limitation des types des variables a été guidé par la volonté des concepteurs de SNMP de définir une approche de gestion simplifiée au maximum.

Afin de permettre l'utilisation d'opérations génériques sur ces objets et dans le but de faciliter l'interopérabilité entre systèmes de gestion, tout objet géré doit être décrit de manière uniforme à l'aide d'une notation standardisée expliquée ci-dessous.

Le langage de description d'objet géré

Dans la structure de l'information de gestion de l'approche SNMP, le langage de description des objets gérés est basé sur une macro ASN.1 pour la définition des objets et sur un sous-ensemble de types de la notation ASN.1 pour la description des types [Rose 91, Rose 90]. Le résultat de cette macro ASN.1 (la syntaxe associée) est présenté dans le modèle de la figure 2.1. Il correspond à une notation valuée

2. . Abstract Syntax Notation number 1

associant à chaque nom d'objet, un type ASN.1, des droits d'accès, un statut de présence, ainsi qu'un identificateur d'objet permettant l'enregistrement de l'objet dans l'arbre d'identification.

```
<object-name> OBJECT-TYPE
 SYNTAX <asn-type-reference>
 ACCESS access-rights
 STATUS presence-requirements
[ DESCRIPTION <string>]
[ REFERENCE <display-string>]
[ INDEX  { index-type [,index-type]*  } ]
[ DEFVAL <default-value>]
::= <object-identifier>

access-rights ->  read-only
                | write-only
                | read-write
                | not-accessible

presence-requirements ->  mandatory
                        | optional
                        | obsolete
                        | deprecated

index-type  ->   value (indexObject ObjectName)
               | type (indextype)
```

FIG. 2.1 - . *Expansion de la macro ASN.1 pour la spécification des objets gérés*

Une spécification d'objet contient le nom de celui-ci (<object-name>) suivi du mot-clé **OBJECT-TYPE**. Elle comprend également des champs de définition et l'affectation d'un identificateur d'objet.

Les champs obligatoires

Le champ SYNTAX contient la définition du type ASN.1 de l'objet géré. Les types ASN.1 supportés par SNMP se limitent à un sous ensemble des types normalisés. La section suivante énumère les types supportés dans SNMP. Le champ **ACCESS** définit les droits d'accès à l'objet (lecture, écriture) et **STATUS** définit les contraintes de présence de l'objet dans la base d'informations de gestion.

Les champs optionnels

Le champ **DESCRIPTION** contient une chaîne de caractères, définition textuelle de l'objet géré. Le champ **REFERENCE** contient une chaîne de caratères de type chaîne affichable dans laquelle des références à d'autres objets gérés précédemment définis peuvent être faites. Le champ **INDEX** est présent dans la définition d'un objet si et seulement si celui-ci correspond à une entrée dans une table. Ce champ permet de définir les mots clés nécessaires à l'identification des instances de cet objet. Le champ **DEFVAL** est utilisé pour la définition d'une valeur par défaut assignée à l'objet lors de sa création.

Le sous-ensemble d'ASN.1 supporté

Une des principales caractéristiques de la modélisation des ressources au sein de l'approche SNMP est le choix de la simplicité des objets. Aussi, seuls quelques types de base sont autorisés dans l'approche. Il en est de même pour les types structurés. La liste ci-dessous énumère l'ensemble de ces types autorisés.

- types simples: **INTEGER** (type entier pouvant être énuméré ou restreint), **OCTET STRING** (chaîne de caractères), **OBJECT IDENTIFIER** (identificateur d'objet), **NULL** (utilisé pour modéliser l'absence d'un champ dans une structure);
- types prédéfinis: **Opaque** (type universel permettant la représentation de tout type d'informations sous forme de chaîne d'octets), **TimeTicks** (unité temporelle basée sur un entier non négatif), **Gauge** (type entier utilisé notamment pour la définition de variables représentant des taux d'erreurs), **Counter** (compteurs), **IpAddress** (format prédéfini pour les adresses Internet), **NetworkAddress** (format des adresses du niveau réseau), **ObjectSyntax** (syntaxe associée à un objet de gestion);
- types structurés: **SEQUENCE** (permet la composition de structures), **SEQUENCE OF** (permet la composition de listes ordonnées).

Les types prédéfinis sont tous composés à partir des types simples autorisés. L'encodage des données transmises se fait à partir des règles d'encodage de base d'ASN.1 [ISO-8825 87]. L'information associée à un objet est donc soit de type simple, soit de type enregistrement (**SEQUENCE**) ou encore tableau (**SEQUENCE OF**).

Le nommage et l'identification des ressources

Pour être accessible à travers une invocation du protocole de gestion, tout objet géré doit être référencé dans l'arbre d'identification. Dans l'approche SNMP, à tout objet géré est associé un identificateur d'objet qui représente le chemin dans l'arbre d'identification au bout duquel l'objet est référencé.

Le nom associé à un objet est défini par la chaîne de caractères utilisée pour sa définition. Ce nom a une portée locale à la base d'informations de gestion et donc à l'agent.

L'organisation des bases d'informations de gestion

L'approche SNMP ne propose pas de mécanismes de hiérarchisation d'une base d'informations de gestion. Toute base d'informations est donc constituée d'une suite de variables qui sont soit de type simple soit définies comme des tables. Le seul élément d'organisation d'une base d'informations est l'ordre lexicographique de ses composants (nom de l'objet géré dans la base).

2.1.3 Le service de communication

Cette partie donne une description sommaire du service de communication offert par SNMP. Une description détaillée est donnée dans [Rose 89].

Comme pour la modélisation des ressources, la principale préoccupation des concepteurs de SNMP a été de définir un protocole simple (d'où le nom de l'approche) permettant néanmoins de couvrir l'ensemble des besoins en communication des applications de gestion.

Deux raisons ont guidé ce choix de simplicité. Tout d'abord la conception d'un protocole simple facilite les étapes de développement et d'implantation, lui assurant ainsi une grande diffusion. De plus, la simplicité des messages échangés en petit nombre, à de faibles fréquences, permet aux paquets de gestion qui circulent au travers du réseau de ne pas trop altérer son trafic.

Les services suivants sont proposés par SNMP:

1. Accès aux informations d'une MIB d'un agent (services GET et GET-NEXT),
2. Affectation de variables dans des objets de gestion d'un agent (service SET),
3. Notification d'événements (service TRAP).

Le service GET

Le service GET est un service confirmé qui permet à un utilisateur du service SNMP se trouvant dans un rôle de Gestionnaire d'accéder aux valeurs des instances d'objets stockées dans la MIB d'un agent. Les paramètres contiennent un identificateur de requête et la liste des variables dont la valeur est demandée. La réponse comporte, tout comme la requète, l'identificateur de requête ainsi qu'une liste de couples (identificateur d'instance d'objet, valeur associée).

Le service GET_NEXT

L'opérateur GET-NEXT permet le parcours complet d'un sous-arbre d'une MIB. Pour chaque invocation du service GET-NEXT, le gestionnaire reçoit une réponse GET-rsp contenant le couple (identificateur, valeur) associé à l'instance qui suit l'objet spécifié dans la requête. Le choix de l'objet suivant se fait d'après l'ordre lexicographique dans la vue de la communauté.

Les opérateurs GET et GET-NEXT sont complémentaires. En effet, un système de gestion ne connait pas nécessairement l'ensemble des objets implémentés ni la taille des tables. Dans ces cas, l'utilisation de l'opérateur GET-NEXT autorise le parcours séquentiel de la MIB et donc de tous les objets et toutes le colonnes de tables.

Le service SET

L'invocation d'une primitive de service SET permet à un gestionnaire d'affecter des valeurs aux instances de variables de la MIB d'un agent. Les paramètres d'une requête SET comprennent un identificateur de requête ainsi qu'une liste de couples (identificateur d'instance d'object, valeur associée). Ce service est confirmé par une réponse de type GET-rsp.

La figure 2.2 illustre une invocation de service SET sur un attribut d'une MIB d'agent. Le gestionnaire invoque une opération d'affectation ayant un identificateur donné (dans l'exemple l'identificateur porte la valeur *122*) et affecte au travers de cette opération la valeur *3* à la variable *a* de la MIB de l'agent. L'agent confirme cette affectation par envoi d'une réponse de type GET ayant même identificateur que l'invocation SET. Cette confirmation GET comporte en paramètre: la variable affectée ainsi que sa nouvelle valeur. Il est possible d'affecter au travers d'une opération SET plusieurs variables d'une même MIB.

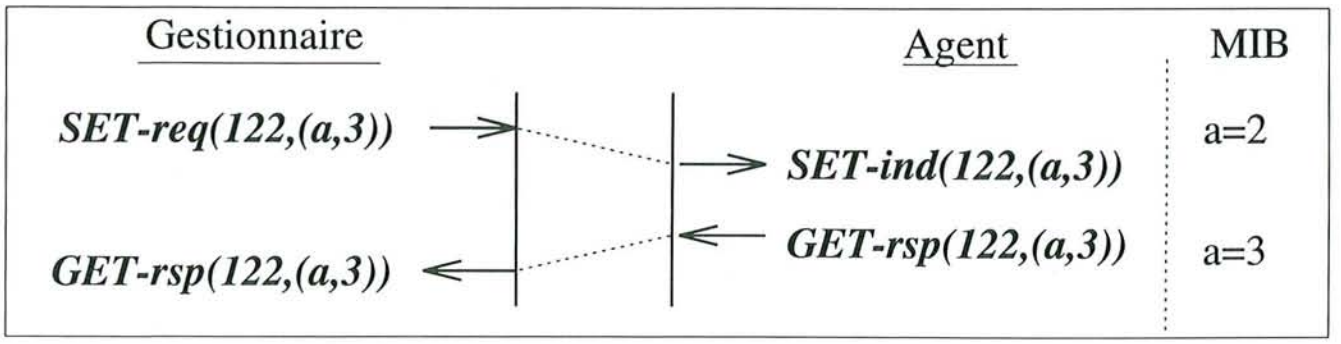

FIG. 2.2 - . *La fonction SET de l'approche SNMP*

Le service TRAP

Afin de permettre à des agents d'informer un système de gestion de troubles éventuels sur l'une de ses ressources, un mécanisme de notification appelé TRAP a été intégré dans le protocole SNMP. Lors de l'occurence d'un événement exceptionnel comme par exemple la rupture d'une liaison, l'agent envoie au système de gestion une requête TRAP contenant un message identifiant l'événement. Ce mécanisme est présent dans la norme. Mais comme les notifications ne sont pas confirmées dans SNMP et qu'il n'y a pas garantie de délivrance du message (voir 2.1.6), le mécanisme de notification est à manipuler avec précautions.

Il est néanmoins fortement utilisé dans la gestion des réseaux locaux où la perte de messages est pratiquement nulle. Ce n'est que lorsque le sous-réseau n'est plus fiable que l'utilisation des notifications de SNMP devient hasardeuse.

2.1.4 Le protocole de communication

Le protocole de communication SNMP est de conception simple et basé sur la pile UDP/IP. La figure 2.3 présente le cheminement des données dans une telle entité de protocole. Pour expliquer rapidement ses fonctions, nous allons différencier le mode d'émission de celui de réception.

Dans le mode d'émission (cheminement *1,2,3,4,5,6,7*), l'entité de protocole envoie chaque invocation au service d'authentification (non défini dans le standard). Si le message peut être émis, il est alors étendu avec l'identificateur de communauté et la version du standard utilisé. Ce message est encodé et envoyé sur le service de transport sous-jacent.

En mode réception (cheminement *8,9,10,11,12,13,14*), l'entité de protocole, décode tout message qui lui est destiné, vérifie sa consistance et la communauté d'origine. Si la communauté est dans l'ensemble des communautés autorisées alors le message est donné au service d'authentification, sinon une notification d'erreur est émise. Après traitement, le service d'authentification renvoie le message à l'entité de protocole qui le délivre à l'utilisateur. Avant l'exécution de la requète, une vérification des droits d'accès aux variables de la MIB (vérification des droits de lecture/écriture en fonction de la communauté) est effectuée.

Le service d'authentification est prévu pour la mise en place de mécanismes de sécurité plus puissants que l'identificateur de communauté. Comme ce service ne fait pas partie intégrante du standard SNMP, il n'est pratiquement jamais implanté dans les plates-formes de gestion ni dans les agents.

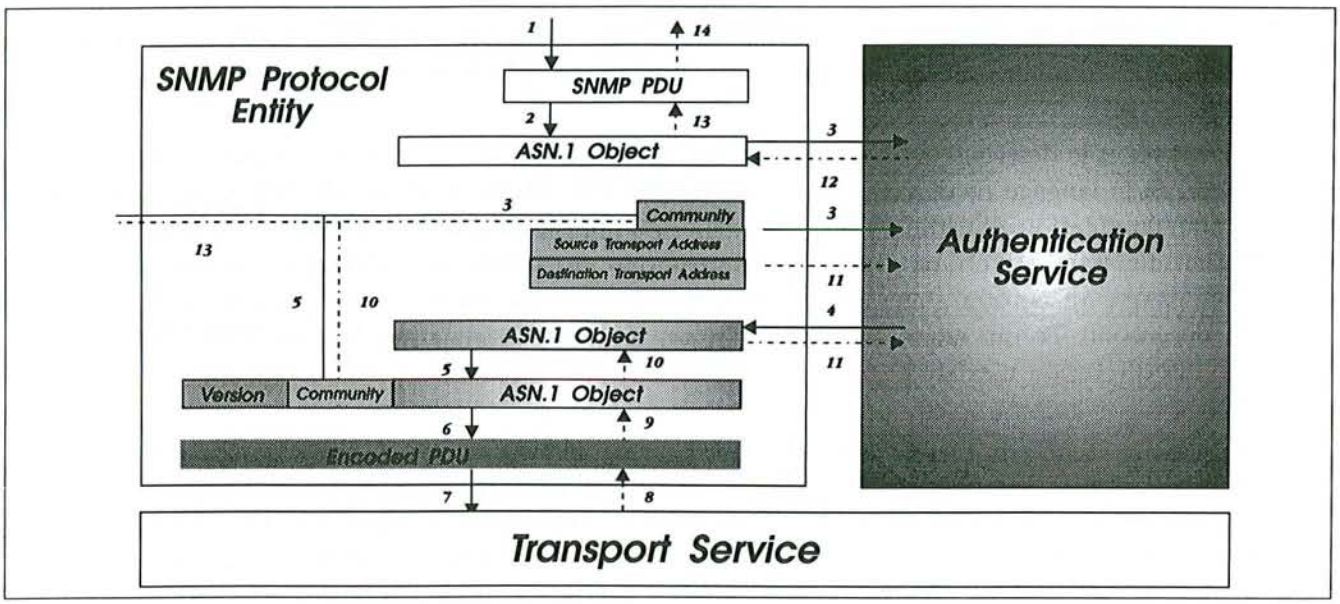

FIG. 2.3 - . *L'entité de protocole SNMP*

2.1.5 Les fonctions génériques

On ne peut dans l'approche SNMP, pas réellement parler de l'existence de fonctions génériques, au sens de la réutilisabilité. Il existe dans l'approche un certain nombre de bases d'informations de gestion pour la gestion de la plupart des composants de réseaux existants. Les plus connues sont les bases MIB-I [McCloghrie 90] et MIB-II [McCloghrie 91] qui contiennent la description de l'ensemble des objets nécessaires à la gestion d'une pile TCP/IP. Ces bases fournissent des fonctions spécifiques à la gestion des composant pour lesquels elles sont conçues.

Seule une MIB standardisée par l'IAB fournit des fonctions génériques. C'est la MIB RMON[3] [Waldbusser 91] utilisée pour la gestion distante de sondes sur un réseau local.

2.1.6 Les limites de l'approche

L'approche SNMP est limitée dans bien des domaines. Nous allons dans les paragraphes suivants énumérer, pour chaque modèle, ses limites dans l'approche.

Le modèle organisationnel ne permet pas la structuration hiérarchique des systèmes de gestion. Ce concept est pourtant primordial dans la gestion de grands réseaux sur lesquels plusieurs systèmes sont implantés. Ce modèle ne comporte pas de concepts adaptés à la mise en place de mécanismes de sécurité fiables. En effet, les seuls mécanismes présents sont basés sur le contrôle des identificateurs de domaine et ne prennent en compte aucun mécanisme de cryptage. Dernière lacune, l'absence de stratégie d'informations basée sur des systèmes de notification d'événements sur le réseau entraîne forcément une augmentation non négligeable du trafic de gestion.

Le modèle de l'information ne comporte aucune méthode qui permet de modéliser de manière homogène différents réseaux. La représentation des ressources gérées sous forme de variables simples ou de tables n'est pas adaptée à l'hétérogénéité et à la complexité des réseaux. En effet, il est nécessaire

3. . Remote MONitoring management information base

pour bien modéliser un réseau d'avoir une approche plus puissante qui permet notamment de décrire en plus des états d'une ressource, des opérations complexes sur celle-ci. Or, ceci n'est pas possible dans l'approche SNMP.

La création et la destruction d'entrées dans des tables au sein d'une MIB ne sont pas normalisés dans l'approche. Si le langage de description des objet gérés prévoit la possibilité de définir des valeurs par défaut pour la création d'instances, ni le modèle de l'information, ni le protocole n'incluent la notion de création d'instances d'objets gérés. Si la destruction est obtenue par affectation de valeur, le choix d'interpréter une affectation d'une instance non existante comme une création est laissé au développeur d'agent de gestion, ce qui altère fortement l'interopérabilité. Notons que dans la MIB RMON, cette liberté d'interprétation n'existe pas et qu'une affectation sur une entrée de table non-existante équivaut à sa création.

La première lacune du service de communication est son manque de fiabilité. En effet, le protocole est basé sur le service de transport UDP [4] [Postel 80] qui ne garantit pas la délivrance des messages. Ceci entraîne l'impossibilité d'utiliser des stratégies de collecte d'informations basées sur des notifications non confirmées. La seconde lacune concerne le compte-rendu d'erreurs dans le cas d'une lecture ou affectation d'un ensemble de variables. Car si plusieurs erreurs sont faites (plus d'une instance d'objet géré non présente, non autorisée en lecture ou écriture), seule la première est identifiée dans la réponse. De plus, le concept de lecture ou écriture atomique n'est pas intégré dans le modèle, ce qui risque d'entraîner des inconsistances dans la MIB d'un agent lors d'une opération multiple.

Lire une table dans une MIB nécessite au moins autant d'échanges de primitives que d'éléments dans la table. L'opérateur GET-NEXT bien que très pratique ne fournit cependant pas d'alternative stable à la lecture en une fois d'un grand nombre de données afin de minimiser le trafic de gestion sur le réseau.

Aucune remarque ne peut être faite sur les fonctions génériques utiles à la mise en place d'une gestion intégrée. Ces fonctions n'existent pas dans l'approche SNMP et cela implique l'impossibilité de les réutiliser lors de la conception d'une nouvelle MIB. Ceci est principalement dû au modèle de l'information trop simple.

La simplicité du modèle a assuré une très grande diffusion à l'approche SNMP. Ce succès et la grande diffusion ont très vite permis l'identification des limites de l'approche. En effet, le modèle est utilisé dans des applications de gestion de plus en plus complexes qui mettent à jour les lacunes de ses modèles respectifs.

Conscients de ces lacunes, les concepteurs du modèle initial ont entrepris des recherches dans le but de fiabiliser et d'étendre celui-ci. Résultat de ces recherches, la version 2 de l'approche SNMP a vu le jour début 1993.

2.2 Les évolutions de SNMPv2

Conçue pour palier les lacunes de l'approche SNMP, SNMPv2 conserve pour des raisons de compatibilité, de nombreuses caractéristiques de la version initiale. Les extensions apportées par SNMPv2 portent principalement sur le modèle organisationnel et le modèle de communication [Case 93e]. Le modèle de l'information a été enrichi tandis que le modèle fonctionnel, absent dans la version 1 du standard, n'a toujours pas été considéré dans la nouvelle version. Nous allons dans cette section passer en revue ces extensions

4. . User Datagram Protocol

2.2.1 Le modèle organisationnel

Le modèle organisationnel de SNMPv2 [Galvin 93] étend le modèle initial basé sur la coopération entre un gestionnaire et plusieurs agents de gestion avec le concept de coopération entre plusieurs gestionnaires. L'introduction de ce concept permet la création d'une organisation hiérarchique de systèmes de gestion sur un réseau. Ce permet à différents systèmes de gestion esclaves d'opérer sous le contrôle d'un système central maître qui coordonne l'activité de gestion. Le concept de coopération entre gestionnaires est modélisé dans l'approche SNMPv2 par un mécanisme d'échange d'informations d'activité basé sur des notifications d'événements confirmées. Le modèle permet en plus de configurer de manière statique ou dynamique cette hiérarchie.

Une autre innovation présente dans la version 2 de l'approche SNMP, est la prise en compte des aspects de sécurité. En effet, le modèle organisationnel comporte la définition de systèmes de sécurité basés sur une version préalable de mécanismes de sécurité pour SNMP et intègre ceux-ci dans les différents autres modèles, notamment le modèle de communication.

2.2.2 Le modèle de l'information

Le modèle de l'information de l'approche SNMPv2 [Case 93d] reste fidèle aux concepts de l'approche initiale. Cependant, des efforts ont été accomplis sur la documentation des agents de gestion pour lesquels des mécanismes de description basés sur des macros ASN.1 sont désormais disponibles. De même, des mécanismes d'identification de version de MIB sont également prévus dans le modèle de l'information. Le langage de description des objets gérés n'a quant à lui subi que quelques modifications minimes et le sous-ensemble d'ASN.1 supporté pour le type des variables, a été étendu à de nouveaux types simples tels que COUNTER64 (entier sur 64 bits) ou BIT-STRING (chaîne de bits).

De plus, le système de création et de destruction d'objets gérés a été clarifié dans cette approche (étendu à toutes les MIB et non seulement la MIB RMON). Ces deux opérations ne sont applicables qu'à des instances de tables (seules des colonnes de tableaux peuvent être créées ou détruites).

Une des extensions les plus intéressantes du modèle de l'information concerne la mise à disposition d'un langage de spécification de notifications. Ce langage est, comme toutes les notations de l'approche, basé sur une macro ASN.1 et permet de définir, pour une notification donnée: son nom, un identificateur d'objet pour son enregistrement dans l'arbre d'identification, la liste des objets gérés présents dans la notification et une description textuelle de celle-ci.

En résumé, le modèle de l'information de SNMPv2 a été enrichi par des mécanismes qui permettent une meilleure documentation d'un couple (Agent de gestion , MIB).

2.2.3 Le modèle de communication

L'ensemble des primitives de services offertes par le service de communication de la version 1 est également supporté dans le modèle de la version 2 [Case 93c]. Ces primitives ont cependant subi quelques améliorations. La version 1 du modèle de communication comportait quelques imperfections au niveau du compte-rendu d'erreurs dans les affectations (SET) ou lectures multiples (GET, GET-NEXT). En effet, dans le cas où plusieurs erreurs dans les listes d'objets gérés étaient détectées par l'agent, seule la première était rapportée. Ces imperfections ont été gommées dans la version 2, où, d'une part l'ensemble des erreurs éventuellement identifiées sont rapportées dans la réponse, et d'autre part seul un type de réponse (RESPONSE) est défini dans la norme, évitant ainsi d'avoir pour chaque type de service confirmé deux primitives de réponses spécifiques.

Le concept de lecture et écriture atomique est explicitement intégré. Il est basé sur un mécanisme

de lecture ou écriture à deux phases. Dans la première phase, toutes les variables selectionnées sont bloquées. Si le bloquage a réussi, elles sont lues ou affectées, sinon la requête est rejetée.

Dans la section décrivant les limites de l'approche SNMP, nous avons évoqué le problème du trafic généré lors de la lecture d'un ensemble de valeurs d'objets gérés. Cette lacune est comblée dans la version 2 par un nouveau service de communication appelé GET-BULK. Ce service confirmé permet la collecte en une seule PDU[5] d'un grand nombre de données d'une MIB.

Le modèle de communication fournit aussi un service de communication entre systèmes de gestion. Ce service, appellé INFORM, permet l'échange en mode confirmé d'informations de gestion entre deux systèmes de gestion via des notifications d'événements accompagnées d'une estampille. Le fait que ce service soit confirmé permet aux applications de garantir la délivrance des notifications ce qui n'était pas possible dans la version 1. Cependant, ce service n'est disponible que pour la communication entre systèmes de gestion et non entre agents et gestionnaires.

L'apport principal de ce nouveau service de communication est une réduction du trafic de gestion sur le réseau (au travers du service GET-BULK), une augmentation de la vitesse d'accès à l'information pour un système de gestion s'il est utilisé avec modération, et une clarification de la sémantique des primitives de service (création, destruction d'instances).

2.2.4 Les fonctions génériques

La notion de fonctions génériques comme définie dans le modèle fonctionnel du premier chapitre (1.3) est toujours absente dans la version 2 de SNMP. Seules de nouvelles bases d'informations de gestion relatives aux extensions sont introduites. Ces bases sont:

- MIB gestionnaire-gestionnaire [Case 93b]: comporte l'ensemble des objets gérés qui permettent, d'une part de configurer les rôles des différents intervenants, et d'autre part de définir les conditions dans lesquelles des échanges d'informations entre gestionnaires doivent avoir lieu (événements + conditions de déclenchement).
- MIB pour le protocole SNMPv2 [Case 93a]: définit les objets gérés qui interfacent une entité de protocole SNMPv2.

2.3 Résumé

Dans ce chapitre, nous avons présenté l'approche SNMP et son extension SNMPv2 conçues pour la gestion des réseaux basés sur une pile de protocoles de type TCP/IP. Les différents modèles ont été présentés et leurs limites identifiées et expliquées.

L'approche SNMP étendue par SNMPv2 permet la mise en place d'une gestion intégrée de réseaux complexes. Les différents modèles basés sur des concepts simples, facilitent le développement de systèmes et d'agents de gestion. Aujourd'hui, de nombreuses plate-formes de gestion basées sur SNMP sont disponibles sur le marché.

Cependant, les approches SNMP et SNMPv2 sont encore trop limitées. La gestion des réseaux s'étendant à celle des systèmes est devenue de plus en plus complexe. Le modèle de l'information doit être plus puissant et proposer des mécanismes de modélisation des activités de gestion adaptés à de gros réseaux hétérogènes. L'absence de fonctions génériques limite également la puissance de l'approche en ne permettant pas leur réutilisation dans de nouvelles applications.

5. . Protocol Data Unit

Les systèmes de gestion futurs, basés sur une approche plus générique de la gestion nécessitent donc des modèles plus adaptés aux besoins des réseaux complexes. Une approche qui propose de tels modèles est celle normalisée par l'ISO. Nous la présentons dans le chapitre suivant.

3
La gestion des réseaux au sein du modèle OSI

L'ISO a entrepris la normalisation des services de télécommunications au début des années soixante-dix. Le but de ces travaux était de définir des normes de communication indépendantes de tout type de machines afin de permettre l'interopérabilité de systèmes issus de différents constructeurs. Depuis, la plupart des constructeurs de réseaux de communications se basent sur ces normes pour développer des systèmes conformes à celles-ci.

Ces activités de normalisation se poursuivent au sein de l'ISO sous le sigle de l'OSI. Dès le début de ces travaux, l'ISO a intégré des aspects relatifs à la gestion des réseaux hétérogènes. Cependant, ce n'est que ces dernières années que des normes stables ont été définies dans ce domaine. Ces travaux font l'objet de ce chapitre.

3.1 Les normes OSI relatives à la gestion de réseaux

Les normes OSI définies pour la gestion de réseaux sont nombreuses et relativement complexes. Elles se divisent en quatre catégories d'après les modèles présentés dans le chapitre 1. Ces catégories sont:

- le modèle organisationnel [ISO-10040 92] définit les rôles des différents intervenants, la stratégie de collecte d'informations et un cadre général pour la modélisation des ressources de gestion.
- le modèle de l'information [ISO-10165.1 92, ISO-10165.2 92, ISO-10165.4 92, ISO-10165.5 92] fournit une approche pour la modélisation des ressources de gestion, et met à la disposition des concepteurs d'éléments de réseaux, des mécanismes de description formelle des objets gérés associés.
- le modèle de communication [ISO-9595 91, ISO-9596 91] définit les services et protocoles nécessaires à l'échange d'informations de gestion entre acteurs.
- les fonctions génériques ([ISO-10164.1 92]..[ISO-10164.7 92]) qui contribuent à la mise en place du modèle fonctionnel. Ces normes comportent un ensemble de définitions de fonctions de gestion relatives aux activités générales de gestion. Une fonction de gestion est définie par la spécification des objets gérés et des opérations nécessaires à sa réalisation. Cette fonction spécifie en fait une interface de gestion.

Chacune de ces normes est à nouveau divisée en de nombreuses parties. Les différents modèles et fonctions ne sont pas indépendants les uns des autres. Nous allons dans ce chapitre présenter les principaux concepts normalisés par l'ISO en passant en revue les différentes normes citées ci-dessus.

3.2 Le modèle organisationnel

Le modèle organisationnel de l'approche OSI intègre l'ensemble des concepts présentés dans le premier chapitre de cette partie (voir page 1.4). Dans le cadre du modèle de référence OSI [ISO-7498.4 89], les différents concepts de gestion-système et gestion de couche présentés au chapitre 1 sont intégrés.

Dans ce modèle, les différents concepts relatifs aux rôles des intervenants dans la gestion (Gestionnaire, Agent) sont inclus. Le modèle est relativement ouvert et permet la conception de hiérarchies de systèmes de gestion en proposant le statut de système dual (i.e. gestionnaire pour un ensemble d'agents et agent pour un gestionnaire situé plus haut dans la hiérarchie). Le modèle prévoit également la mise en place de domaines de gestion (division du réseau en unités de gestion) ce qui permet à plusieurs gestionnaires d'opérer sur un même réseau.

De par sa généricité, le modèle permet la mise en place de tout type d'organisation de ses systèmes de gestion tout en gardant la relation Gestionnaire-Agent de base.

Au niveau de la stratégie de collecte d'informations, les deux mécanismes présentés au chapitre 1 (collecte et compte-rendu d'événements) sont retenus dans le modèle. La présence du mode de compte-rendu d'événements est avantageuse car elle permet, d'une part d'informer à tout moment le gestionnaire de l'occurence d'une anomalie sur le réseau, et d'autre part de minimiser le trafic de gestion en se substituant partiellement au polling. La présence de ces deux modes implique cependant une implantation plus complexe et coûteuse des systèmes de gestion et des agents.

3.3 Le modèle de l'information

L'ISO a, dans le cadre de la gestion OSI, retenu un modèle orienté-objets pour structurer son information de gestion et spécifier ses objets de gestion [ISO-10165.1 92]. Nous allons dans cette section présenter les concepts retenus dans le modèle de l'information.

Définition d'un objet géré

De manière générale, un objet de gestion est défini comme étant l'abstraction des ressources de traitement de données et de communication pour des besoins de gestion. Ces objets représentent la vue d'un système de gestion sur l'ensemble des ressources à gérer. Ces ressources peuvent être soit réelles (ex. câble, répéteur), soit conceptuelles (une association, connexion entre deux entités de protocole). L'ensemble des objets de gestion instanciés au sein d'un système ouvert sont regroupés dans une base d'informations de gestion (MIB).

Un objet géré est défini par un ensemble d'attributs visibles au travers de son interface, un ensemble d'opérations qui peuvent être appliquées sur les attributs, un ensemble d'actions qui peuvent être invoquées sur l'objet et un ensemble de notifications qui peuvent être émises par l'objet géré. La spécification d'un objet géré est complétée par une description de son comportement.

Un objet géré est composé d'un ensemble de modules obligatoires et d'un ensemble de modules optionnels. Un module obligatoire doit être présent dans toute instance de la classe, tandis que la présence d'un module optionnel est fonction, soit des capacités de la ressource modélisée soit de la présence, resp. absence d'autres modules de la classe. Un module ne peut être instancié qu'une seule fois dans un objet géré. Chaque module comporte un certain nombre d'attributs, d'actions, de notifications et de paramètres (voir figure 3.1).

Les attributs représentent l'état de la ressource modélisée et le comportement de l'objet géré auquel

ils appartiennent. Un attribut est défini par un nom, un type, des opérations de gestion autorisées (lecture, affectation, ...) et des opérateurs de comparaisons supportés. Un attribut peut être de type simple (entier, booléen, ...) ou structuré (ensemble, liste, structure). Ces types sont décrits en ASN.1. A tout attribut est associé une valeur. La valeur de l'attribut est visible au travers de l'interface de l'objet géré. Cette valeur peut être lue ou modifiée uniquement au travers d'opérations de gestion.

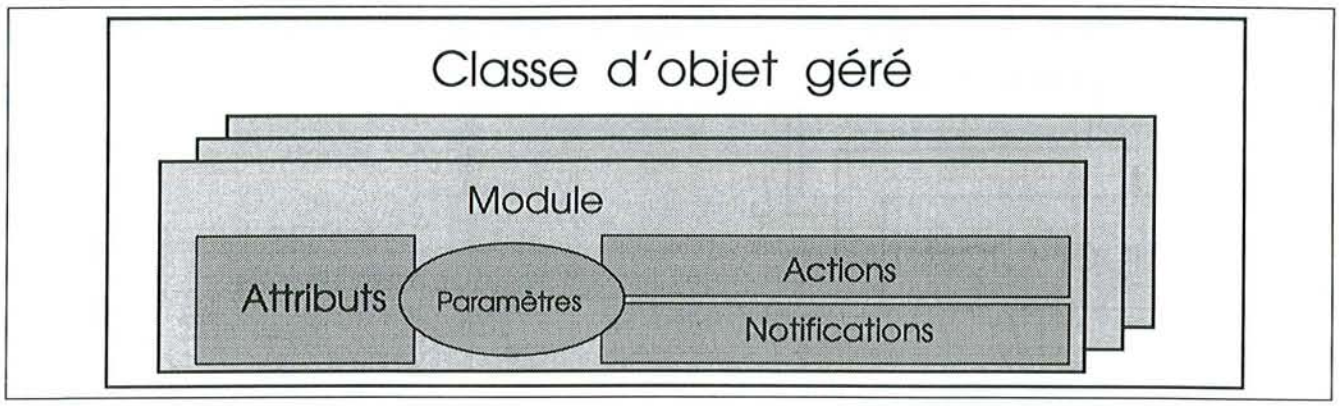

FIG. 3.1 - . *Élements de la spécification d'un objet géré*

Toutes les opérations de base sur un attribut sont normalisées. Ces opérations sont la lecture (GET) et l'affectation (SET). Cette dernière est divisée en un certain nombre de sous opérations qui sont, l'affectation simple (REPLACE), l'adjonction ou la destruction d'un élément (ADD-MEMBER, REMOVE-MEMBER) pour un attribut multi-valué et l'affectation par défaut (REPLACE-WITH-DEFAULT). L'opération d'affectation de valeur peut influencer le comportement de la ressource modélisée par l'instance affectée.

La définition d'une action dans un module permet à tout concepteur d'objet géré d'étendre les fonctionnalités offertes par les opérations de base sur les attributs. Une action est définie par un nom, la liste de ses arguments, la syntaxe de la réponse et par une description de son comportement.

Des objets de gestion peuvent contenir des notifications. Celles-ci sont émises par l'objet géré lors de l'occurence d'événements internes (ex. lors d'un débordement de jauge) ou externes à l'objet géré (ex. destruction d'un objet géré). Une notification est définie par un nom, par les conditions de son émission ainsi que par l'information qu'elle véhicule (c'est à dire ses paramètres).

Les paramètres permettent de faire le lien entre un attribut, une action ou notification et un champ de la PDU transportant l'information. Dans le cas des attributs, les paramètres sont utilisés pour représenter uniquement des informations d'erreurs de traitement. Tandis que pour les actions et les notifications, les paramètres peuvent représenter tout type d'information nécessaire à l'action ou la notification.

A chaque élément composant la définition d'un objet géré est associée une description du comportement. Nous allons, en premier lieu définir de manière claire la notion de comportement.

Définition du comportement

Le comportement d'un objet géré est défini par la séquence des interactions avec son entourage visibles à son interface. Cet entourage comporte le système de gestion, les autres objets gérés et la ressource qu'il modélise.

Une description du comportement d'un objet géré intègre l'ensemble des contraintes, relations et dépendances entre attributs, opérations, actions et notifications. Nous reviendrons de manière très détaillée sur ce contenu dans le premier chapitre de la seconde partie.

L'héritage

Une classe d'objet géré (appelée sous-classe) peut être définie comme spécialisation d'une ou plusieurs autres classes (ci-après appelées super-classes). Dans ce cas, la nouvelle classe hérite de l'ensemble des propriétés des super-classes. Les propriétés héritées par la sous-classe sont les opérations, attributs, notifications, modules, et le comportement de l'ensemble des super-classes. Lorsqu'une même caractéristique est héritée de plusieurs super-classes, celle-ci n'est prise en compte qu'une seule fois dans la sous-classe.

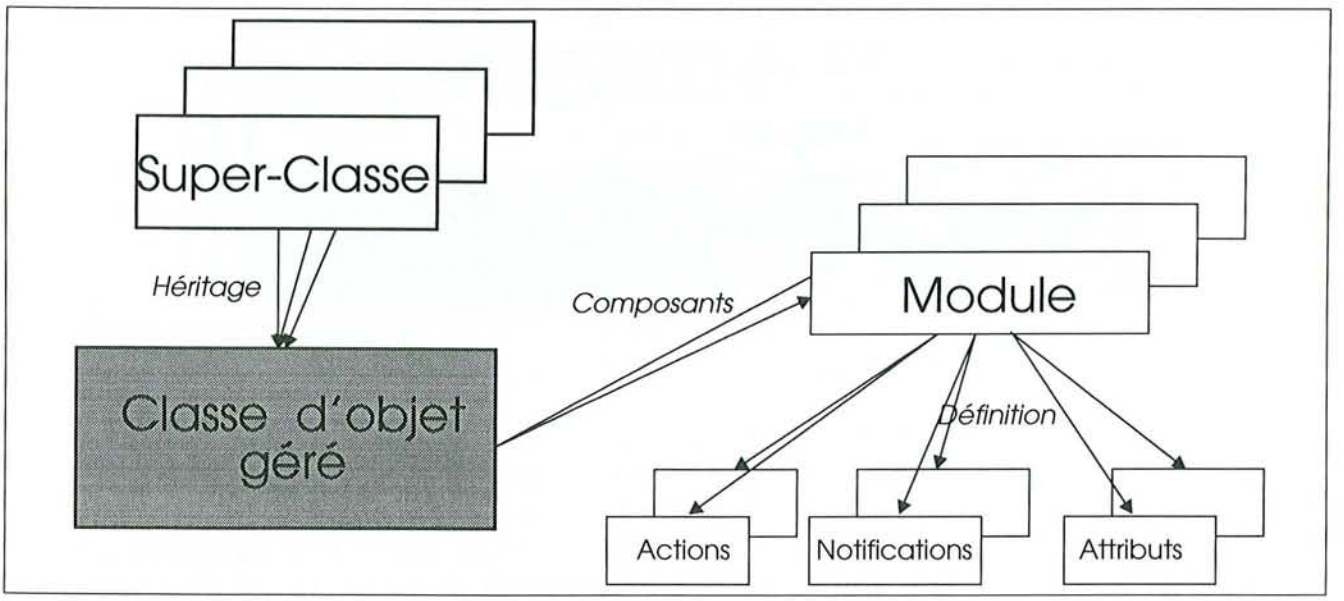

FIG. 3.2 - . *Spécification par héritage et extension d'une classe d'objet de gestion*

La sous-classe peut être étendue par adjonction de nouvelles opérations de gestion, de nouveaux attributs, de nouvelles notifications, par la définition d'un nouveau comportement, et par extension des caractéristiques des super-classes (voir figure 3.2). La spécialisation d'une sous-classe ne doit cependant pas introduire de contradiction par rapport à la définition des super-classes. En effet, seul l'héritage strict est autorisé dans l'approche, ce qui implique que les fonctionnalités héritées doivent être offertes par la sous-classe sans aucune altération.

Un arbre d'héritage générique a été normalisé par l'ISO. La racine est une classe appelée `Top` qui ne comporte qu'un attribut. Cet arbre comporte également l'ensemble des classes d'objets gérés normalisés par l'ISO dans le cadre de la spécification des fonctions génériques de gestion. Tout concepteur d'agent de gestion peut utiliser cet arbre d'héritage comme base pour la spécification de nouveaux objets gérés.

L'organisation des bases d'informations de gestion

Afin de structurer les informations contenues dans une MIB, le modèle de l'information prévoit un mécanisme qui permet la mise en place d'une organisation hiérarchique des instances d'objets gérés. Ce mécanisme est basé sur le concept de contenance qui est défini comme la possibilité d'une instance d'un objet donné de contenir de manière virtuelle une ou plusieurs instances d'autres classes. Ce concept permet de définir une relation de dépendance verticale entre classes d'objets gérés. Cette relation est appelée **corrélation de noms** ("name-binding" [1]). Les corrélations de noms sont définies lors de la spécification des objets gérés et définissent une arborescence statique d'une MIB. L'arbre associé s'appelle

1. . le terme "relation de confinement" est également utilisé dans certaines normes pour décrire les corrélation de noms

l'arbre de contenance (voir figure 3.3). Celui-ci sert à contruire au sein de la MIB une arborescence dynamique utilisée pour le nommage des objets gérés. L'arbre dynamique associé est l'arbre de nommage (voir figure 3.4).

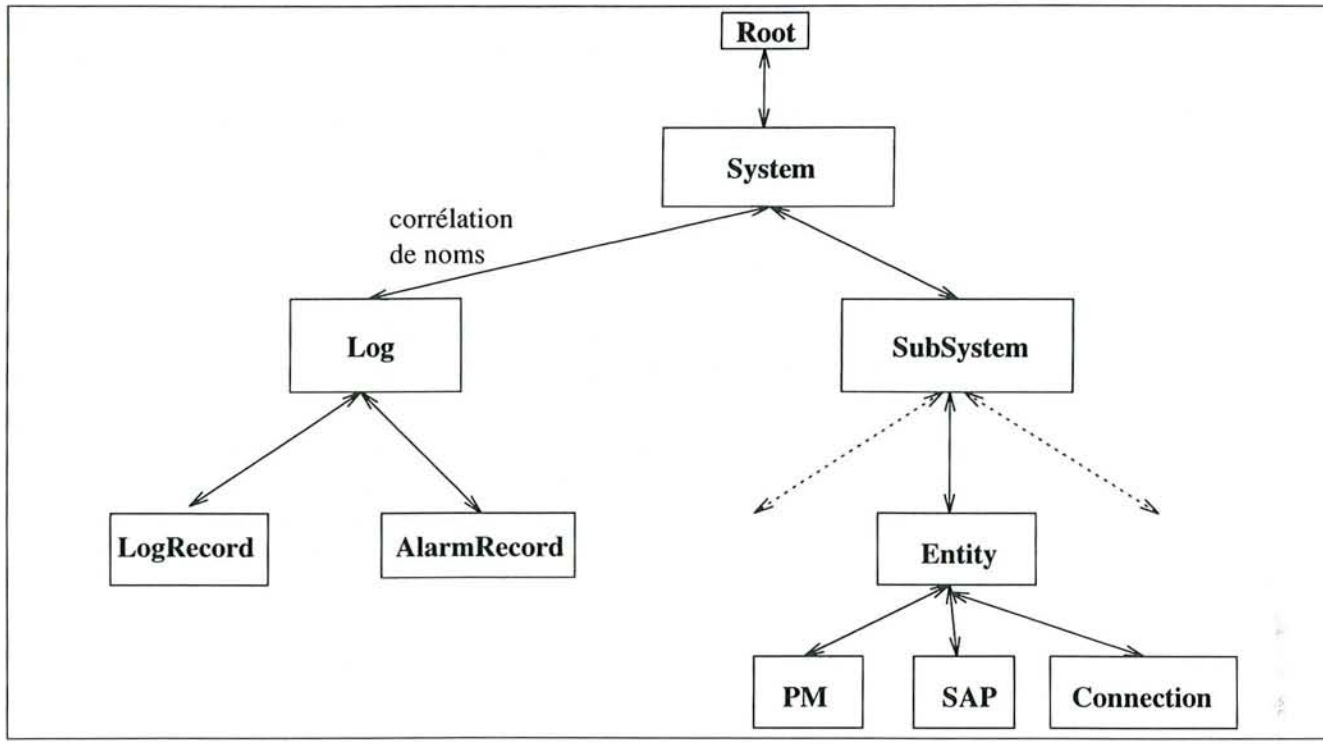

FIG. 3.3 - . *Un exemple d'arbre de contenance*

L'exemple de la figure 3.3 ci-après décrit une organisation typique de MIB (arbre de contenance). L'ensemble des objets gérés sont inclus dans l'objet `root`. Les journaux (`Log`) sont rattachés à un système (`system`) et les enregistrements (`LogRecord`) sont liés à un journal. Chaque sous-système (`Subsystem`) comporte une entité de communication (`Entity`) qui inclut une entité de protocole (`PM`), un point d'accès au service (`SAP`) au travers duquel une connexion (`connection`) est établie. Ce mécanisme de structuration permet de définir une organisation générique de MIB qui peut être instancié pour tout type de réseau [Schott 92].

La définition d'une corrélation de nom comporte la classe englobante, la classe englobée, une définition de contraintes sur la création d'instances de la classe englobée dans une instance de la classe englobante, ainsi que des contraintes sur la destruction d'instances de la classe englobante.

Cette organisation arborescente des MIBs permet de définir des mécanismes puissants de sélection d'instances d'objets gérés. Ces mécanismes sont celui de portée et celui de filtrage. La fonction de portée permet la sélection d'une partie de sous-arbre de la MIB à partir d'une instance de base (appelée racine) et d'un indicateur de profondeur. Ce dernier spécifie la longeur maximale des branches à partir de la racine.

Dans le cadre de l'exemple de la figure 3.4, une sélection par le mécanisme de portée à partir de l'objet `System` permet de sélectioner l'ensemble des instances d'objets contenus (`Log, LogRecord, Subsystem, LogRecord, AlarmRecord`). En limitant la profondeur de portée à 1, seuls les objets directement contenus dans le système sont sélectionnés (`Log, Subsystem`).

Associé à un mécanisme de filtrage des instances, le mécanisme de portée permet l'expression de

sélections d'instances très puissantes. Le filtrage permet au travers d'un prédicat de la logique du premier ordre de spécifier des contraintes sur les attributs d'une instance d'objet géré. Toujours dans le cadre de l'exemple de la figure 3.3, si une opération de filtrage avait pour filtre la présence obligatoire d'un identificateur d'enregistrement, seul les objets `LogRecord` seraient retenus.

Le modèle de l'information permet également la définition de relations horizontales entre instances de classes comme par exemple le lien entre deux objets de connexion. Ces relations sont réalisées dans les objets gérés à l'aide d'attributs spécifiques. Le mécanisme actuellement normalisé par l'ISO présente un certain nombre d'inconsistances et de nombreux travaux sont en cours sur la définition d'un nouveau modèle de relations dans le modèle de l'information OSI [Clemm 93a].

Le nommage et l'identification des ressources

Le nommage d'une instance d'objet se fait sur la base de l'organisation hiérarchique d'une MIB, donc sur la base des corrélations de noms (arbre de contenance). Les noms ont pour but d'identifier une instance d'objet dans le contexte de l'objet englobant. Aussi, le nom d'une instance d'objet est obtenu par combinaison du nom de l'instance d'objet englobante et d'un identificateur unique. Cet identificateur est modélisé dans tout objet par un de ses attributs. La désignation de l'attribut contenant l'identificateur de l'instance d'objet est faite dans la spécification des corrélations de noms pour la classe à laquelle appartient l'instance. Dans l'exemple de la figure 3.4, le nom associé à la seconde instance d'enregistrement de journal est la séquence suivante:

```
(SystemId=" tycho ", logId=" el ", recordId=2).
```

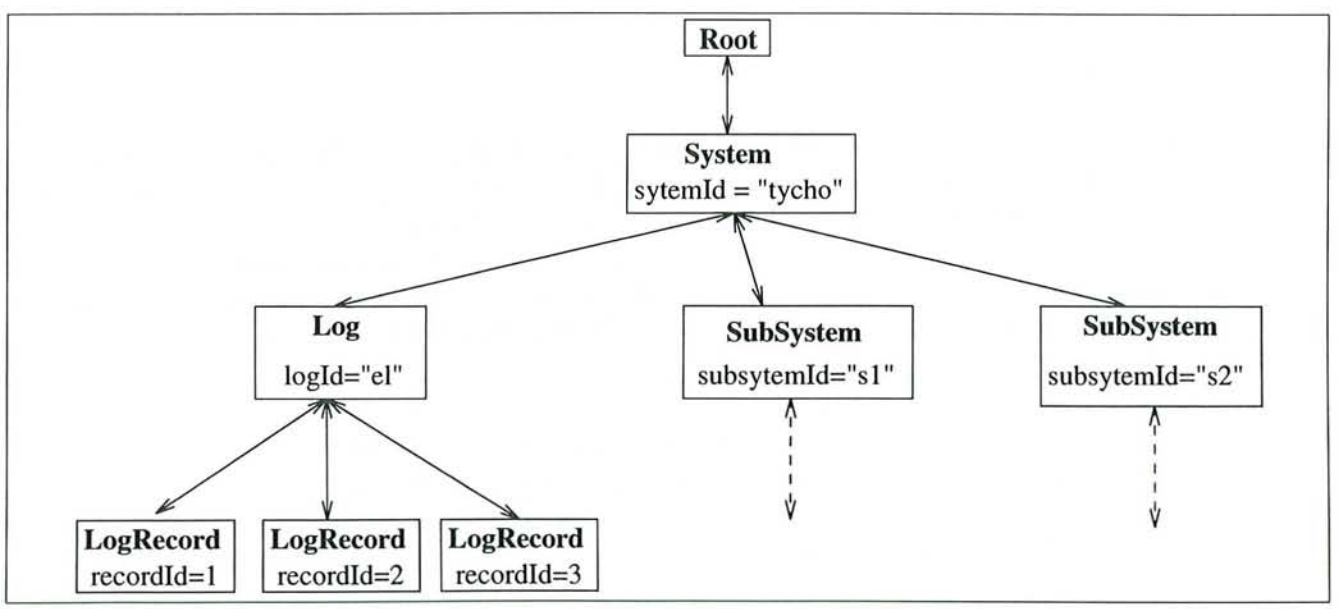

FIG. 3.4 - . *Une instance de MIB*

Une classe d'objet est identifiée par un identificateur d'objet (type ASN.1 prédéfini). Un tel identificateur permet de localiser la classe dans un arbre d'enregistrement des classes d'objets gérés. Par exemple, l'identificateur d'objet associé à la classe d'enregistrement d'événements est la séquence d'entiers suivante:

```
(joint-iso-ccitt ms(9) smi(3) part2(2) managedObjectClass(3) 7).
```

Plusieurs mécanismes permettent, au travers de ces différents identificateurs, la localisation de toute instance d'une classe d'objets gérés dans un réseau [Sylor 93].

GDMO: un langage pour la modélisation des ressources

Dans le cadre du modèle de l'information pour la gestion des systèmes, l'ISO a normalisé un langage de spécification des objets gérés. Le but du langage est de mettre à disposition des concepteurs d'objets gérés un formalisme pour leur spécification. Le langage est basé sur un ensemble de neuf formulaires[2] qui permettent chacun de spécifier une partie d'un objet géré. Cet ensemble comporte des formulaires pour la spécification des classes d'objets gérés, des modules, des attributs, des groupes d'attributs, des actions, des notifications, des paramètres, des corrélations de noms et des comportements. Ce langage très modulaire, incorpore le langage ASN.1 pour la description des types de données, et met à disposition des concepteurs, des mécanismes qui permettent de spécifier de manière formelle leurs objets gérés.

Ce langage va servir de base aux travaux présentés dans cette thèse. Aussi, une présentation détaillée de celui-ci sera faite dans le chapitre 4 de cette partie.

3.4 Le modèle de communication CMIS/CMIP

La gestion à distance des informations présentes dans une MIB basée sur un modèle de l'information OSI, nécessite des services de création et de destruction d'objets gérés, de lecture et d'affectation d'attributs, d'invocation et de confirmation d'actions et finalement de transport de notifications. Dans le cadre de l'OSI, ces fonctionnalités sont offertes par le service CMIS qui est basé sur le protocole CMIP.

3.4.1 Le service CMIS

Le service CMIS[3] est situé dans la couche application du modèle de référence. Il est composé de six unités fonctionnelles et propose un ensemble limité de primitives pour l'échange d'informations de gestion entre processus. Nous allons dans les lignes suivantes présenter ses différentes fonctions.

La création d'une instance d'objet géré

Le service de création d'objet géré (M-CREATE) est invoqué par un utilisateur CMIS pour ordonner à un autre utilisateur la création d'une instance de classe d'objet géré. Ce service est toujours confirmé. L'invocation de ce service nécessite les cinq paramètres suivants:

- nom de la classe de l'instance à créer,
- identificateur de l'instance à créer,
- identificateur de l'instance dans laquelle l'objet doit être inclus,
- liste des attributs à instancier,
- valeurs initiales.

La réponse de la création comporte le résultat de l'opération (succès ou échec).

2. . Dans les normes, le terme formulaire est remplacé par celui de gabarit ou modèle. Il semble cependant que formulaire soit le terme le mieux adapté pour GDMO.

3. . Common Management Information Service [ISO-9595 91]

La destruction d'objet géré

Le service d'effacement d'instances d'objets (M-DELETE) permet à un système de gestion de demander à un agent de gestion de détruire une ou plusieurs instances d'objets gérés dans sa MIB. Le service M-DELETE est toujours confirmé. Les paramètres de la requête sont:

- la classe et l'identificateur de l'instance à détruire,
- éventuellement un paramètre de portée et un filtre pour un ensemble d'instances.

En réponse à une invocation d'effacement d'objets gérés, une réponse est émise par l'agent de gestion pour chaque instance effacée.

Le service d'accès aux valeurs d'attributs

Le service de collecte de valeurs d'attributs (GET) permet à un utilisateur du service (système de gestion) de demander à un autre utilisateur (agent) des valeurs d'attributs d'objets gérés de sa MIB. Le service GET est un service confirmé. La requête comporte les paramètres suivants:

- nom de la classe de base,
- portée de l'opération,
- filtre éventuel,
- liste des attributs dont les valeurs sont demandées.

Pour chaque instance d'objet sélectionnée, une réponse contenant la liste des attributs et leurs valeurs est émise.

Le service d'annulation de lecture de valeurs d'attributs (M-CANCEL-GET) permet d'annuler une requête de collecte en cours (non confirmée à ce moment). Ce service est toujours confirmé. La requête comprend l'identificateur de la requête de collecte à annuler et la confirmation comporte un message d'erreur dans le cas d'un échec.

Ce service est intéressant pour annuler de grosses requêtes de collecte qui, en raison d'événements spéciaux sur le réseau, n'ont plus aucun intérêt pour le gestionnaire. Dans ce cas, l'annulation de la requête permet d'économiser le trafic de gestion éventuellement engendré par l'envoi de nombreuses réponse. Cela permet également une économie des ressources de calcul au niveau du gestionnaire et peut-être au niveau de l'agent, la norme ne précisant pas si ce dernier arrête les exécutions en cours après réception de la demande d'annulation.

Le service d'affectation de valeurs

Le service SET offre la possibilité à un utilisateur du service de demander à un autre utilisateur d'affecter des valeurs à des attributs d'objets gérés de sa MIB. Ce service n'est pas nécessairement confirmé. En plus des paramètres de sélection d'instances d'objets (instance de base dans l'arbre de nommage, portée, filtre), la requête véhicule une liste de triplets (attribut, valeur, opération) qui définissent pour chaque attribut à modifier son identificateur, la valeur à affecter et le type d'opération d'affectation choisie (remplacement, ajout, suppression ou valeur par défaut).

Si le service a été invoqué en mode confirmé, alors l'utilisateur ayant traité la requête émet autant de réponses que d'objets gérés affectés par l'opération.

Les actions

Un concepteur d'objets gérés peut, lors de la spécification, définir pour tout objet des actions spécifiques différentes des opérations de base (lecture, affectation d'attributs). Afin de permettre l'invocation de telles actions sur des objets gérés distants, CMIS propose un service spécifique d'invocation d'action (M-ACTION). Le service M-ACTION peut être en mode confirmé à la demande de l'utilisateur. En plus des paramètres de sélection d'objets gérés identiques à ceux des primitives GET, SET et DELETE, la requête M-ACTION nécessite la présence d'un paramètre qui indique le type de l'action à exécuter. Ce type est défini dans la spécification des objets gérés.

Si l'action est confirmée, le processus ayant traité l'action envoie autant de réponses que d'objets gérés affectés.

Le service de notification d'événements

Le service de notification d'événements (M-EVENT-REPORT) est utilisé par un processus de gestion pour informer un autre processus de gestion de l'occurence d'un événement exceptionnel. Ce service est confirmé à la demande de l'émetteur de la notification (dans la requête). Les paramètres d'une requête de notification d'événements sont en plus du mode, la classe et l'instance d'objet géré à la source de la notification, le type d'événement généré et éventuellement une estampille ainsi que des informations additionnelles sur l'événement. Il est possible de spécifier des paramètres pour les réponses de notifications confirmées.

3.4.2 Le protocole CMIP

CMIP[4] est un protocole de la couche application du modèle OSI. Il réalise le service CMIS tout en étant opérationnel au travers d'une association d'application établie entre deux processus utilisateurs.

CMIP et la couche application du modèle OSI

Au sein de la couche application du modèle de référence [ISO-9545 89] , l'entité CMIP est définie comme un élément de service d'application (ASE[5]). Une entité de protocole CMIP opérationnelle comprend trois élements d'application de base [Collins 89]: l'élément de service de contrôle d'associations (ACSE[6]), l'élément de service CMIP et l'élément de service d'invocation d'opérations distantes (ROSE[7]). L'organisation de ces élements au sein d'une entité de protocole est illustrée dans la figure 3.5.

L'entité ACSE fournit les services d'établissement, de contrôle, et de terminaison d'associations d'application entre deux processus de gestion. L'entité CMIP fournit le service CMIS au processus de gestion. Pour ce faire, l'entité CMIP utilise les services d'invocation d'opérations distantes offerts par ROSE. La coordination de ces ASEs est assurée par une fonction de contrôle d'association simple appelée SACF[8].

Extensions possibles

L'architecture modulaire de la couche application du modèle OSI permet d'étendre assez facilement les fonctionnalités d'un fournisseur de service. Dans le cas de la gestion de réseaux, des besoins d'exécution

4. . Common Management Information Protocol [ISO-9596 91]
5. . Application Service Element
6. . Association Control Service Element [ISO-8649 88]
7. . Remote Operations Service Element [ISO-9072.1 89]
8. . Simple Association Control Function

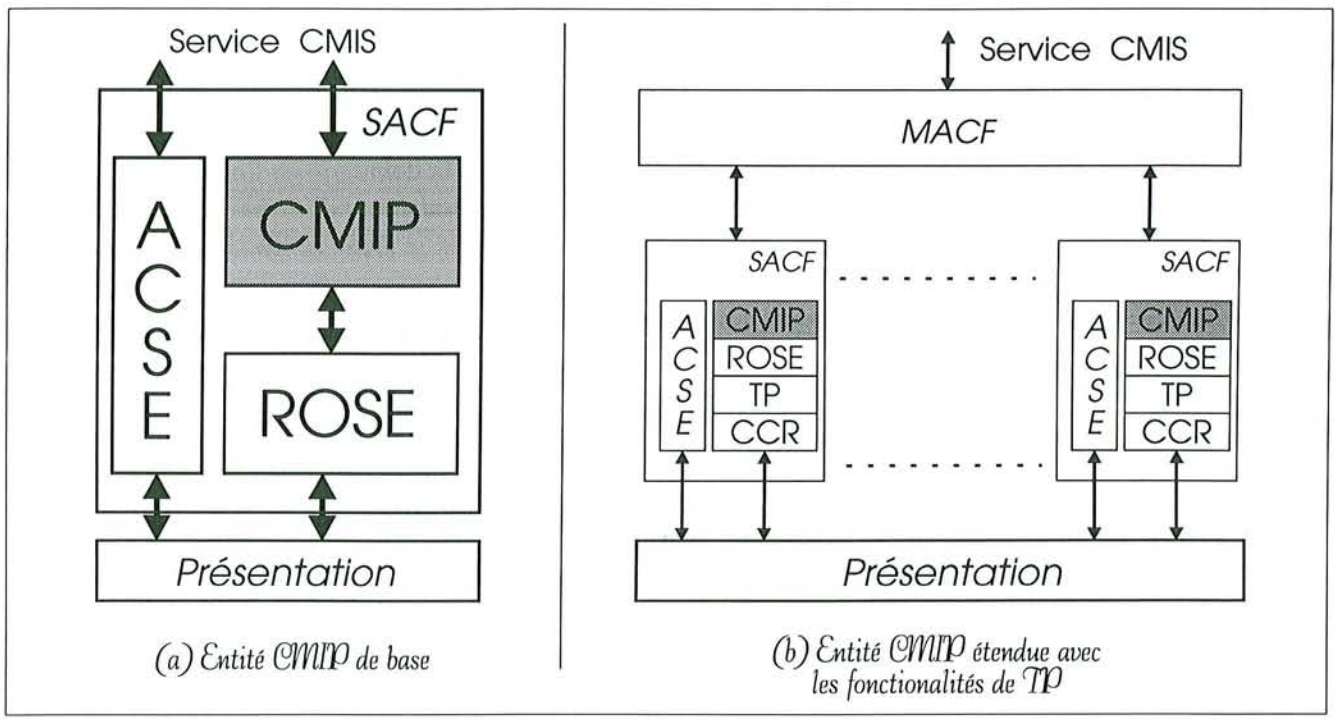

FIG. 3.5 - . *Architectures d'entités de protocole CMIP*

atomique d'opérations distantes ont été identifiés dans de nombreuses situations [Nakai 92, Wester 93]. Pour proposer un tel service, il a été facile de composer un nouveau fournisseur CMIS utilisant en plus des ASEs de base (ACSE, CMIP et ROSE), les services d'autres ASEs tels que TP[9] et CCR[10]. Ces derniers fournissent des services qui permettent la mise en place de transactions distribuées. La figure 3.5.(b) illustre une telle architecture. Plusieurs associations sur lesquelles transitent des opérations de gestion y sont établies. La synchronisation et le contrôle du déroulement des transactions en cours sont assurés par la fonction de contrôle des accès multiples (MACF[11]) qui fournit dans ce cas le service CMIS étendu.

De par la structure modulaire de la couche application, d'autres architectures d'entités de protocole sont imaginables. Leur nombre est limité par celui des éléments de service d'application spécifiques existants.

3.5 Le modèle fonctionnel

L'approche OSI intègre dans son modèle fonctionnel les cinq activités principales de la gestion de réseaux définies au chapitre 1 section 1.3. Dans le cadre de ses activités, l'ISO a normalisé à ce jour un certain nombre de fonctions qui définissent des éléments de base pour la gestion. La définition d'une fonction de gestion comporte l'identification du champ d'action de celle-ci, la définition des opérations fournies par la fonction, une spécification des objets de gestion nécessaires à son intégration et la description des attributs et notifications spécifiques.

9. . Distributed Transaction Processing [ISO-10026.2 92]
10. . Commitment, Concurrency and Recovery [ISO-9804 90]
11. . Multiple Access Control Function

Les fonctions de gestion existantes sont relativement nombreuses. Le but de cette section n'est pas de présenter en détail chacune d'entre elles. Le lecteur intéressé pourra en trouver une description très minutieuse dans [Lecerf 93]. Nous allons simplement donner un bref aperçu des fonctions les plus importantes.

Gestion des objets gérés

La fonction de gestion des objets gérés [ISO-10164.1 92] fournit à l'utilisateur un certain nombre de services qui lui permettent de manipuler les objets gérés et les attributs de ceux-ci. Dans ce cadre, la fonction définit un service de transparence (PT[12]) entre les opérations normalisées offerts à l'interface d'une MIB et le service de communication utilisé pour le transfert de ces opération (actuellement CMIS). Le service de transparence comprend autant de primitives que d'opérations normalisées (neuf au total). Il définit pour chaque opération la manière dont elle est mappée sur un service CMIS.

Un objet de gestion n'est pas toujours créé suite à l'invocation d'une opération de gestion par un gestionnaire. Un objet peut éventuellement être créé suite à un changement dans le réseau (ex. établissement d'une nouvelle connexion). Il en est de même pour les valeurs des attributs qui peuvent aussi changer suite à un événement externe (ex. un compteur du nombre de paquets corrompus reçus est incrémenté de 1). Afin d'informer le gestionnaire de l'occurence de tels événements, la fonction définit en plus du service de transparence, quatre notifications. Celles-ci sont la création d'objet géré (`Object-Creation-Notification`), la destruction d'objet géré (`Object-DeletionNotification`), la notification de changement de nom d'un objet géré (`Object-Name-Change-Notification`) et celle de changement de valeur d'un attribut (`Attribute-Value-Change-Notification`).

Gestion des états

La fonction de gestion des états [ISO-10164.2 92] définit le modèle général pour la représentation de l'état d'une ressource. Dans ce modèle, l'état d'une ressource est représenté à l'aide de trois attributs de base (les plus importants) et six attributs de statut (attributs secondaires). Les trois attributs de base sont:

1. l'état opérationnel (`OperationalState`) qui décrit l'état physique de la ressource. Les valeurs possibles de l'état sont `enable` (la ressource est installée et elle est opérationnelle), `disable` (la ressource n'est pas opérationelle).

2. l'état d'utilisation (`UsageState`) qui décrit l'utilisation actuelle de la ressource en termes de nombre de requêtes en cours. Ces valeurs sont `idle` (aucune requête n'est en cours de traitement par la ressource), `active` (une ou plusieurs requêtes sont en cours de traitement), `busy` (le nombre maximum de requêtes pouvant être traitées a été atteint) et `unkown` (il n'est pas possible de connaître la charge de la ressource).

3. l'état administratif (`AdministrativeState`) qui décrit l'état administratif de la ressource. Il est géré par le système de gestion. Les valeurs possibles sont `unlocked` (la ressource n'est pas verrouillée et autorisée à être utilisée), `locked` (la ressource est verrouillée par le système de gestion) et `shutting-down` (la ressource est en cours de désactivation, aucune autre opération n'est autorisée).

Les six attributs de statut sont utilisés pour complèter les trois attributs d'état. Ils permettent d'obtenir une information plus précise sur l'état actuel de la ressource. Les valeurs de ces attributs définissent un certain nombre de conditions applicables à la ressource.

12. . Pass Through Service

La fonction de gestion des états définit la syntaxe et la sémantique de tous ces attributs ainsi que leurs dépendances. En plus, la fonction de gestion des états définit une notification de changement d'état (`State-Change-Notification`) émise lorsque la valeur d'un ou plusieurs attributs d'états d'un objet géré est modifiée. Nous allons dans les chapitres suivants de cette thèse utiliser ces attributs et notifications.

Autres fonctions

De nombreuses autres fonctions qui définissent des interfaces de gestion négociables sont aujourd'hui normalisées ou en cours de l'être (alarmes, journaux, gestion d'événements, mesure de coûts, supervision de charge, gestion des tests, contrôle d'accès, etc...). Une fonction n'est pas spécifique à une activité de gestion. Par exemple, la fonction de gestion des journaux peut servir à la fois dans l'activité de gestion de la sécurité et dans celle de la gestion des fautes. L'ensemble de ces fonctions procure une bibliothèque de base stable pour le développement d'applications normalisées de gestion.

3.6 Quelques remarques sur l'approche

L'approche OSI pour la gestion de réseau propose des modèles puissants et complexes.

Son modèle organisationnel permet la définition de toutes sortes de répartitions de rôles entre les différents intervenants. Le modèle fonctionnel, riche de nombreuses fonctions de gestion prédéfinies, fournit une base solide pour le développement d'applications normalisées.

Le modèle de l'information basé sur une approche de la modélisation des ressources orientée-objets est le premier qui permet, d'une part d'encapsuler des données, et d'autre part de structurer de manière hiérarchique des bases d'informations de gestion.

Le modèle de communication étant basé sur un protocole d'application du modèle de référence, permet de répondre à tout type d'exigences en raison de l'architecture modulaire de la couche. Dans ce cadre, on pense notamment au couplage du protocole CMIP avec les fonctionnalités du protocole de traitement transactionnel OSI TP.

De nombreuses extensions sont possibles et de nombreux travaux de recherches s'activent sur les différents modèles (organisationnel: domaines de gestion, information: modèles de relations entre objets gérés, communication: transactions).

3.7 Résumé

Nous avons présenté dans ce chapitre l'approche retenue par l'ISO pour la gestion des réseaux hétérogènes. Cette approche propose des mécanismes très puissants pour la modélisation des ressources et pour la collecte d'informations de gestion au travers de son protocole de communication.

Si les mécanismes sont puissants, ce qui est notamment le cas pour le modèle de l'information, le coût du développement de systèmes de gestion de réseaux basés sur une telle approche est beaucoup plus lourd que dans une approche simple. Le succès de cette dernière est conditionné par la disponibilité d'un environnement de développement qui facilite la conception de systèmes basés sur cette approche.

Dans le cadre du modèle de l'information, l'ISO a développé et normalisé GDMO, un langage formel pour la spécification des objets gérés. Ce langage est reconnu et largement utilisé dans le monde des télécommunications pour la description des objets gérés. Il doit servir de base à tout environnement de développement d'applications de gestion. Nous allons présenter ce langage dans le chapitre suivant.

4
Le langage GDMO

Afin d'uniformiser la modélisation des ressources, l'ISO a retenu une approche orientée-objets pour la modélisation des ressources de gestion. Cette approche s'appuie sur un ensemble de concepts de modélisation (héritage, encapsulation, ...) et comprend une notation normalisée appellée GDMO[1] [ISO-10165.4 92] utilisée comme technique de description formelle pour la spécification des interfaces de gestion des ressources gérées.

Le langage GDMO fournit un ensemble de neuf formulaires permettant une description modulaire et normalisée des objets gérés et de leur organisation au sein d'une base d'informations de gestion. Dans ce chapitre, nous allons présenter de manière détaillée les formulaires les plus importants.

4.1 Les formulaires normalisés

Le langage GDMO, basé sur un ensemble de neuf formulaires, permet de spécifier les différents élements d'une base d'informations de gestion. Les formulaires du langage sont:

- le **formulaire de classe d'objets gérés** qui permet de spécifier des classes d'objets gérés en définissant les modules qui les composent et en les liant aux super-classes dont elles héritent.
- le **formulaire de module**[2] utilisé pour définir les modules en spécifiant pour chacun d'eux les attributs, actions, notifications et le comportement qui le composent.
- le **formulaire d'attribut** permet de définir des attributs types.
- le **formulaire de groupe d'attributs** permet regrouper sous une étiquette un certain nombre d'attributs. Un groupe peut par la suite être référencé dans un module.
- le **formulaire d'action** utilisé pour définir des actions autorisées sur des objets gérés.
- le **formulaire de notification** qui permet de spécifier des notifications pouvant être émises par des objets gérés. Ces notifications sont inclues dans des modules.
- le **formulaire de paramètre** pour définir des paramètres d'attributs, d'actions et de notifications.
- le **formulaire de comportement** qui permet de définir et d'enrengistrer sous une étiquette, la définition d'un comportement.

1. . Guidelines for the Definition of Managed Object
2. . Le module est également appelé paquetage ou ensemble suivant les personnes ou les normes

- le **formulaire de corrélation de noms** utilisée pour spécifier les liens de contenance entre différents objets gérés au sein d'une base d'informations de gestion. Ces liens sont à la base de la hiérarchie d'une MIB.

Le lien entre les différents formulaires se fait toujours au travers de références. Par exemple, un module utilisé dans une classe est référencé dans la définition de celle-ci par son label.

Nous allons dans la suite de ce chapitre, présenter de façon détaillée les formulaires de classe d'objets gérés, de module, d'action, de notification, de comportement et de corrélation de noms. Ces formulaires sont à la base des extensions proposées dans la suite de la thèse. Une description détaillée des autres formulaires de GDMO est donnée en annexe de cette thèse.

La présentation comporte, pour chaque formulaire, une description de ses éléments, suivie d'un exemple de son utilisation. Notre exemple sera basé sur la spécification d'un système de communication issue du catalogue d'objets gérés établi dans le cadre *OmniPoint*[3] [NMF 92].

4.2 La définition d'une classe

Une classe d'objets de gestion est caractérisée par son nom, la liste des classes dont elle hérite les caractéristiques, un ensemble de modules qui définissent l'extension des fonctionalités, ainsi qu'un identificateur pour l'enregistrement de la classe dans l'arbre d'enregistrement.

4.2.1 Le formulaire de spécification de classe

Toute classe est définie à l'aide du modèle décrit dans la figure 4.1. Une classe d'objets gérés (**MANAGED OBJECT CLASS**) est identifiée de manière locale par une étiquette et de façon globale par un identificateur d'objets (clause **REGISTERED AS** <object-identifier>) servant d'entrée dans l'arborescence de d'enregistrement.

```
<class-label> MANAGED OBJECT CLASS
[ DERIVED FROM <class-label> [ ,<class-label>]*;
]
[ CHARACTERIZED BY <package-label> [ ,<package-label>]*;
]
[ CONDITIONAL PACKAGES <package-label> [ ,<package-label>]*
                                    PRESENT IF condition
                    [ ,<package-label> [ ,<package-label>]*
                                    PRESENT IF condition]*;
]
 REGISTERED AS <object-identifier>;
```

FIG. 4.1 - . *Le formulaire de définition de classe d'objets gérés*

La définition d'une classe est quant à elle obtenue, d'une part par héritage en énumérant la liste des super-classes dont elle hérite les caractéristiques (champ **DERIVED FROM**), et d'autre part par

3. . OmniPoint comprend un certain nombre de catalogues dont des spécifications d'objets gérés, rédigés par le Network Management Forum qui regroupe un ensemble de partenaires industriels dont le but est de fournir des systèmes de gestion de réseaux interopérables basés sur les normes OSI.

spécialisation (adjonction de nouvelles fonctionnalités au travers de modules supplémentaires, clauses **CHARACTERIZED BY** et **CONDITIONAL PACKAGES**).

La clause **CHARACTERIZED BY** contient la liste des modules obligatoires de l'objet géré. La clause **CONDITIONAL PACKAGES** permet de référencer des modules optionnels. A chaque liste de modules est associée une expression conditionnelle après le mot clé **PRESENT IF**. La condition de présence d'un module est, dans la version actuelle de la norme, exprimée en langage naturel. Celle-ci concerne la plupart du temps le support par la ressource d'une fonctionnalité spécifiée dans le module et les dépendances entre modules optionnels.

4.2.2 Un exemple

La figure 4.2 présente la spécification GDMO d'une classe d'objets gérés de type système informatique (`computerSystem`). Cette classe est dérivée de la classe `top` et en hérite les deux attributs qui sont `allomorphicClasses` (liste des classes allomorphiques) et `packages` (liste des modules instanciés).

```
computerSystem MANAGED OBJECT CLASS
  DERIVED FROM top;
  CHARACTERIZED BY computerSystemPkg;
  CONDITIONAL PACKAGES
   peripheralNamePkg        PRESENT IF "an instance supports it and the
                                        peripheralListPkg is not present",
   peripheralListPkg        PRESENT IF "an instance supports it and the
                                        peripheralNamePkg is not present",
   processingEntityNamePkg  PRESENT IF "an instance supports it and the
                                    processingEntityListPkg is not present",
   processingEntityListPkg  PRESENT IF "an instance supports it and the
                                     processingEntityNamePkg is not present",
   systemTimePkg            PRESENT IF "an instance supports it",
   upTimePkg                PRESENT IF "an instance supports it",
   userLabelPackage         PRESENT IF "an instance supports it",
   usageStatePkg            PRESENT IF "the ressource can detect usage ";
 REGISTERED AS { forum-objectClass  1 };
```

FIG. 4.2 - . *Exemple de la définition d'une classe de système informatique*

La classe est composée d'un module obligatoire (`computerSystemPkg`), et de huit modules conditionnels. La présence d'un module conditionnel est fonction du support de ses caractéristiques dans la ressource modélisée et de la présence d'autres modules conditionnels. Par exemple, les modules `peripheralListPkg` et `peripheralNamePkg` sont de par leurs conditions de présence, en mutuelle exclusion. Le module `peripheralNamePkg` permet de représenter un périphérique utilisé par la ressource, tandis que le module `peripheralListPkg` permet de représenter plusieurs périphériques en relation avec la ressource (ici le système informatique). Le module `systemTimePkg` est instancié si et seulement si le système dispose d'une horloge. Il en est de même pour le module `upTimePkg` qui permet de représenter à tout moment le temps écoulé depuis le dernier démarrage du système. L'objet géré `computerSystem` est enregistré dans l'arbre d'enregistrement sous la racine {`forum-objectClass 1`}.

4.3 Les modules

Composant de base d'une classe de ressources de gestion, le module regroupe un ensemble d'attributs, d'actions et de notifications pouvant être émises par une instance de classe le contenant. Le lien entre les différents éléments d'un module est réalisé dans une description du comportement de ce dernier.

4.3.1 Le formulaire

A chaque composant d'un module (**PACKAGE**) est associé un champ dans lequel les éléments peuvent être définis (voir figure 4.3).

Le formulaire de comportement (**BEHAVIOUR**) comprend soit des labels référençant des descriptions de comportement, soit directement une description en langage naturel du comportement associé. Ce comportement décrit les conditions et effets des actions et opérations incluses dans le module sur les attributs, ainsi que les circonstances dans lesquelles les notifications sont émises.

Le champ **ATTRIBUTES** permet l'inclusion d'attributs par référence à des étiquettes d'attributs <attribute-label> dans le module. A tout attribut référencé correspond une liste de propriétés *property-list*. Celle-ci définit l'ensemble des opérations valides sur l'attribut ainsi que les contraintes sur les valeurs par défaut, valeurs initiales requises et permises. Une description détaillée de ces propriétés sera donnée plus loin dans cette section. A tout attribut peuvet également être associés un ou plusieurs paramètres <parameter-label>. Ces derniers sont utilisés lors de l'émission d'une notification d'erreurs suite à une défaillance lors du traitement d'une opération de remplacement de valeur de l'attribut.

```
<package-label> PACKAGE
[  BEHAVIOUR <behaviour-label>   [ ,<behaviour-label>]*;
]
[  ATTRIBUTES <attribute-label> property-list [<parameter-label>]*
              [ ,<attribute-label> property-list [<parameter-label>]*]*;
]
[ ATTRIBUTE GROUPS <group-label> [<attribute-label>]*
                   [ ,<group-label> [<attribute-label>]*]*;
]
[ ACTIONS <action-label> [<parameter-label>]*
          [ ,<action-label> [<parameter-label>]*]*;
]
[ NOTIFICATIONS <label-notification> [<parameter-label>]*
                [ ,<label-notification> [<parameter-label>]*]*;
]
[ REGISTERED AS <object-identifier>];
```

FIG. 4.3 - . *Le formulaire de définition de module*

Le champ de groupe d'attributs (**ATTRIBUTE GROUPS**) permet de regrouper sous une étiquette <group-label> un certain nombre d'attributs référencés par leurs étiquettes <attribute-label>. Ce regroupement est intéressant car il permet au travers d'une seule opération d'affecter l'ensemble des attributs du groupe. Un tel groupe est soit fixe, soit extensible via l'adjonction de nouveaux attributs dans une spécialisation de la classe. Le statut d'extensibilité est défini dans le formulaire de groupe.

La clause **ACTIONS** référence toutes les actions présentes dans le module. Ces actions sont référen-

cées par des labels <action-label>. Il est possible d'associer à une action un ensemble de paramètres relatifs à une invocation ainsi qu'à une réponse ou erreur spécifique à la classe d'objet géré dans laquelle le module est inclus. La sémantique du paramètre est décrite dans sa définition.

La clause **NOTIFICATIONS** contient la liste des notifications présentes dans le module. Chaque notification est référencée par une étiquette <notification-label> et peut supporter un certain nombre de paramètres supplémentaires <parameter-label> spécifiques au module.

L'enregistrement d'un module dans l'arbre d'enregsitrement (clause **REGISTERED AS**) n'est pas obligatoire.

```
property-list     ->   default initial subtyping get-replace add-remove

default     -> [ REPLACE-WITH-DEFAULTS]
               [ DEFAULT VALUE value-descriptor]
value-descriptor -> value-reference
                  | DERIVATION RULE <behaviour-label>
initial     -> [ INITIAL VALUE value-reference]
             | [ INITIAL VALUE DERIVATION RULE <behaviour-label>]
subtyping   -> [ PERMITTED VALUES type-reference]
               [ REQUIRED VALUES type-reference]
get-replace -> GET | REPLACE | GET-REPLACE
add-remove  -> ADD | REMOVE | ADD-REMOVE
```

FIG. 4.4 - . *Les propriétés d'attributs dans un module*

Les propriétés associées à un attribut peuvent également être définies dans le formulaire de module à l'aide des éléments décrits dans la figure 4.4. La liste des propriétés qui peuvent être exprimées comporte quatre éléments distincts.

Première clause de propriété d'attribut, la clause *default* indique si l'opération d'affectation de valeurs par défaut est autorisée sur l'attribut dans le module (champ **REPLACE-WITH-DEFAULT**). Si la propriété **DEFAULT VALUE** est présente, la valeur prise lors d'une opération de réaffectation par défaut, est celle de la valeur obtenue par traitement du descripteur de valeur *value-descriptor*. Ce dernier peut référencer une valeur de référence `value-reference` ou une règle de dérivation **DERIVATION RULE** dont le comportement associé par <behaviour-label> décrit la manière dont la valeur par défaut est obtenue. Si la clause de remplacement par défaut est présente et la propriété de valeur par défaut absente, le système de gestion est responsable de l'affectation de cette valeur.

Le concepteur d'objets gérés peut également définir une valeur initiale pour chaque attribut du module. Exprimée dans la liste des propriétés grâce à la clause **INITIAL VALUE**, elle peut être définie comme référence à une valeur `value-reference` ou comme algorithme de dérivation (cas **DERIVATION RULE**).

Tout attribut est défini par le type ou sous-type ASN.1 auquel il se réfère. Lors de l'intégration de l'attribut dans le module, il est possible de restreindre le domaine des valeurs en définissant, d'une part le domaine des valeurs permises (**PERMITTED VALUES**) dans le module, et d'autre part le domaine des valeurs requises (**REQUIRED VALUES**).

Il est également possible de définir pour chaque attribut la liste des opérations normalisées autorisées sur l'attribut. Ces opérations sont celles de lecture (**GET**), de réaffectation (**REPLACE**), d'adjonction (**ADD**) et de suppression (**REMOVE**) d'éléments ou toute combinaison de ces opérations (**GET-**

REPLACE, ADD-REMOVE).

4.3.2 Un exemple

La figure 4.5 comporte la définition du module **computerSystemPkg**. Il est le seul module obligatoire de la classe `computerSystem` décrite précédemment. Son comportement est défini dans deux entités de comportement qui sont `computerSystemPkgDefinition` et `computerSystemPkgBehaviour`. Le module comporte cinq attributs, un groupe d'attributs et sept notifications.

```
computerSystemPkg PACKAGE
  BEHAVIOUR computerSystemPkgDefinition, computerSystemPkgBehaviour;
  ATTRIBUTES
    computerSystemId      GET,
    operationalState      GET,
    availabilityStatus    GET;
    administrativeState   GET-REPLACE,
    alarmStatus           GET-REPLACE, ADD-REMOVE,
  ATTRIBUTE GROUPS state
    administrativeState operationalState alarmStatus availabilityStatus;
  NOTIFICATIONS
    objectCreation, objectDeletion, attributeValueChange, stateChange,
    processingErrorAlarm, environmentalAlarm, equipmentAlarm;;
```

FIG. 4.5 - . *Définition du module système informatique*

Sur les cinq attributs, trois ne sont autorisés qu'en lecture: `computerSystemId`, `operationalState` et `availabilityStatus`. Deux attributs sont autorisés en lecture-écriture: `administrativeState` et `alarmStatus`. L'attribut `alarmStatus` est de type ensemble. Les opérations d'ajout et de supression d'éléments y sont également autorisées.

Le groupe d'attributs `state` est normalisé par l'ISO. Il comporte l'ensemble des attributs d'état d'un objet. Etendu avec les attributs d'état, il est ici inclus dans la définition d'un objet géré qui modélise un système informatique. Le module inclut les notifications relatives à la création et destruction d'une instance d'un objet `objectCreation` et `objectDeletion`, les notifications de changement d'état et de changement de valeur d'attribut `attributeValueChange`, `stateChange` ainsi que les trois notifications d'alarmes normalisées `processingErrorAlarm`, `environmentalAlarm` et `equipmentAlarm`.

4.4 Les actions

L'approche OSI permet de définir pour chaque objet géré, des actions qui étendent les opérations prédéfinies sur les objets et leurs attributs. Une action est définie par un label, un comportement, une liste de paramètres et une syntaxe d'invocation et de réponse.

4.4.1 Le formulaire

Le formulaire de spécification d'action **ACTION** comporte cinq champs principaux illustrés par la figure 4.6. Le champ **BEHAVIOUR** contient une liste de références à des labels de comportement.

Ceux-ci contiennent la définition de l'action, une spécification des paramètres associés, les résultats de l'action ainsi que sa sémantique.

Tous les paramètres associés à une action sont référencés dans le champ **PARAMETERS** de la spécification de celle-ci. Ces paramètres sont d'invocation, de réponse, spécifiques à un contexte, ou d'erreur.

```
<action-label> ACTION
 [ BEHAVIOUR <behaviour-label> [,<behaviour-label>]*;
 ]
 [ MODE CONFIRMED;
 ]
 [ PARAMETERS <parameter-label> [,<parameter-label>]*;
 ]
 [ WITH INFORMATION SYNTAX type-reference;
 ]
 [ WITH REPLY SYNTAX type-reference;
 ]
REGISTERED AS <object-identifier>;
```

FIG. 4.6 - . *Le formulaire de spécification d'action*

Toute invocation d'action au travers d'un réseau de communication peut être confirmée ou non. La présence du champ **MODE CONFIRMED** indique la nécessité de confirmer toute invocation d'action. La ou les confirmations résultantes sont alors transmises à travers une ou plusieurs primitives de service `M-ACTION-response` de CMIP. En l'absence de ce champ, la responsabilité de gérer les confirmations est laissée au gestionnaire.

Le champ **WITH INFORMATION SYNTAX** définit la structure ASN.1 associée à une invocation de l'action au travers d'un appel au service CMIS `M-ACTION`. Le lien s'effectue au travers de la référence de type. Le champ **WITH REPLY SYNTAX** définit la structure ASN.1 associée à une réponse d'invocation de l'action. Cette réponse est envoyée au travers de la primitive de service `M-ACTION-response`. Les références de types, faites dans ces deux champs, correspondent à des types ASN.1 également utilisés dans la définition du service spécifique CMIS. Ces données sont le plus souvent de type `SEQUENCE` et contiennent dans le premier cas, la structure des paramètres associés à un appel à l'action, et, dans le second cas, la structure des données fournies en réponse à une invocation.

4.4.2 Un exemple

Très peu d'actions sont présentes dans les catalogues d'objets gérés. La raison de ce manque repose, d'une part sur l'existence d'opérations normalisées sur l'ensemble des attributs existants, et d'autre part sur la généricité des objets gérés normalisés. L'exemple de spécification d'action donné ci-dessous est, comme l'ensemble des exemples du chapitre, issu du catalogue d'objets gérés publié par le Network Management Forum [NMF 92]. Cette action appelée `activate` a pour but de modéliser la mise en service d'une ressource. L'éxécution de cette opération doit pouvoir vérifier l'ensemble des contraintes entre les attributs, initialiser les attributs de relations de l'objet géré, et autoriser l'utilisation de la ressource en positionnant les attributs d'état opérationnel à `enable` et administratif à `unlocked`. Si l'action a pu être menée à bien, la réponse doit comporter la mention `successful`, sinon le message `failureResponse` doit être spécifié dans la confirmation.

```
activate ACTION
   BEHAVIOUR activateBehaviour;
   MODE CONFIRMED;
   WITH REPLY SYNTAX SYNTAX-1.ActivateActionReply;
 REGISTERED AS {forum-action 1};
```

FIG. 4.7 - . *L'action d'activation de système*

La figure 4.7 présente la spécification GDMO de cette action. Son comportement se réfère à une définition qui contient les explications données ci-dessus. De par la présence du mode **CONFIRMED**, l'action est toujours confirmée. Elle ne nécessite pas d'informations spécifiques lors de son invocation, ce qui explique l'absence du champ **WITH INFORMATION SYNTAX**. Par contre, la réponse comporte un champ d'informations de type ASN.1 `ActivateActionReply`. Ce dernier correspond à une définition de séquence comprenant, d'une part le code de retour de l'action (succès ou échec) sous forme d'identificateur d'objet, et d'autre part une liste optionnelle de paramètres spécifiques aux objets qui implémentent l'action.

4.5 Les notifications

Une notification représente un événement généré par un objet suite à un changement de son état. En GDMO, une notification est définie par un label, un ensemble de comportements, des paramètres, la syntaxe de transfert de l'information au sein d'une notification émise au travers du protocole de gestion (CMIP) et la syntaxe de l'information contenue dans une confirmation éventuelle de la notification.

4.5.1 Le formulaire

La spécification d'une notification comporte quatre clauses principales (voir figure 4.8). La clause **BEHAVIOUR** comprend la sémantique de la notification, les conditions dans lesquelles elle est émise, et les données associées à cette émission.

La clause **PARAMETERS** contient la liste des paramètres d'information ou de réponses associés à la notification. Le champ **WITH INFORMATION SYNTAX** référence le type ASN.1 associé à l'information de la notification.

Il est possible lors de la définition de la syntaxe de l'information associée à l'envoi d'une notification au travers du protocole de gestion, d'associer à chaque champ de cette syntaxe un attribut spécifique dont la valeur sera transmise dans ce champ (section **AND ATTRIBUTE IDS**). Pour chaque champ `<field-name>` de la notification, l'attribut associé `<attribute-label>` devra être défini dans le même module que la notification.

Le champ **WITH REPLY SYNTAX** définit la structure de données ASN.1 associée à la réponse de la notification, si celle-ci est émise par l'agent de gestion en mode confirmé.

```
<label-notification> NOTIFICATION
 [ BEHAVIOUR <behaviour-label> [,<behaviour-label>]*;
 ]
 [ PARAMETERS <parameter-label> [,<parameter-label>]*;
 ]
 [ WITH INFORMATION SYNTAX type-reference
   [ AND ATTRIBUTE IDS
      <field-name> <attribute-label>
      [,<field-name> <attribute-label>]*
   ];
 ]
 [ WITH REPLY SYNTAX type-reference;
 ]
REGISTERED AS <objet-identifier>;
```

FIG. 4.8 - . *Le formulaire de spécification de notification*

4.5.2 Un exemple

La spécification illustrée dans la figure 4.9 correspond à la description de la notification de changement d'état émise par un objet géré lorsqu'un ou plusieurs de ses attributs d'état voient leurs valeurs modifiées. Cette notification est normalisée par l'ISO dans le modèle de gestion des états [ISO-10164.2 92]. Elle est émise dès qu'un attribut d'état change de valeur sans tenir compte de la source de ce changement (événement interne ou opération de gestion).

```
stateChange NOTIFICATION
   BEHAVIOUR stateChangeBehaviour;
   WITH INFORMATION SYNTAX  Notification-ASN1Module.StateChangeInfo
   AND ATTRIBUTE IDS
       sourceIndicator          sourceIndicator,
       attributeIdentifierList  attributeIdentifierList,
       stateChangeDefinition    stateChangeDefinition,
       notificationIdentifier   notificationIdentifier,
       correlatedNotifications  correlatedNotifications,
       additionalText           additionalText,
       additionalInformation    additionalInformation;
REGISTERED AS    { joint-iso-ccitt ms(9) smi(3) part2(2) notification(10) 14};
```

FIG. 4.9 - . *La notification de changement d'état*

La notification n'est pas confirmée. L'information associée est de type `StateChangeInfo`. Ce champ comporte la liste des paramètres énumérés dans la partie droite du champ **AND ATTRIBUTE IDS**. Pour chaque attribut représentant une information importante pour la notification (attributs énumérés dans la partie gauche du champ **AND ATTRIBUTE IDS**), le champ de l'information de la notification correspondant est spécifié. Par exemple, si l'attribut `correlatedNotification` est présent dans le module, toute notification de changement d'état émise comporte dans ses paramètres le champ

```
<behaviour-label> BEHAVIOUR
   DEFINED AS delimited-string;
```

FIG. 4.10 - . *Le formulaire de comportement*

`correlatedNotification` qui véhicule la valeur de cet attribut. Si ce dernier n'est pas instancié dans un objet géré, la notification ne comporte pas le champ `correlatedNotifications`. Les attributs et les paramètres de notification liés ne portent pas forcément le même nom. Cependant, pour des raisons de compréhension et d'homogénéité, il est préférable de les faire correspondre le plus souvent possible.

4.6 Le formulaire de comportement

Référencées dans de nombreux autres formulaires, les définitions de comportement permettent d'associer à une étiquette une définition textuelle du comportement relative à un module, une action, une notification, un attribut ou un paramètre comme le montre le formulaire de la figure 4.10.

Ce formulaire de comportement sera étendu dans la suite de cette thèse.

4.7 Corrélation de noms

Afin de mettre en place une organisation hiérarchique des objets gérés au sein d'une MIB, GDMO fournit un formulaire de corrélation de noms. Ce formulaire permet de définir des liens d'inclusion entre classes d'objets gérés et donc de construire des bases d'informations de gestion arborescentes comme nous l'avons vu dans le chapitre précédent.

4.7.1 Le formulaire

La figure 4.11 comporte le formulaire de spécification de corrélation de noms. Définir un lien revient à définir les rôles que prennent les classes qu'il lie. Comme ce lien est d'inclusion, il faut définir la classe englobante et la classe englobée.

Ceci est réalisé à l'aide des champs **NAMED BY SUPERIOR OBJECT CLASS** qui référence la classe englobante et **SUBORDINATE OBJECT CLASS** qui définit la classe englobée. Pour chacune de ces références, l'extension de celle-ci avec le mot-clé **AND SUBCLASSES** implique la validité du lien d'inclusion pour toutes les sous-classes de la classe considérée.

Pour illustrer ce point, prenons l'exemple de la figure 4.12 qui décrit deux graphes d'héritages. Dans le premier, trois classes sont définies (*A, A1* et *A2*). Les classes *A1* et *A2* sont des sous-classes de *A*. Dans le second graphe, quatre classes d'objets gérés sont définies: *B, B1, B2, B11*. Imaginons que toutes ces classes soient instanciables et qu'il existe une corrélation de noms entre *A* et *B* (*B* étant la classe englobante et *A* est la classe englobée). Le formulaire représente les quatre possibilités de contenance en fonction de la présence ou pas de la clause **AND SUBCLASSES** dans l'une ou l'autre des affectations de rôles. Si la clause n'est présente dans aucun des rôles (cas *(1)*), alors seules des instances de *A* peuvent être liées à des instances de *B*. Si la clause est présente dans la définition de la classe englobée (cas *(2)*), alors toute instance de *A* ou d'une de ses sous-classes *A1, A2* peut être liée à une instance de *B* dans

```
<name-binding-label> NAME BINDING
   SUBORDINATE OBJECT CLASS <class-label> [ AND SUBCLASSES];
   NAMED BY SUPERIOR OBJECT CLASS <class-label> [ AND SUBCLASSES];
   WITH ATTRIBUTE <attribute-label>;
  [ BEHAVIOUR <behaviour-label> [,<behaviour-label>]*;]
  [ CREATE  [ create-modifier [,  create-modifier]] [<parameter-label>]*;]
  [ DELETE  [ delete-modifier] [<parameter-label>]*;]
REGISTERED AS <object-identifier>;

create-modifier → WITH-REFERENCE-OBJECT
                | WITH-AUTOMATIC-INSTANCE-NAMING

delete-modifier → ONLY-IF-NO-CONTAINED-OBJECTS
                | DELETES-CONTAINED-OBJECTS
```

FIG. 4.11 - . *Le formulaire de spécification de corrélation de noms*

la MIB. Si la clause **AND SUBCLASSES** n'est attachée qu'à la classe englobante B (cas *(3)*) alors seules les instances de A sont liées à une instance de B ou à une instance de l'une de ses sous-classes *B1, B2, B11* dans la MIB. Le cas *(4)* illustre la situation dans laquelle la clause **AND SUBCLASSES** est présente dans les deux rôles (englobant, englobée). Dans ce cas, toute instance de A ou de l'une de ses sous-classes est liée à une instance de B ou une instance de l'une de ses sous-classes dans la MIB.

La clause **WITH ATTRIBUTE** référence l'attribut de la classe englobée utilisé pour construire le nom distinctif des instances de celle-ci. Le champ **BEHAVIOUR** référence les labels de comportement qui définissent les influences de la relation d'inclusion sur le comportement des instances de classes liées par cette relation.

La clause **CREATE** spécifie la manière dont sont initialisés les attributs de noms des instances de la classe englobée. Si, pour *create-modifier*, l'option **WITH-REFERENCE-OBJECT** est spécifiée, une instance de la classe subordonnée peut être initialisée à partir d'un object de référence qui contient les valeurs initiales des différents attributs de la classe. L'option **WITH-AUTOMATIC-INSTANCE-NAMING** indique que le nom de l'instance créée sera généré automatiquement et que celui-ci peut être omis dans l'invocation de l'opération de création. Les paramètres éventuellement associés à la fonction de création `<parameter-label>` sont utilisés pour contenir des informations relatives à des erreurs de traitement de l'opération de création.

La clause **DELETE** permet de spécifier les conditions de destruction d'une instance de la classe englobée dans le cadre de la corrélation de noms. Deux options sont normalisées:

- **ONLY-IF-NO-CONTAINED-OBJECT** indique que l'instance englobante ne peut être détruite que si elle ne contient pas d'instances d'objets englobés,
- **DELETES-CONTAINED-OBJECTS** indique que la destruction de l'instance englobante entraîne la destruction récursive de toutes les instances englobées.

Les paramètres associés à la fonction de destruction ont le même rôle que pour la fonction de création (support d'informations d'erreurs en cas de défaillance de traitement de la fonction de destruction).

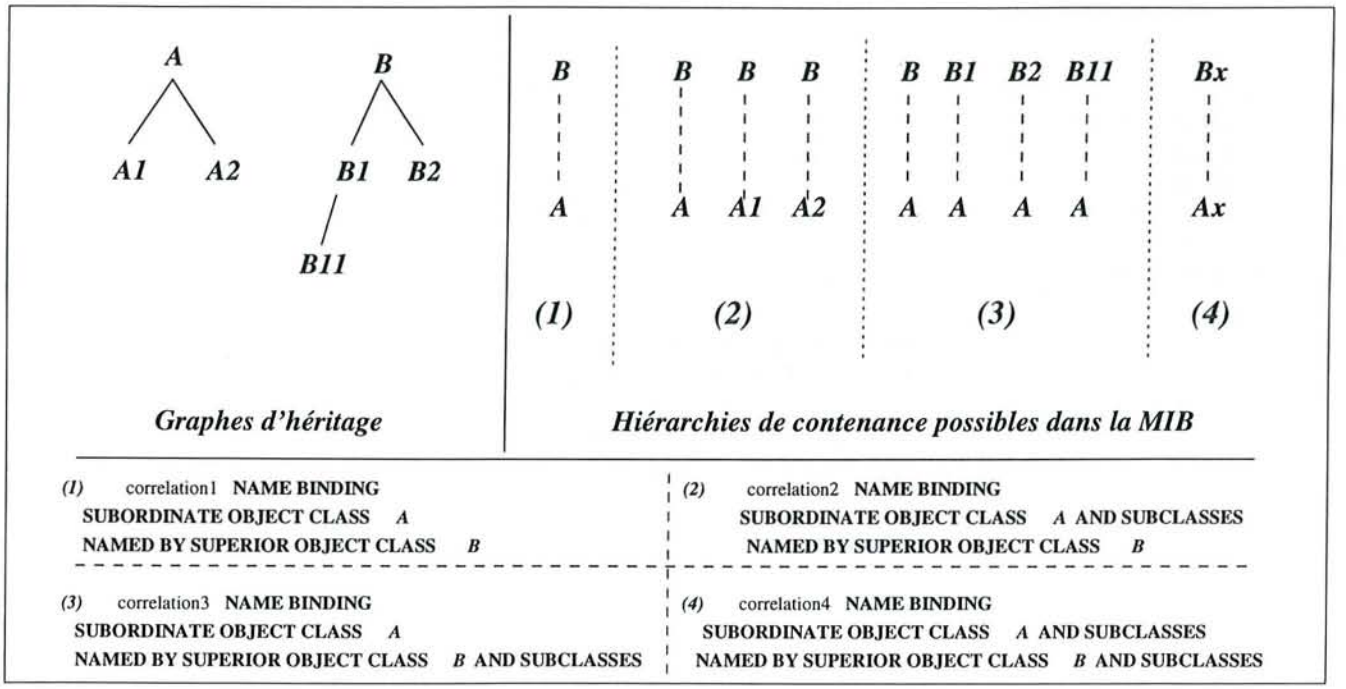

FIG. 4.12 - . *Illustration de la corrélation de noms*

4.7.2 Un exemple

Un objet de la classe `opEquipment` ([CCITT.M.3100 92]) représente un composant d'une ressource physique. La corrélation de noms avec un objet de type `computerSystem` permet au sein d'une MIB de structurer les objets `opEquipment` en les liant aux systèmes informatiques auxquels ils sont rattachés. Ce lien permet donc de connaître pour tout système informatique, la liste de ses équipements physiques.

```
opEquipment-computerSystem NAME-BINDING
  SUBORDINATE OBJECT CLASS opEquipement AND SUBCLASSES;
  NAMED BY
  SUPERIOR OBJECT CLASS computerSystem AND SUBCLASSES;
  WITH ATTRIBUTE EquipmentId;
  CREATE WITH-REFERENCE-OBJECT,
        WITH-AUTOMATIC-INSTANCE-NAMING;
  DELETE ONLY-IF-NO-CONTAINED-OBJECTS;
REGISTERED AS { forum-nameBinding 10};
```

FIG. 4.13 - . *Un exemple de corrélation de noms*

Lors de l'instanciation d'un objet de gestion de type `opEquipment` ou l'une de ses sous-classes, la corrélation de noms va permettre la liaison entre cette instance et une instance de type `computerSystem` ou toute sous-classe.

L'identification d'une instance d'équipement se fera au travers de l'identificateur du système informatique et de l'attribut `EquipmentId` de l'instance de la classe équipement. La création d'une instance au

travers de cette corrélation de noms peut utiliser une instance d'équipement comme objet de référence pour l'initialisation des attributs. Le nommage de l'instance peut se faire de façon automatique. Dans le cadre de la corrélation de noms, une instance d'équipement ne peut être effaçée que si elle ne contient pas d'autres instances d'objets. Cela signifie que l'instance ne doit être supérieure dans aucune autre corrélation de noms.

4.8 Limites du langage

Le langage GDMO propose un ensemble de neuf formulaires pour la spécification formelle des objets gérés d'une base d'informations de gestion. Le langage ne comporte aucun formalisme pour la spécification du comportement des objets gérés. Cette absence a pour conséquence de laisser toute liberté au concepteur d'un objet de gestion de spécifier ou non le comportement de son objet. Ceci mène bien souvent à des incomplétudes dans les spécifications.

L'approche modulaire permet de décrire tous les élements d'un objet géré de manière indépendante. Elle est très intéressante vis-à-vis de la réutilisation de spécifications, mais engendre dans la version actuelle du langage un problème majeur vis-à-vis de la formalisation du comportement. En effet, le langage autorise la spécification de parties de comportements associés à différents éléments d'un objet sans permettre la description du comportement au niveau d'un objet de gestion. La description du comportement d'un objet est en conséquence limitée au comportement de ses différents modules. Or, il est fréquent qu'au niveau du comportement, différents modules puissent s'influencer. Cette influence ne peut en fait être décrite nulle part dans une description GDMO.

De plus, le langage GDMO permet de spécifier le comportement d'un élément (action, notification) à différents niveaux et dans différents formulaires. Ce qui autorise de multiples descriptions d'un même élément, et donc l'introduction de contradictions dans la spécification d'un objet géré.

L'approche OSI étant basée sur un héritage strict, la notion de module conditionnel a probablement été introduite pour contourner cette contrainte d'héritage strict sans accompagner cette possibilité de règles d'utilisation de ces modules. Ceci a pour conséquence de permettre à tout concepteur de ne pas utiliser du tout l'héritage en ne jouant que sur les modules conditionnels pour spécifier ses objets gérés. Ce qui contredit l'idée originale de l'approche. De plus, l'approche permet l'extension d'actions par adjonction de nouveaux paramètres. Or, aucun élément du langage GDMO ne permet de décrire une telle extension ce qui a pour conséquence de ne pas favoriser son utilisation.

4.9 Résumé

Nous avons, dans ce chapitre, présenté les éléments de la syntaxe proposés par le langage GDMO pour la spécification des objets gérés. Dans ce cadre, nous avons présenté de manière détaillée les six formulaires de base qui seront étendus par la suite. Ces formulaires sont celui d'objet géré, de module, d'action, de notification, et de comportement. Nous avons également présenté le formulaire de corrélation de noms qui joue un rôle très important dans l'organisation des bases d'informations de gestion. Les autres formulaires de GDMO sont présentés en annexe.

Intéressant par son indépendance de tout langage de programmation et par sa modularité forte, le langage GDMO forme une bonne base pour la spécification des bases d'informations de gestion dans le cadre OSI. Vu le nombre impressionant de catalogues d'objets décrits en GDMO disponibles, le franc succès du langage ne peut en aucun cas être mis en doute.

Nous avons cependant identifié un certain nombre de problèmes liés à l'approche et au langage qui

ne sont pas résolus à ce jour. Ils concernent principalement la partie comportement du langage, non formalisée à ce jour.

Nous allons dans la deuxième partie de cette thèse, analyser de manière plus précise ces problèmes liés au comportement des objets gérés et proposer une approche formelle pour la spécification de celui-ci dans le cadre de GDMO.

PARTIE II

La formalisation du comportement

«Une théorie n'émerge qu'au sein de passionnés.»

Robert Doisneau.

Cette partie comporte la présentation et la motivation des extensions proposées au formalisme normalisé GDMO en vue d'y intégrer des mécanismes de description formelle du comportement des objets gérés. Elle comporte une étude détaillée des problèmes liés à GDMO, une étude des différentes techniques de description formelle candidates à la spécification du comportement des objets gérés, une presentation de l'approche retenue ainsi que les extensions que nous y avons apportée. Enfin nous présenterons l'intégration de la technique retenue pour la description formelle du comportement dans le langage GDMO.

Chapitres

1. **La définition et la restriction du comportement des objets gérés**
2. **Les approches existantes pour la formalisation du comportement**
3. **Le choix de CRS**
4. **L'extension de CRS par les mécanismes d'héritage**
5. **La formalisation du comportement dans GDMO**

1
La définition et la restriction du comportement des objets gérés

Ce chapitre fixe les bases de la spécification formelle du comportement des objets gérés dans l'approche OSI. Il comporte une étude détaillée des différents problèmes qui apparaissent dans le langage GDMO pour le traitement du comportement. Pour chaque problème identifié dans cette étude, nous proposerons une solution. Ces solutions induisent à la fois des restrictions sur le comportement, et des exigences envers les techniques de description formelle candidates. Un résumé de l'étude présentée ici est donné dans [Clemm 93b]. Ces exigences une fois identifiées pourront servir de base au formalisme que nous proposerons.

1.1 Introduction

Le besoin de formaliser le comportement associé aux objets gérés croît avec la complexité et le nombre d'objets proposés [Kilov 92]. L'étude conjointe de nombreux catalogues d'objets gérés existants, ainsi que des concepts et mécanismes permettant la description du comportement dans GDMO, fait apparaître cinq problèmes qui compliquent la mise en place d'un formalisme pour la spécification du comportement. Ces problèmes ne peuvent être que partiellement résolus par l'introduction d'un formalisme pour le comportement dans GDMO. Afin de les résoudre tous, il faut en plus proposer une méthode d'élaboration d'une spécification de comportement. Ces problèmes sont:

1. **le contenu:** le comportement est, dans la version actuelle du langage GDMO, spécifié en langage naturel. L'étude de nombreuses définitions a montré que le formulaire de comportement servait très souvent aux concepteurs de fourre-tout dans lequel était décrit tout ce qui ne pouvait être spécifié ailleurs. Pour éviter cela, en plus d'un formalisme pour la description du comportement, il faut extraire des définitions existantes d'autres informations significatives qui ne concernent pas directement le comportement. Il faut ensuite analyser ces informations et mettre à disposition des concepteurs de spécifications GDMO des mécanismes leurs permettant d'inclure ce type d'informations dans leurs spécifications. Ceci doit permettre de garantir le maintien d'une séparation entre ces différents types d'informations.

2. **la dispersion:** GDMO permet l'éclatement de la description du comportement en permettant à chaque spécification d'élément de classe, d'une part de contenir sa propre définition de comportement, et d'autre part de redéfinir ce comportement dans un autre formulaire. Elaborer une spécification homogène nécessite donc à la fois des règles précises qui définissent quel comportement doit être spécifié à quel endroit et des mécanismes d'aggrégation des différentes parties de

comportement. Ces mécanismes doivent permettre de dériver le comportement général d'un objet géré. De tels mécanismes ne sont pas proposés dans la norme actuelle.

3. **l'héritage:** l'approche OSI permet à un objet géré d'hériter les caractéristiques d'une ou plusieurs super-classes. Permettre l'héritage dans le cadre du comportement nécessite la définition de règles strictes sur la manière dont le comportement associé à une classe d'objets gérés peut être étendu dans une sous-classe. Comme l'approche OSI autorise également l'héritage multiple, il faut fixer les contraintes qui régissent ce mécanisme pour le comportement de la sous-classe.

4. **les influences extérieures:** un objet géré ne peut jamais être considéré en isolation au sein d'une MIB. Dans de nombreux cas, son comportement est influencé par d'autres objets gérés présents dans la base. Il n'existe dans la norme actuelle aucun moyen de décrire le comportement commun d'objets liés de manière indépendante. Ceci a pour conséquence la complication de la spécification du comportement d'un seul objet.

5. **les interfaces:** Un objet géré peut avoir plusieurs interfaces. Or, seule l'interface de gestion est prise en compte dans le langage GDMO. Il peut être utile pour différentes raisons (comme par exemple la spécification des interactions entre objets) de vouloir spécifier d'autres interfaces que celle de gestion, ceci plus spécialement à un moment donné de l'étape de développement d'une base d'informations de gestion. De plus, pour permettre de spécifier le comportement d'un objet géré sous la forme de l'ensemble des interactions avec le monde extérieur, il est nécessaire de spécifier toutes ses interfaces.

Le formalisme GDMO comporte d'autres ambiguïtés qui n'influencent cependant pas le traitement du comportement d'un objet géré. La principale difficulté à ce niveau est la manière dont sont spécifiées les conditions de présence des modules optionnels. En effet, les conditions de présence de modules optionnels y sont exprimées en langage naturel et risquent d'introduire des inconsistances et imperfections dans la spécification des objets gérés. Nous allons proposer pour ce problème spécifique un formalisme basé sur les prédicats de la logique du premier ordre. Celui-ci sera présenté dans le chapitre 4 de cette partie.

Nous allons dans la suite de ce chapitre analyser de manière détaillée les cinq problèmes décrits ci-dessus et y proposer des solutions. Celles-ci nous mènerons à l'identification de contraintes envers les techniques de descriptions formelles candidates à la formalisation du comportement dans GDMO. Ces techniques seront étudiées par la suite.

1.2 Le problème du contenu

Suite à l'étude de différents catalogues d'objets, il apparait que les descriptions de comportement associés sont très fournies et comportent différents types d'informations qui ne sont pas tous liés au comportement. Les différents types d'informations recensés sont:

- le **comportement** regroupe des descriptions sur la manière dont l'objet géré doit se comporter. Ces descriptions sont utiles aux développeurs d'objets gérés car elles contiennent toute la partie dynamique de la modélisation de l'objet. La spécification des conditions d'émission d'une notification est un exemple type d'une description de comportement.

- la **documentation** regroupe toutes les définitions sémantiques trouvées dans une spécification. Par exemple, la description suivante d'une action est considérée comme documentation: *"Cette action permet à un système de gestion d'activer la ressource "*. Ce type d'informations est utile aux

concepteurs d'objets de gestion et aux exploitants de systèmes de gestion car il apporte une description du sens de l'élément spécifié. Pour ne pas se confondre avec la description du comportement la documentation doit en être séparée.

- l'**expertise** concerne tout type d'informations recensant des liens entre différents aspects du comportement. Cette information descriptive définit des lois naturelles sur les objets gérés. Très utile aux systèmes de gestion, elle permet la mise en place de systèmes experts en vue d'une administration automatisée qui repose sur des systèmes à base de connaissances. Un exemple d'un tel type d'informations pourrait être la phrase suivante: *"Si l'état opérationnel d'un circuit est non actif, alors l'état du circuit composite l'est également".*

La présence de ces différents types d'informations dans la même description sous le label comportement complique énormément le travail des développeurs qui se perdent dans la masse d'informations et ne peuvent pas toujours en extraire les aspects importants liés directement au comportement de l'objet géré.

Or tout n'est pas formalisable. Envisager une formalisation du comportement nécessite de séparer ces différents types d'informations pour ne formaliser que la partie dite de **comportement** de la spécification.

Ce problème est certainement le plus simple à résoudre. En effet, à partir du moment où la description du comportement des objets gérés est faite de manière formelle, seuls les aspects strictement liés au comportement peuvent y apparaître. Le formalisme élimine donc automatiquement toute possibilité de mélange des informations.

Cependant, il ne faut pas considérer les informations non relatives au comportement comme des informations superflues. Au contraire, celles-ci sont utiles à la fois aux développeurs des agents de gestion lors de l'implantation des objets gérés, et aux exploitants de systèmes de gestion (gestionnaires). Ces derniers profitent de ce type d'informations pour acquérir une meilleure connaissance de l'objet qu'ils gèrent.

Dans le but de permettre à tout concepteur d'objet géré de garder une description des aspects documentaires et d'expertise de leurs objets de gestion, un mécanisme simple basé sur une séparation du formulaire de comportement en plusieurs entités distinctes et spécifiques à chaque type d'informations est facilement réalisable. Cette approche a également été préconisée dans [NMF 92] dont la clarté des spécifications en tire profit dans les différents catalogues.

Nous avons retenu cette approche pour isoler le comportement des autres types d'informations utiles dans la description d'un objet géré. Les implications de cette approche sur le langage GDMO sont décrites dans le chapitre 4 de cette partie.

1.3 Le problème de la répartition

Le langage GDMO permet de spécifier des parties de comportement dans les formulaires de module, d'attribut, d'action, de notification et de paramètre (voir figure 1.1). Pour chacun de ces formulaires, nous allons étudier quelle partie du comportement il doit supporter. Cette étude nous permettra de considérer les différents formulaires de comportement en isolation et de proposer un mécanisme de regroupement de ces différentes parties du comportement.

Le concept de base défendu et mis en place ici est l'interdiction de décrire n'importe quoi, n'importe où. Cette idée très simple. n'est pas présente dans la norme actuelle.

1.3.1 Les différents formulaires de comportement

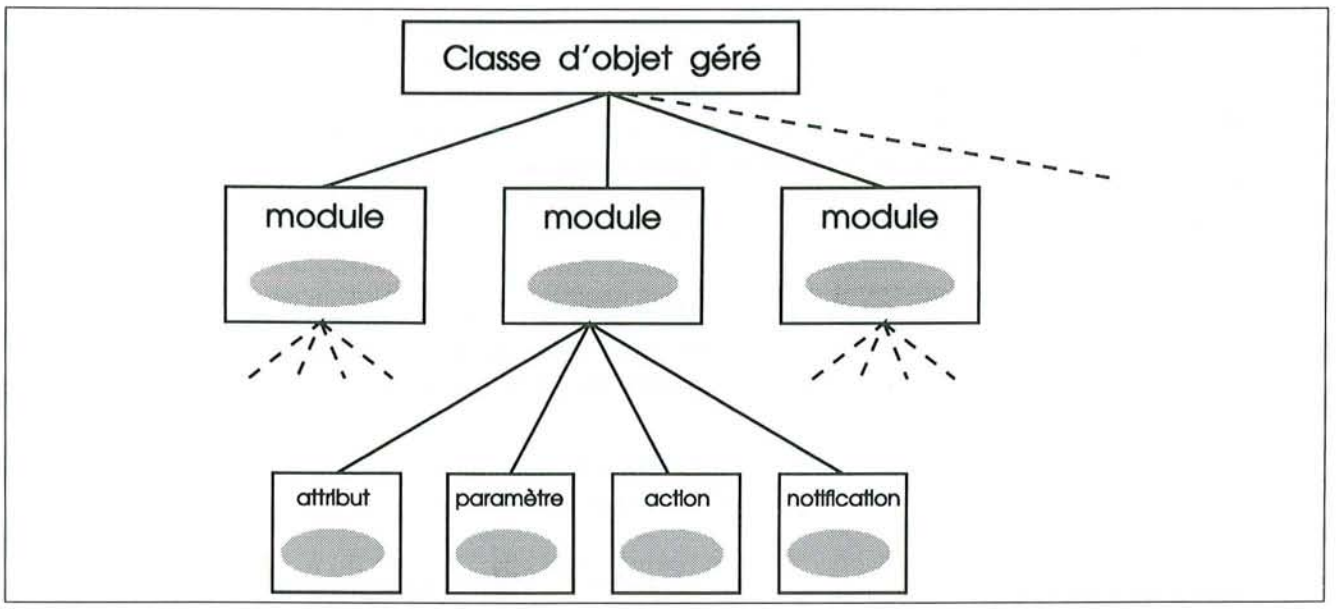

FIG. 1.1 - . *Formulaires entrant dans la composition du comportement d'un objet géré*

Comportement d'attribut

Le comportement associé à un attribut doit décrire la manière dont les opérations de comparaison doivent être appliquées à celui-ci. Il comprend également toute information générique à cet attribut, i.e. uniquement le comportement vérifié dans toute instance de celui-ci. Comme un attribut est inclus dans un module, l'influence du module sur le comportement de l'attribut doit figurer dans le formulaire de comportement associé au module.

Le lien entre l'attribut et les opérations de comparaison se faisant au niveau de l'attribut, il est possible de définir le comportement de celles-ci dans la partie comportement de l'attribut.

Comportement d'action

La définition du comportement associée à une action doit fournir la spécification des conditions dans lesquelles l'action peut être invoquée et les effets engendrés par son exécution. Une spécification peut être faite dans le formulaire d'action, de module ou d'objet géré. Au niveau du formulaire d'action, on ne peut spécifier que les aspects génériques du comportement, i.e. les conditions générales de déclenchement et les effets applicables dans tout objet qui recense cette action.

Une description du comportement d'action peut être spécialisée dans un module, mais ne doit en aucun cas être traitée ailleurs (formulaire de paramètre ou d'attribut par exemple).

La pré-condition et la post-condition associées à une action sont le plus souvent liées aux attributs présents dans le module référençant l'action. Aussi, nous pensons qu'il est préférable de spécifier le comportement d'une action dans le cadre du module dans lequel elle est incluse. La partie comportement spécifique à l'action ne devrait comprendre qu'une définition du profil de celle-ci et des paramètres qu'elle véhicule.

Comportement de notification

Le comportement associé à une notification décrit les conditions dans lesquelles la notification est émise ainsi que les informations véhiculées par ses paramètres.

Les conditions d'émission d'une notification sont toujours liées aux attributs de l'objet de gestion dans lequel cette notification est incluse. Dans le but de maintenir une certaine indépendance entre les différentes spécifications, il est préférable de différer la description du comportement de la notification. Ceci jusqu'au moment ou le spécifieur connait l'environnement dans lequel il désire inclure cette notification, c'est à dire dans un module, voire un objet.

Comportement de paramètre

Le comportement associé à un paramètre est le plus curieux du langage GDMO. Conçu pour décrire la sémantique d'un paramètre, il n'a été utilisé dans aucune spécification à ce jour. Ceci est compréhensible, car les liens entre paramètres et valeurs sont spécifiés ailleurs dans le formulaire. Alors pourquoi l'avoir défini dans la norme? Probablement pour permettre d'y inclure des informations sur la sémantique du paramètre et non des expressions de comportement.

Dans le but de clarifier la répartition du comportement, nous n'avons dans notre approche prévu aucune description de comportement au niveau d'un paramètre. Seules des informations générales y sont autorisées.

Comportement de module

Le comportement associé à un module a, dans le cadre de l'approche, une importance toute particulière. En effet, c'est dans les modules que sont regroupés les attributs, actions, notifications et paramètres. C'est donc au sein du comportement du module que doivent être décrites les influences entre les différents éléments. Ces influences sont:

- les dépendances entre valeurs d'attributs,
- les liens entre les valeurs d'attributs et les pré/post-conditions des actions et des opérations.
- les conditions sur les attributs qui déclenchent l'émission de notifications.

Un module décrit une entité indépendante comportant des attributs des actions et des notifications. Losrqu'un objet intègre plusieurs modules, les interférences entre ces modules doivent, d'après la norme, être décrites dans le comportement des modules concernés, ce qui contredit la notion d'indépendance associée au module.

Toujours dans le but de rester fidèle aux concepts de modularité des spécifications GDMO, il semble préférable de décrire les interférences entre différents modules au sein d'un objet géré dans le comportement de l'objet lui-même. Dans notre approche, nous avons retenu cette solution. Et comme la norme actuelle ne prévoit pas la disponibilité d'un formulaire de comportement d'objets gérés, nous en avons ajouté un.

1.3.2 Le concept de collecte de comportement

Il ressort de l'étude des différents types de comportement entrant dans la spécification d'un comportement d'objet que, non seulement la dispersion des informations dans la norme n'est pas en harmonie

avec les concepts de modularité du langage, mais qu'aucun mécanisme n'est proposé pour construire le comportement d'un objet géré à partir du comportement de ses éléments.

Le traitement de la répartition du comportement d'un objet dans les différents formulaires qui le composent peut être envisagé de deux manières. Nous allons dans les lignes suivantes les présenter rapidement.

Approche simple

La première et la plus triviale consiste à interdire aux spécifieurs de définir des clauses de comportement ailleurs que dans les objets gérés eux-mêmes. Cette solution quelque peu expéditive, permet cependant d'avoir une description homogène du comportement pour chaque objet sans aucune manipulation particulière. Elle a cependant le désavantage de forcer les spécifieurs à ne plus profiter du concept de modularité forte proposé par GDMO qui permet de définir des opérations, actions, modules indépendamment de tout objet géré. Cette solution n'est donc certainement pas la plus appropriée pour traiter la distribution du comportement au sein d'un objet géré et ne sera pas retenue dans l'approche proposée dans cette thèse.

Approche structurée

Mettre en place un mécanisme fidèle à l'approche OSI pour construire le comportement d'un objet géré à partir des différentes descriptions de ses composants implique la nécessité de définir un algorithme de collecte qui pourra dériver le comportement d'un objet géré à partir des différentes descriptions. Afin d'éviter tout problème de conflit de description (redéfinition d'une action dans un module par exemple) deux solutions sont envisageables:

1. définir pour chaque formulaire de comportement spécifique les limites de son champ d'action pour éviter tout conflit lors de l'élaboration du comportement global de l'objet géré.
2. laisser libre la répartition des informations relatives au comportement dans les formulaires et mettre en place un mécanisme de priorités de prise en compte de celui-ci en cas de conflit.

Dans le cadre de l'approche retenue dans cette thèse, la première solution de l'approche structurée a été retenue. Ce choix a en effet plusieurs avantages:

- Il implique la nécessité de fournir une méthode de spécification aux concepteurs d'objets gérés et par là même, facilite le processus de modélisation du comportement et permet d'appréhender sa formalisation d'une manière moins globale. En résumé, cette solution apporte de la rigueur au développement.
- De par la définition claire du contenu des différents éléments, tout risque de contradictions dans la description du comportement est écarté.
- Les règles de priorité éventuellement fournies dans la seconde solution de l'approche structurée, c'est à dire une précendence sur les comportements (ex. si le comportement d'une action est défini dans un module et dans le formulaire d'action, seul le comportement défini dans le module est considéré comme valide), ne sont pas faciles à définir et n'évitent en rien l'introduction de contradictions ou d'incohérences dans la spécification. On peut par exemple, spécifier dans un module, un comportement diamétralement opposé à celui spécifié dans le formulaire d'action et par le mécanisme de précedence changer complètement un comportement ce qui ne constitue pas une solution viable.

Pouvoir mettre en place un algorithme de collecte de comportement à partir de la solution retenue nécessite la définition de règles strictes sur le contenu des différents formulaires. Dans la section précédente traitant du contenu des descriptions de comportement, nous avons abordé de manière générale ce problème. Aussi, à partir de cette étude, nous pouvons énoncer les contraintes sur le contenu qui permettront pour chaque formulaire de limiter l'information qu'il peut contenir et forcer le spécifieur à respecter une certaine discipline lors de la description du comportement.

Ces règles de spécification sont:

- le comportement d'attribut ne doit spécifier que la manière dont les opérations de comparaison s'appliquent sur celui-ci. Toute autre description ne peut être spécifiée à ce niveau, et donc dans ce formulaire.
- le comportement de paramètre ne doit contenir aucune description et devient inutile pour un langage formel.
- le comportement d'une action ne doit inclure qu'une description de l'action en termes de nom, de paramètres d'invocation et de réponse.
- le comportement d'une notification ne doit contenir qu'une description de celle-ci en terme de nom et de paramètres. Les conditions de déclenchement étant liées aux attributs, celles-ci ne peuvent être spécifiées à ce niveau.
- le comportement associé à un module doit spécialiser la manière d'agir de ses composants, i.e. attributs, actions, opérations et notifications. Dans ce cadre, le comportement du module doit décrire les liens entre les différents attributs, les relations entre les valeurs de ces derniers sur le comportement des actions et les conditions d'émission des notifications. La spécification du comportement de module devient donc prépondérante dans notre processus de développement.
- le comportement associé à un objet de gestion ne doit contenir qu'une spécialisation des comportements des différents modules qui le composent en décrivant leurs interférences.

La définition de telles règles permet de résoudre les conflits de comportement de manière horizontale c'est à dire entre les différentes spécifications de comportement au sein d'un même objet géré (intégration du comportement d'action respectivement notification dans celui d'un module, couplage des comportements de modules et intégration de ces comportements dans celui de l'objet géré). Ces règles permettent d'avoir une description homogène du comportement d'un objet et apportent une méthodologie de spécification.

Basé sur le respect de ces règles, il est également possible de concevoir un algorithme de collecte du comportement au sein d'un objet géré. L'algorithme utilisé dans le cadre de notre méthode sera présenté dans le chapitre 5 de cette partie.

Le comportement d'un objet géré n'est pas uniquement obtenu par construction de ceux de ses composants, mais est également influencé par les caractéristiques qu'il hérite de sa ou ses super-classes. Cet aspect de construction de comportement que nous appellerons vertical est abordé dans la section suivante.

1.4 Le comportement et l'héritage

Comme l'approche OSI est orientée-objets, le langage GDMO comporte un mécanisme d'héritage permettant à une sous-classe d'hériter toutes les propriétés de sa ou ses super-classes. Nous allons dans

cette section étudier les influences de l'héritage sur le comportement d'un objet géré. Dans ce cadre, nous allons dans un premier temps, présenter les concepts d'héritage retenus dans l'approche OSI et leurs influences sur les données c'est-à-dire sur les attributs des objets gérés et, par la suite, étudier la manière dont ces concepts s'appliquent au comportement d'un objet géré.

1.4.1 Les contraintes de l'héritage sur les données

L'approche objet retenue dans la norme OSI pour la modélisation des ressources inclut le concept d'héritage multiple. Or, plusieurs interprétations de l'héritage multiple existent. Elles différent sur la résolution des conflits d'héritage (héritage d'un même attribut de plusieurs super-classes par exemple). Une seule interprétation a été retenue dans l'approche OSI:

a: si une sous-classe hérite un même attribut de plusieurs super-classes, cet attribut doit être de même type dans toutes les super-classes. Dans ce cas, l'héritage est équivalent à un héritage de l'attribut d'une seule classe ayant les caractéristiques suivantes:

- l'ensemble des valeurs permises est égal à l'intersection des ensembles des valeurs permises dans chaque classe,
- la liste des valeurs requises est égale à l'union des valeurs requises pour chaque classe.

L'attribut résultant doit avoir une liste des valeurs requises incluse dans l'ensemble des valeurs permises.

b: si une sous-classe hérite d'une super-classe un attribut existant déjà dans la sous-classe, les valeurs requises de l'attribut dans la sous-classe doivent être une extension des valeurs requises de la super-classe. Par contre les valeurs permises doivent être une restriction de celles permises dans la super-classe.

L'exemple de la figure 1.2 présente un cas général de construction de classe à partir des règles d'héritage décrites ci-dessus. Les valeurs requises pour l'attribut `availabilityStatus` de la classe `machineB` sont par héritage, égales à l'union des valeurs de l'attribut dans la sous-classe `machineB` et celles requises pour la super-classe `machineA`. La liste des valeurs permises est réduite à l'intersection de l'ensemble des valeurs permises dans la super-classe et dans la sous-classe. La valeur `offLine(3)` n'étant pas autorisée dans la super-classe, elle ne peut-être supportée dans la sous-classe. L'exemple de la figure 1.2 enfreint la contrainte (b) sur les valeurs permises. La spécification de la sous-classe est ici erronée.

Le second aspect important de l'héritage dans l'approche OSI est la présence de la contrainte d'héritage strict. Elle a une influence indirecte sur les données et est vérifiée de manière implicite dans les règles d'héritage ci-dessus. L'incidence de cette contrainte est beaucoup plus importante au niveau du comportement associé aux objets gérés.

1.4.2 Le traitement de l'héritage strict dans le comportement

L'héritage strict retenu dans l'approche OSI a, en plus d'une influence sur la composition des attributs hérités, une influence sur le comportement de la sous-classe par rapport à sa ou ses super-classes. En effet, la contrainte d'héritage strict présentée dans [Meyer 92], restreint les transformations du comportement d'une sous-classe en émettant les deux contraintes suivantes:

- toute action ou opération applicable dans certaines conditions à toute instance de la super-classe doit également l'être dans la sous-classe et ceci dans les mêmes conditions.

```
machineA MANAGED OBJECT CLASS
...
availabilityStatus  REQUIRED VALUES {powerOff(2)}
                    PERMITTED VALUES {intest(0), powerOff(2), offDuty(4)}

machineB MANAGED OBJECT CLASS
 DERIVED FROM machineA
 ...
 availabilityStatus  REQUIRED VALUES {inTest(0)};
   PERMITTED VALUES {intest(0), offLine(3), offDuty(4), powerOff(2)}

par héritage on obtient:

machineB MANAGED OBJECT CLASS
     ...
 availabilityStatus REQUIRED VALUES {inTest(0), powerOff(2)}
                    PERMITTED VALUES {inTest(0), powerOff(2), offDuty(4)}
```

FIG. 1.2 - . *Un exemple d'héritage complexe sur un attribut*

- tous les effets de l'application d'une action ou opération sur une instance de la super-classe doivent être visibles dans toute instance de la sous-classe.

Ces contraintes informelles impliquent que la sous-classe ne peut être étendue que par adjonction de nouveaux attributs ou extension de l'espace d'état des attributs existants, par l'adjonction de nouvelles opérations de gestion et notifications ou extension de celles existantes par introduction de nouveaux arguments. De plus, aucune caractéristique de la super-classe ne peut être supprimée dans la sous-classe. Ceci signifie que les services fournis par un objet de la sous-classe doivent au moins incorporer ceux de la super-classe. D'un point de vue comportement, l'héritage strict impose que toute invocation d'opération de gestion valable sur un objet de la super-classe doit être supportée par tout objet de la sous-classe dans les mêmes conditions. Ces contraintes se retrouvent dans la définition de la notion de compatibilité entre objets gérés (voir [ISO-10165.1 92]). Si l'on considère une action en termes de pré- et post-conditions cela signifie que pour toute opération de la super-classe, sa pré-condition doit être affaiblie et sa post-condition renforcée dans la sous-classe. Dans notre approche, nous avons retenu cette contrainte pour le traitement de l'héritage strict sur le comportement.

La figure 1.3 illustre la notion de contrainte d'héritage strict sur une action. On remarque que la pré-condition de celle-ci est affaiblie au niveau de la sous-classe dans la pré-condition PRE car celle de la super-classe implique toujours celle de la sous-classe. La post-condition est bien renforcée car celle de la sous-classe implique celle de la super-classe.

1.5 Les influences extérieures

Le comportement d'un objet géré peut donc être spécifié à partir des éléments qui le composent et par héritage des caractéristiques de sa ou ses super classes. Ce comportement peut cependant être influencé au sein d'une base d'informations de gestion par d'autres instances d'objets gérés et par la corrélation

```
machineA MANAGED OBJECT CLASS
...
action;
PRE: a1 = 0;
POST: a1 = 2;

machineB MANAGED OBJECT CLASS
 DERIVED FROM machineA
 ...
action;
PRE: TRUE;
POST: a1 = 2 AND a2 = 10;
```

FIG. 1.3 - . *Un exemple de respect d'héritage strict sur les actions*

de noms retenue pour l'instanciation de l'objet. Nous allons dans cette section étudier cette influence et sa prise en compte dès la conception de l'objet géré.

1.5.1 Le problème

Un objet de gestion existe au sein d'une base d'informations parmi d'autres objets de gestion et interagit avec ceux-ci. Ceci est notamment le cas lorsqu'il existe des relations de type "client" [Meyer 92] ou l'invocation d'une opération sur l'attribut d'un objet engendre un changement d'état chez un autre objet de la MIB. Un exemple de ce type d'interactions est donné dans [Clemm 93b]. Il est fréquent que des objets doivent interagir pour maintenir la consistance de la base. Le cas de la corrélation de nom est relativement facile à traiter car le formulaire de corrélation de noms comporte un formulaire de comportement. Ce dernier permet de spécifier les influences de cette corrélation sur toute instance de la classe contenue comme sur celle de la classe contenante. Cependant, dans la version actuelle, il n'est pas possible de spécifier de manière adaptée, les interactions entre objets gérés qui ne sont pas liés par une corrélation de noms..

1.5.2 Vers une utilisation extensive des relations

Afin d'éliminer la description du comportement d'un lien entre deux objets gérés dans les formulaires de comportement des objets eux-mêmes, il faut pouvoir séparer les descriptions des objets de celles des liens. Dans l'état actuel de la norme, cette séparation n'est pas possible car les liens entre objets gérés sont modélisés par des attributs dans chacun des objets impliqués dans une relation. Il n'existe pas de modèle spécifique ni de formulaire associé pour décrire ces relations indépendamment des objets qu'elles lient.

De nombreux travaux sont en cours sur la définition d'un modèle relationnel entre objets gérés. Ce modèle prévoit notamment la mise en place de formulaires de relations indépendants dans lesquels le comportement de la relation entre objets gérés pourra être décrit. Les travaux les plus significatifs dans ce domaine ont été entrepris dans [Clemm 93a] et [Sibilla 93]. Nous avons dans le cadre de nos travaux retenu la méthode GDMR [1] définie dans [Clemm 93a] pour la spécification des relations entre objets gérés

1. . Guidelines for the Definition of Managed Relationships

à l'aide de formulaires spécifiques. Cette approche nous paraît la plus consistante et la mieux adaptée pour l'intégration dans GDMO.

1.6 Le problème des interfaces

Dans la plupart des approches orientées-objets, le comportement d'un objet peut être défini comme l'ensemble des interactions et de leurs paramètres visibles à l'interface de l'objet. Dans ce cas, un objet peut être considéré comme un automate actif sur lequel des états peuvent être changés ou lus au travers d'une interface clairement définie. La spécification de l'objet géré se résume donc à la spécification de cette interface, c'est à dire à la description de l'ensemble des suites d'interactions possibles entre l'objet géré et son environnement.

Dans l'approche OSI, une seule interface par objet est normalisée. Celle-ci, dénommée **interface de gestion**, décrit les actions, les opérations applicables sur l'objet et les notifications que l'objet peut émettre. Or, tout objet géré possède d'autres interfaces non prises en compte dans la spécification de l'objet. Ceci empêche de considérer le comportement de l'objet comme un ensemble d'interactions observables. Ces interfaces additionnelles sont (voir figure 1.4):

- **interface de ressource**: utilisée par l'objet géré pour valider sur la ressource les opérations de gestion invoquées sur ses attributs. Elle est également utilisée par la ressource pour affecter des attributs de l'objet géré dans le cas d'un changement d'état de la ressource.

- **interface objets (MOIF[2])**: au travers de laquelle l'objet géré met à disposition d'autres objets de la MIB des opérations permettant d'influencer son comportement.

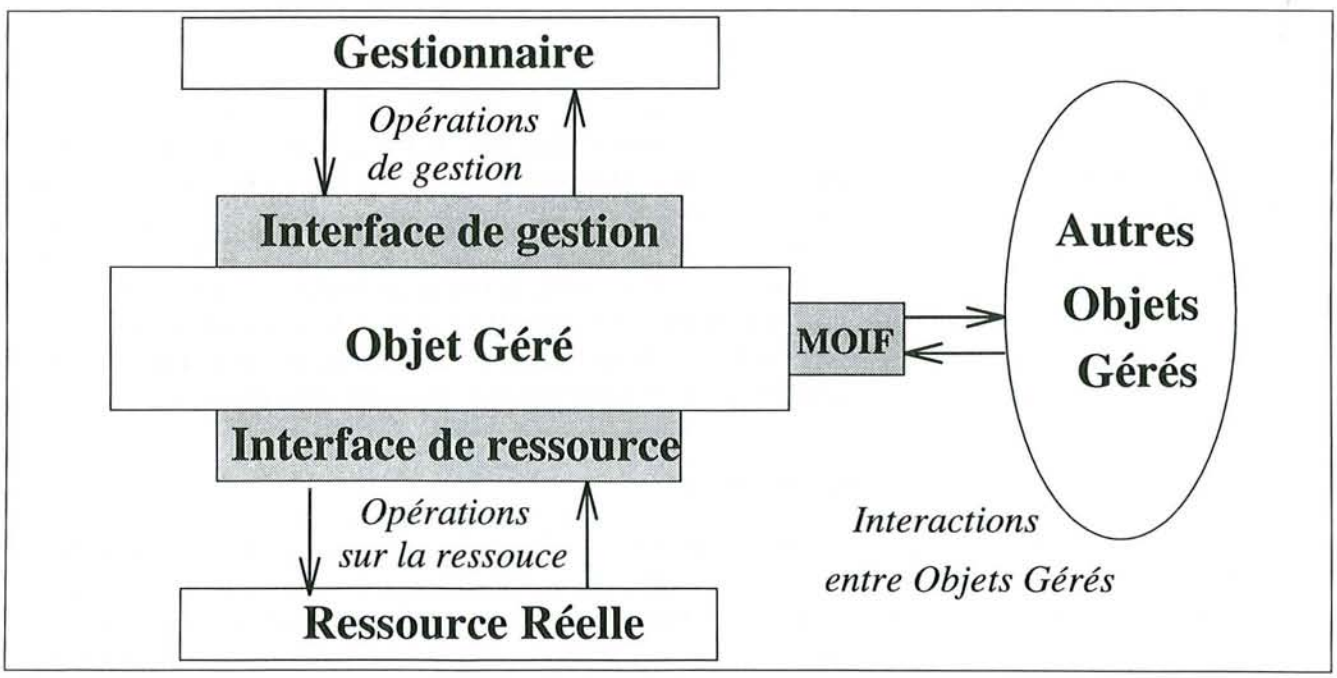

FIG. 1.4 - . *Les différentes interfaces d'un objet géré*

2. . Managed Object InterFace

Tout objet géré ne possède pas nécessairement ces trois interfaces. En effet, seule l'interface de gestion existe dans tous les objets gérés. Seuls les objets qui modélisent une ressource réelle contiennent une interface de ressource, et seuls les objets dont le comportement est influençable par d'autres instances d'objets dans la MIB (c'est à dire des objets en relation avec d'autres objets) possèdent une interface vers ces objets.

1.7 Les exigences

L'intégration dans GDMO, d'un formalisme pour la description du comportement doit satisfaire aux exigences issues des contraintes présentées dans les sections précédentes de ce chapitre c'est à dire l'héritage strict, la répartition du comportement ainsi qu'un mécanisme pour les interfaces. Cette intégration doit en plus saisfaire des contraintes personnelles que nous nous sommes fixées sur la compatibilité avec GDMO et la simplicité d'utilisation, à savoir l'intégration du comportement formel dans la notation et une expressivité pour la spécification du comportement adaptée au contexte. Nous allons dans cette section présenter plus en détail ces contraintes. Ces besoins formeront le cahier des charges à respecter par toute technique de description formelle candidate à l'intégration dans GDMO.

1.7.1 Un support objet

L'approche OSI etant basée sur une approche orientée-objets pour la spécification des objets gérés et le langage GDMO proposant des mécanismes pour la description de telles relations d'héritage, il est absolument nécessaire que toute technique utilisée pour la spécification du comportement d'objets gérés comporte un mécanisme d'héritage identique à celui de l'approche OSI, c'est à dire un héritage strict. Cette exigence est la plus évidente de toutes celles formulées dans cette section.

1.7.2 Support d'ASN.1

Tout type d'attribut d'objet géré, de paramètre d'action ou de notification est spécifié dans GDMO en ASN.1. De plus, le langage ASN.1 est très largement utilisé dans le monde des télécommunications, notamment pour la spécification des paramètres des primitives de service et des unités de données des protocoles. Ceci est également le cas pour le protocole CMIP utilisé pour le transport d'informations relatives aux objets gérés, entre un gestionnaire et un agent de gestion. Pour rester fidèle aux normes existantes et pour limiter le nombre de formalismes manipulés, la technique utilisée pour la spécification du comportement dans GDMO doit pouvoir manipuler des définitions ASN.1. Cela signifie qu'en plus du support de définition de types de données en ASN.1, la technique de description formelle doit également fournir des opérateurs permettant la manipulation de variables dont les types sont décrits en ASN.1.

1.7.3 Une distribution du comportement

Comme nous l'avons vu dans les sections précédentes, il est possible dans GDMO de répartir la définition du comportement d'un objet géré sur les différents formulaires qui le composent. Dans ce cadre, nous avons fixé des règles sur la localisation des informations et proposé la définition d'un algorithme de collecte du comportement qui permet à partir des différents blocs, de construire le comportement général d'un objet géré.

Toute technique candidate à la formalisation du comportement dans GDMO se doit de proposer un mécanisme de description non monolytique du comportement d'un objet géré. Elle doit permettre

une répartition de la description du comportement et proposer une méthode influençant la localisation des informations et mettant à disposition des concepteurs d'objets gérés un algorithme de collecte et d'agrégation du comportement. Cette exigence est certainement la plus restrictive de toutes celles que nous envisageons.

1.7.4 Un mécanisme pour les interfaces

De manière conceptuelle et dans la plupart des approches orientées-objets, un objet ne comporte qu'une interface au travers de laquelle sont invoquées les opérations sur celui-ci. Or, comme nous l'avons vu précédemment, un objet géré peut comporter plusieurs interfaces distinctes.

La technique formelle utilisée pour la spécification du comportement doit donc fournir des possibilités de décrire plusieurs interfaces pour un objet donné. Ceci pouvant se faire par l'affectation d'interactions à des types d'interfaces données. Ceci se traduit en fait par l'instauration d'un mécanisme de vues sur un objet géré. À chaque vue (un ensemble d'interactions) on associera un type d'interface et l'on pourra ainsi définir pour chaque élément externe à l'objet géré considéré (gestionnaire, autre objet géré, ressource) quelles sont les opérations de l'objet auxquelles il aura accès.

1.7.5 La simplicité d'utilisation

Nous avons dès l'introduction de la thèse émis le souhait que toute technique de description formelle utilisée pour la spécification du comportement des objets gérés soit relativement simple d'utilisation. Or, nous n'avons jusqu'ici pas défini de critères de simplicité.

Lorsqu'elle est parfaitement maîtrisée, toute FDT[3] est simple d'utilisation. Lorsque celle-ci est utilisée dans un cadre bien précis qui est ici celui du modèle OSI, des critères de simplicité autres que des considérations personnelles entrent en compte. En effet, comme les normes OSI imposent des formalismes pour la spécification des objets gérés, il est naturel que le premier critère de simplicité d'utilisation soit la compatibilité entre les normes et la FDT retenue pour la description du comportement des objets gérés. Cette compatibilité se traduit par une intégration et un support des différents formalismes utilisés (ici ASN.1 et GDMO). Elle permet aussi à tout spécifieur d'objet géré de ne manipuler qu'un seul formalisme et de ne pas séparer, lors de la spécification, la partie informelle et la partie formelle. Ce critère est prédominant pour la simplicité d'utilisation et pour la promotion de la FDT utilisée pour le comportement.

Le second critère important est le support logiciel offert avec la FDT qui est utilisée pour la description du comportement. Il est impératif pour la promotion de l'utilisation d'un formalisme, que le développement de spécifications soit assisté d'un ensemble d'outils facilitant l'élaboration et permettant une exploitation accrue des spécifications dans les différentes phases du développement.

Le dernier critère de simplicité énonçé porte certainement à discussion. En effet, nous souhaitons que la FDT utilisée pour la spécification du comportement dans GDMO, fournisse des mécanismes relativements proches du mode de raisonnement humain afin que la spécification du comportement ne devienne pas la partie insurmontable de la conception d'un objet géré.

En résumé, les trois exigences que nous formulons afin d'accroître la simplicité d'utilisation d'une FDT pour la description du comportement des objet gérés sont:

- la compatibilité avec les formalismes utilisés (GDMO, ASN.1), afin de manipuler un minimum de formalismes et si possible un seul,

3. . Formal Description Technique

- un support logiciel facilitant la conception et la spécification des systèmes,
- et finalement une approche voisine du raisonnement humain.

1.8 Résumé

Dans ce chapitre, nous avons étudié les différents problèmes liés au comportement dans le langage GDMO en vue d'une formalisation de sa description.

L'établissement d'exigences précises pour la formalisation a été possible grâce à une étude détaillée des cinq problèmes de base spécifiques aux objets gérés et au langage GDMO: le contenu, la répartition, l'héritage, les interactions entre objets et la multiplicité des interfaces offertes par un objet de gestion. L'identification des informations utiles à la formalisation ainsi que l'élaboration de stratégies d'agrégation de comportement nous ont permis de traiter les deux premiers problèmes. La clarification des mécanismes d'héritage utilisés dans l'approche nous permet d'envisager son support dans une FDT. L'utilisation du concept de relation entre objets nous permet de séparer la spécification du comportement d'un objet et celle de ses influences sur d'autres objets avec lesquels il est en relation. L'identification de toutes les interfaces possibles d'un objet géré ainsi que la motivation pour spécifier toutes ses interfaces a rendu possible l'élimination du dernier problème.

Ces exigences vont nous servir de critères de choix pour la sélection d'une technique de description formelle à utiliser pour la spécification du comportement des objets gérés dans GDMO. L'étude des différentes techniques proposées à ce jour fait l'objet du chapitre suivant.

2
Les approches existantes pour la formalisation

De nombreux travaux sont en cours pour permettre une formalisation du comportement des agents de gestion. Les normes de gestion de réseaux étant les premières à retenir une approche orientée-objets pour la conception de systèmes, il est naturel que des techniques de description formelle issues du monde des télécommunications et de celui du développement des logiciels soient candidates pour une telle formalisation.

Dans ce chapitre, nous allons présenter les différentes approches proposées à ce jour en présentant brièvement les concepts orientés-objets qu'elles comportent, et en comparant les fonctionnalités avec les besoins énoncés dans le chapitre précédent.

2.1 L'approche VDM

L'approche VDM[1] [Jones 90] a été la première à être proposée comme candidate à la formalisation du comportement des objets gérés. L'approche VDM, orientée-modèles, est basée sur la théorie des ensembles et sur les prédicats de la logique du premier ordre. Un système est spécifié en VDM par ses états et l'ensemble des opérations autorisées sur ses états en termes de pré- et post-conditions. La méthode fournit une notation et des obligations de preuves lors des différentes étapes de la spécification. L'approche initiale ne contient pas de concepts orientés-objets. Cependant, des travaux entrepris ces dernières années sur ce domaine ont permis l'adaptation de VDM à la spécification de systèmes orientés-objets comme nous allons le voir dans la section suivante.

2.1.1 VDM et l'héritage

VDM n'offre pas directement le concept d'héritage mais inclut celui très proche de satisfiabilité [Jones 90]. En effet, cette dernière est définie dans VDM comme une relation entre deux spécifications. Cette relation est définie comme suit:

Une spécification A satisfait une spécification B si et seulement si toutes les opérations définies dans B sont applicables sur A dans au moins les mêmes conditions. De même si tous les effets des opérations définies dans B sont au moins aussi stricts dans A, c'est à dire que les effets qui doivent être vérifiés dans B le sont aussi dans A. La spécification A peut naturellement être étendue par de nouveaux types et de nouvelles opérations.

1. Vienna Development Method

Comme les opérations sont spécifiées en VDM en termes de pré- et post-conditions, ceci se traduit au niveau d'une spécification par la nécessité d'alléger les pré-conditions et de renforcer les post-conditions sur les opérations de la spécification à laquelle on veut être lié par la relation de satisfiabilité.

Cette définition de la satisfiabilité est équivalente au concept d'héritage strict retenu dans l'approche OSI.

2.1.2 VDM et GDMO

L'utilisation de VDM pour la spécification du comportement des objets gérés a été proposée à plusieurs reprises [Simon 92, Marshall 92].

Bien que l'approche propose des mécanismes d'héritage équivalents à ceux retenus par l'OSI, les points suivants ne sont pas traités dans les propositions faites à ce jour:

- la répartition des descriptions sur plusieurs formulaires n'a pas été envisagée, et en conséquence, aucun algorithme de collecte n'a été proposé. L'approche ne considère qu'une description monolythique du comportement d'un objet géré.
- aucun lien entre les types VDM et le formalisme ASN.1 n'a été réalisé et le comportement des objets gérés est spécifié indépendamment des descriptions GDMO.

La non-prise en compte de ces aspects ne facilite pas la construction de la spécification formelle du comportement des objets gérés et oblige les concepteurs à manipuler plusieurs formalismes. Ce fait nous a conduit à ne pas retenir VDM pour la description du comportement des objets gérés.

Cependant, il faut noter que le langage est assez puissant pour envisager une intégration de ces exigences dans le formalisme, et que le langage est en cours de normalisation à l'ISO. Aussi, VDM reste un candidat sérieux au sein des groupes de normalisation.

2.2 Z

Z est un langage de description formelle orienté-modèles, basé sur la théorie des ensembles et sur les prédicats de la logique du premier ordre [Spivey 89]. Le langage de description Z est relativement proche du langage utilisé dans la méthode VDM. Une spécification d'un système en Z consiste en l'énumération des variables d'états du système,de la spécification de son état initial et de la description des opérations possibles en termes de pré-/post-conditions. Tout comme pour VDM, de nombreux travaux ont porté ces dernières années sur l'apport de nouveaux mécanismes orientés-objets à Z. Aussi, allons nous brièvement présenter ces extensions dans les lignes suivantes.

2.2.1 Object-Z

Object-Z [Duke 90, Carrington 89] étend le formalisme Z avec le concept de classe. Cette extension fournit, d'une part un mécanisme syntaxique qui permet de spécifier des formulaires de classes dans des description Z, et d'autre part un ensemble de concepts et une sémantique qui permet de manipuler ces concepts orientés-objets.

Le formalisme étendu propose plus particulièrement le support de l'héritage multiple, la notion d'encapsulation par le concept de visibilité des opérations et des données d'une classe ainsi qu'une sémantique basée sur l'historique des occurences d'événements pour un objet donné. Dans ce cadre, le comportement d'une classe est caractérisé par l'énumération du séquencement possible de toutes les occurences

d'opérations. Comme l'étude des pré et post-conditions des différentes opérations d'une classe ne suffit pas pour déterminer les séquencements autorisés, les concepteurs d'Object-Z ont ajouté au formulaire de classes un prédicat de la logique temporelle du premier ordre au travers duquel un concepteur de classes peut spécifier des relations entre les opérations en termes de séquencement.

2.2.2 Z et GDMO

A ce jour, une seule étude a porté sur l'adéquation de Z à la description du comportement des objets gérés [Rudkin 91]. Dans cette étude, le formalisme Z a été utilisé pour décrire la fonction de discrimination et de renvoi d'événements. Dans le cadre de cette spécification, les formulaires de GDMO n'ont pas été pris en compte. Nous pouvons cependant formuler les mêmes voeux que pour VDM, les deux techniques étant relativement proches.

Les reproches que nous pouvons faire à cette approche sont les mêmes que ceux que nous avons formulé pour VDM, c'est à dire:

- la non-prise en compte du langage ASN.1 pour la description des données,
- la spécification du comportement non intégrée à GDMO et donc la manipulation de plusieurs formalismes.

Mais tout comme VDM, l'approche Z ou plus particulièrement Object-Z reste une candidate sérieuse à l'intégration d'un mécanisme formel pour la description du comportement dans GDMO en raison de sa puissance expressive et des possibilités de définir des liens précis entre GDMO, ASN.1 et Z. De plus, il semble que Z soit en bonne voie de normalisation, ayant été retenue comme Technique de Description Formelle dans le cadre des travaux sur l'ODP[2].

2.3 LOTOS

Dans le cadre des Techniques de Description Formelle normalisées par l'ISO, une seule approche pour la spécification des formulaires d'informations de gestion a été proposée à ce jour. Dans [Kouyzer 92], une approche basée sur LOTOS[3] [ISO-8807 87], propose une infrastructure générale pour la spécification d'activités de gestion. Dans le but de présenter quelque peu cette architecture, nous allons tout d'abord présenter comment la notion d'héritage est incorporée dans LOTOS, puis, expliquer l'utilisation de LOTOS pour la description des objets gérés. Nous finirons cette présentation par une confrontation de l'approche proposée aux besoins énoncés précédemment.

2.3.1 LOTOS et l'héritage

Les travaux basés sur le mariage de LOTOS avec les concepts de l'approche orientée-objets ont été motivés par l'émergence de l'approche dans le monde des télécommunications au travers des concepts de l'ODP. Les travaux sur les aspects objet dans LOTOS ont abouti à des extensions du langage.

Dans [Dijkerman 89], le concept d'héritage est modélisé par la composition sous forme d'alternative de processus dénommés super-classes comme l'illustre l'exemple de la figure 2.1. Il est cependant nécessaire que les processus, dont la sous-classe hérite le comportement, ne soient pas récursifs. Ce qui impose

2. . Open Distributed Processing [N109 89]
3. . Language Of Temporal Ordering Specifications

une contrainte sur le style de spécification et ne permet de réutiliser que des parties de spécifications de processus.

```
Q, X1, X2..Xn : processus

Si Q hérite de X1, X2..Xn alors

Q ::= (X1 [] X2 [] .. Xn
   []
   < extensions de Q >
 ) >> accept p1,p2,..,pn in Q(p1,p2,..pn)
```

FIG. 2.1 - . *Composition d'une super-classe par héritage multiple en LOTOS*

Dans le même ordre d'idées, [Black 89] propose d'étendre le langage LOTOS par une clause d'héritage et une clause de comportement organisée sous forme de liste d'alternatives de méthodes.

Si des mécanismes existent pour permettre l'inclusion de l'héritage dans la technique de description formelle LOTOS, l'ensemble des problèmes posés dans ce domaine n'est pas résolu. Citons dans ce cadre l'absence de sémantique de l'opérateur d'héritage dans LOTOS et le non-support de l'héritage sur les données.

2.3.2 L'infrastructure proposée

Nous avons vu ci-dessus qu'il était possible de spécifier des classes d'objets en LOTOS et d'y inclure le concept d'héritage. L'approche proposée dans [Kouyzer 92] pour la spécification du comportement des systèmes de gestion utilise ces concepts pour la spécification des classes d'objets de gestion et propose toute une infrastructure de processus qui permet la modélisation des interactions entre un gestionnaire et un agent comme l'illustre la figure 2.2.

L'infrastructure proposée permet de spécifier un agent de gestion avec sa MIB tel qu'il est perçu par le gestionnaire. Le processus `CMISE` fournit l'interface d'accès au protocole CMIP tandis que le processus `Interface` a pour rôle de convertir ces primitives de service en actions ou opérations et de gérer les tâches complexes de filtrage et de portée.

La définition de l'ensemble des classes d'objets gérés est centralisée dans un processus appelé `classes`. Cette architecture est indépendante des types de classes utilisés. En effet, il est très facile d'y introduire de nouvelles classes.

2.3.3 LOTOS et GDMO

L'approche LOTOS pour la spécification des objets gérés, ne fournit aucune relation avec le langage GDMO. En effet, les aspects de répartition du comportement au sein d'une spécification d'objet ne sont pas pris en compte dans cette approche. Il est donc difficilement concevable de lier les deux langages et d'en attendre un formalisme simple pour la description du comportement. De plus, bien que des liens aient été établis entre le formalisme ACT one et le langage ASN.1 [Thomas 92, Bochmann 89], la technique de description formelle LOTOS n'intègre pas ASN.1 dans son langage. Or, GDMO est basé sur un usage intensif d'ASN.1. Afin d'unifier LOTOS et GDMO sur cette partie données, il faudrait dans un premier temps normaliser l'usage d'ASN.1 dans les spécifications LOTOS.

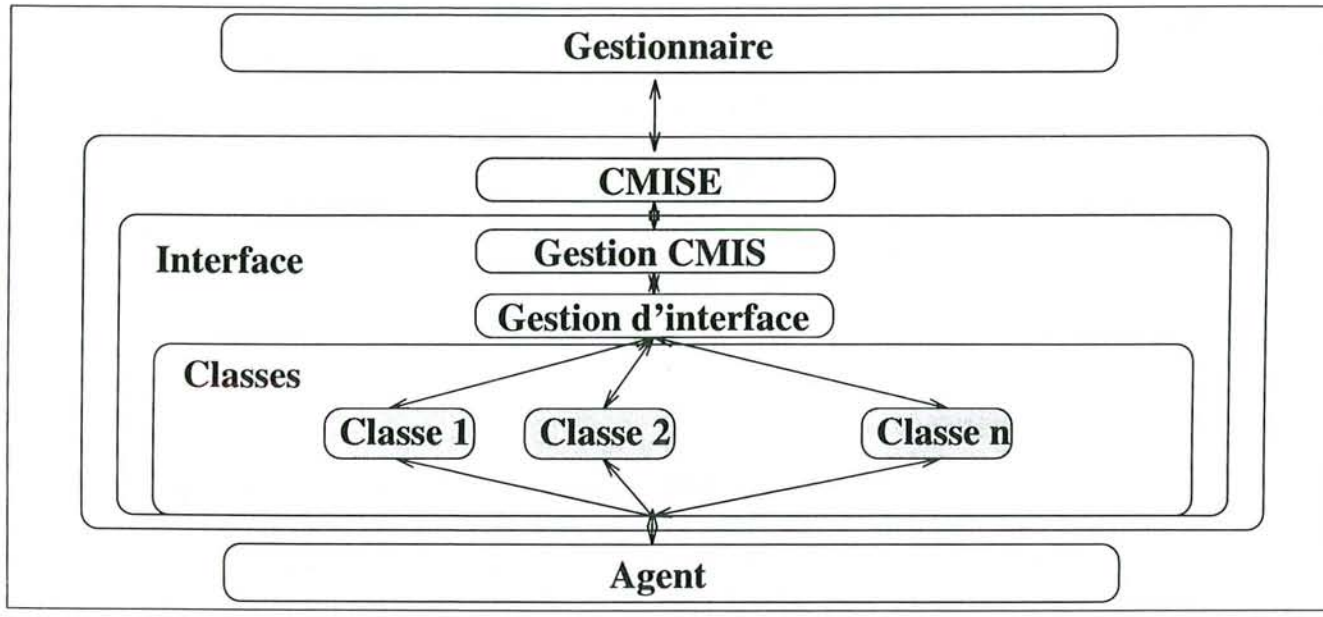

FIG. 2.2 - . *Infrastructure LOTOS d'une spécification de système de gestion*

Bien que LOTOS propose un mécanisme incluant les concepts d'héritage pour la description du comportement, le langage ne se prête pas facilement à une intégration dans GDMO. Ceci est principalement dû à la répartition des informations de comportement dans les spécifications GDMO. Comme pour les techniques déjà évoquées précédemment, la nécessité de manipuler plusieurs formalismes en même temps complique énormément le traitement et engendre des problèmes de consistance de spécifications. Contrairement aux formalismes vu précédemment, il apparait difficile, voir impossible, de marier le formalisme LOTOS à GDMO et donc de supprimer le problème des formalismes multiples. De ce fait, LOTOS ne constitue pas la solution idéale au problème initialement posé.

2.4 ESTELLE

Contrairement à LOTOS, la technique de description ESTELLE[4] [ISO-9074 87] n'a pas été à ce jour présentée comme une technique candidate à la formalisation du comportement dans GDMO. Cependant, des travaux ont été réalisés sur l'adéquation entre ESTELLE et les concepts d'une approche orientée-objets [Sijelmassi 89]. Ces travaux ont principalement abordé les modes de communication. Il n'a pas été établi de lien entre le modèle de comportement d'ESTELLE et le concept d'héritage tel qu'il apparaît dans GDMO. De plus, ESTELLE n'utilisant pas ASN.1 pour la description de ses données mais PASCAL, cette technique n'est pas dans sa structure actuelle une candidate idéale à la spécification du comportement des objets gérés de GDMO.

4. Extended State Transition Language

2.5 SDL

SDL[5] est un langage normalisé par le CCITT[6] [CCITT.Z.100-104 84, CCITT.Z.100-A 88]. SDL est, tout comme ESTELLE, basé sur le concept d'automates finis étendus pour la description du comportement mais construit sur le formalisme ACT.one pour la spécification des données. Récemment, des travaux très intéressants ont porté sur l'adéquation entre SDL et les modèles objets. Notamment ceux menés dans [Moller-Pedersen 90] qui ont conduit à la normalisation d'une nouvelle version du langage appelée SDL'92 ou OSDL[7] [CCITT-Z-100-92 92]. Ce dernier intègre des concepts d'approche objet dans le formalisme. Une bonne introduction à SDL est donnée dans [Turner 93].

2.5.1 OSDL et l'héritage

La technique OSDL fournit des extensions à SDL qui permettent de transformer des définitions d'instances SDL en définitions de types. Ces dernières peuvent alors être utilisées pour générer des instances. De plus, le langage comporte le concept d'héritage qui est défini comme méthode de spécialisation de types. Des mécanismes d'héritage sont fournis pour les principaux éléments du langage SDL. Ces éléments sont les systèmes, les processus, les blocs et les types de services. De plus, les procédures et les signaux sont également considérés comme des types dans la nouvelle version du langage.

Le concept d'héritage dans OSDL est basé sur la notion de spécialisation d'une classe. Un sous-type spécialise un autre type en héritant l'ensemble de ses fonctionnalités et en y ajoutant de nouvelles. La version 92 autorise également le paramétrage des types et propose le mécanisme de module pour encourager la réutilisation. A noter que seul l'héritage simple est considéré dans OSDL.

2.5.2 OSDL et GDMO

Deux approches ont été proposées à ce jour pour combiner GDMO et la technique de description formelle SDL. La première approche [Bartocci 93] propose une méthode de construction de spécifications SDL à partir de spécifications GDMO. L'idée de cette approche est de spécifier un objet géré en deux temps. Dans un premier temps, la spécification GDMO est établie sans comportement formel. Une fois cette description complète, une spécification de cet objet en OSDL est établie et le comportement est formalisé à l'aide des mécanismes de OSDL. Cette approche a le désavantage de maintenir séparées la description de l'objet en GDMO et la spécification de son comportement en OSDL. Cette séparation, bien que virtuelle en raison de l'existence de clauses de mapping de GDMO vers SDL et de ASN.1 vers ACT.one [Karner 93, Fischer 93], oblige le développeur d'objets gérés de manipuler à tout moment au moins deux formalismes différents. Cette contrainte viole les exigences de simplicité que nous nous sommes fixés dans le chapitre précédent.

La seconde approche proposée dans le cadre de l'utilisation de OSDL comme technique de description formelle pour la spécification du comportement des objets gérés dans GDMO est celle de [Mazaher 93]. Cette approche propose également de traiter séparément la spécification de l'objet géré et celle de son comportement. La spécification de l'objet (ses attributs, ses modules, le profil de ses actions, ...) peut être faite en GDMO. Mais le comportement doit être décrit en OSDL, de manière indépendante. Cela nécessite de passer dans un premier temps de la description GDMO de l'objet vers une description SDL de celui-ci, puis, de spécifier le comportement sur cette dernière spécification. Il est alors impératif de

5. . SDL: Specification and Description Language

6. . Comité Consultatif International Télégraphique et Téléphonique, récemment rebaptisé ITU (International Telecommunications Union)

7. . Object Specification and Description Language

maintenir une consistance entre la spécification GDMO et son correspondant en SDL. De plus, l'approche ne statue pas sur la sémantique de l'extension de comportement par héritage, ce qui est pour le moins surprenant. Cette approche est néanmoins intéressante pour nos travaux car elle prend en compte la nécessité de spécifier l'agent de gestion en même temps que les objets de la base d'informations de gestion qu'il manipule. Ceci est très intéressant car une telle spécification permet éventuellement la simulation ultérieure d'un agent sur une interface CMIP.

L'étude de l'adéquation entre OSDL et GDMO fait apparaître un certain nombre de points de discordance. Les principaux points sont le non-support de l'héritage multiple dans OSDL et l'abscence de concept de module tel qu'il est présent dans GDMO. En effet, dans OSDL le module ne comporte que des types et non des comportements comme cela est le cas dans GDMO.

De plus, les approches proposées pour la spécification du comportement ne satisfont pas l'exigence d'un formalisme unique tel que nous nous l'étions fixé au départ. Cependant, la technique de description formelle SDL n'est pas à écarter comme technique candidate. En effet, elle possède l'avantage indéniable de sa représentation graphique vis-à-vis de l'évolution des outils et méthodes de développement. Ces derniers visent à intégrer de manière plus importante que jamais les aspects graphiques dans les méthodes de développement de systèmes tels qu'ils sont décrit notamment dans [Rumbaugh 92]. La sémantique de l'héritage n'est pas formellement définie dans GDMO. Il est donc concevable de retenir celle définie dans OSDL comme définition de référence.

Les problèmes de manipulation de formalismes multiples peuvent être résolus en intégrant d'une part le langage ASN.1 dans SDL comme cela a déja été proposé à de nombreuses reprises et en définissant une syntaxe de spécification du comportement qui puisse être intégrée dans GDMO. Nous avons dans le cadre de cette thèse retenu une approche de ce type qui est cependant basée sur un autre formalisme que SDL.

2.6 Autres techniques

D'autres approches sont également proposées pour la formalisation du modèle de l'information de l'OSI. Le premier langage proposé dans ce cadre fut le langage MONDEL conçu à l'université de Montréal [Bochmann 91b, Bochmann 91a]. Le caractère confidentiel et l'abscence de documents concernant ce formalisme ne nous permet pas d'en donner l'ensemble des caractéristiques. Il est cependant intéressant d'y faire référence car il est apparemment basé tout comme notre approche sur les automates finis étendus pour la description du comportement des objets gérés.

D'autres types d'approches ont également été retenus pour la description du comportement des objets gérés. Citons à titre d'exemple l'approche basée sur le langage EIFFEL retenue dans le cadre de la spécification d'objets gérés pour les domaines de gestion [Fisher 93]. Cette approche, dénommée DML[8], a l'avantage de permettre une utilisation directe de la spécification du comportement par compilation, mais est limitée vis-à-vis de la répartition de ces descriptions dans les différents formulaires GDMO. En effet dans cette approche, le omportement doit être entièrement spécifié dans les formulaires d'actions et de notification. Ceci implique une limitation de la modularité au niveau de l'approche.

2.7 Le besoin d'une alternative

Dans les sections précédentes, nous avons présenté les différentes techniques de description formelles candidates à la formalisation du comportement des objets gérés dans GDMO.

8. . Domain Management Language

Concernant l'héritage, l'ensemble des techniques formelles présentées ci-dessus (excepté ESTELLE), proposent des mécanismes qui permettent de couvrir cette exigence même si le concept d'héritage prévu dans telle ou telle technique ne correspond pas exactement à celui préconisé dans GDMO (dans OSDL par exemple).

Les limites de ces techniques apparaissent dès que l'on envisage une fusion de leurs mécanismes de description du comportement et de GDMO. Aucun langage proposé à ce jour, ne prend en compte les formalismes existants dans les normes OSI. En effet, aucune approche ne s'est attachée à rester compatible à la fois avec le langage ASN.1 pour les types de données et GDMO pour la spécification des classes d'objets gérés. S'il est techniquement possible d'intégrer ASN.1 dans les formalismes, solution proposée à de nombreuses reprises, cela ne s'est jamais traduit dans la réalité, c'est à dire dans les normes. De plus, pour que les formalismes candidats s'intègrent à GDMO, il convient de fournir un mécanisme de construction du comportement d'un objet à partir des différents formulaires qui le composent. Cela non plus n'a pas été envisagé jusqu'à ce jour.

Il apparaît donc particulièrement intéressant de trouver une alternative à ces langages pour la description du comportement des objets dans le cadre précis de la normalisation des activités de gestion au sein du modèle OSI. C'est à partir des fonctionnalités intéressantes des techniques présentées dans ce chapitre et des exigences énoncées au chapitre précédent que nous allons devoir proposer une alternative.

2.8 Résumé

L'étude des différentes approches proposées nous a montré que malgré leurs nombreux avantages, aucune ne permettait de couvrir l'ensemble des besoins que nous avons identifiés dans le chapitre précédent. Notamment les contraintes d'intégration dans GDMO et de support d'ASN.1 ne sont pas prises en compte dans la plupart de ces techniques candidates. Si l'intégration d'ASN.1 dans ces techniques ne pose en théorie aucun problème majeur, les problèmes de la répartition et de l'intégration dans GDMO sont plus difficiles à traiter. Il apparait néanmoins que trois techniques pourraient les supporter. Elles sont VDM, Object-Z et OSDL.

Dans cette thèse, nous avons retenu une approche différente de celles présentées ci-dessus. Dans le chapitre suivant, nous la présenterons sur la base des besoins identifiés et des enseignements tirés des autres formalismes proposés.

3
Le choix de CRS

Pour la spécification du comportement des objets gérés, nous avons retenu une approche basée sur les concepts issus de la technique de description formelle Communicating Rule Systems CRS [Mackert 87, Neumeier-Mackert 88b]. Initialement conçu pour la spécification des systèmes à base de connaissances répartis, CRS est basé sur des systèmes de règles qui communiquent entre eux au travers de portes de communications [Neumeier-Mackert 88a]. Nous allons dans ce chapitre présenter les caractéristiques de la technique ainsi que les raisons qui nous ont menés à sa sélection.

3.1 Les élements d'une spécification CRS

Objets de base d'une spécification CRS, les systèmes de règles modélisent des systèmes concurrents faiblement couplés. Basé sur un automate fini étendu (EFSM[1]), un système de règles est formé d'un ensemble d'attributs (appelés objets dans la terminologie CRS) et de règles comme nous le montrent les systèmes de règles *RSa*, *RSb* et *RSc* de la figure 3.1. Les objets déterminent l'espace d'état de l'automate associé et les règles en spécifient le comportement (les transitions).

Les systèmes de règles ne peuvent communiquer avec l'extérieur qu'au travers de portes de communication (*GATEab, GATEac, GATEbc* de la figure 3.1). Ces portes sont les seuls objets d'une spécification CRS partageables entre systèmes de règles. Une communication se fait toujours entre deux systèmes: le fournisseur et le consommateur de l'événement.

Une spécification CRS définit l'ensemble des types de systèmes de règles et l'ensemble des types de portes nécessaires à leur interconnexion. Un système réel est obtenu par instanciation des types de systèmes de règles et des types de portes de communication.

3.2 Les systèmes de règles

Les systèmes de règles sont les éléments actifs d'une spécification CRS. La figure 3.2 comporte une spécification simple d'un système de règle. Celle-ci modélise un journal ayant une variable d'état et sur lequel on peut effectuer certaines opérations telles que la lecture de l'état ou le stockage de nouveaux enregistrements.

Un système de règles peut avoir des paramètres utilisables lors de l'instanciation de systèmes avec différents paramètres réels qui se substituent aux paramètres formels. Ils peuvent être des variables ASN.1 (comme `init` à la ligne 1 de l'exemple) ou des portes de communications (portes `inRec (ligne 2), outN (ligne 2), ...`).

1. . Extended Finite State Machine

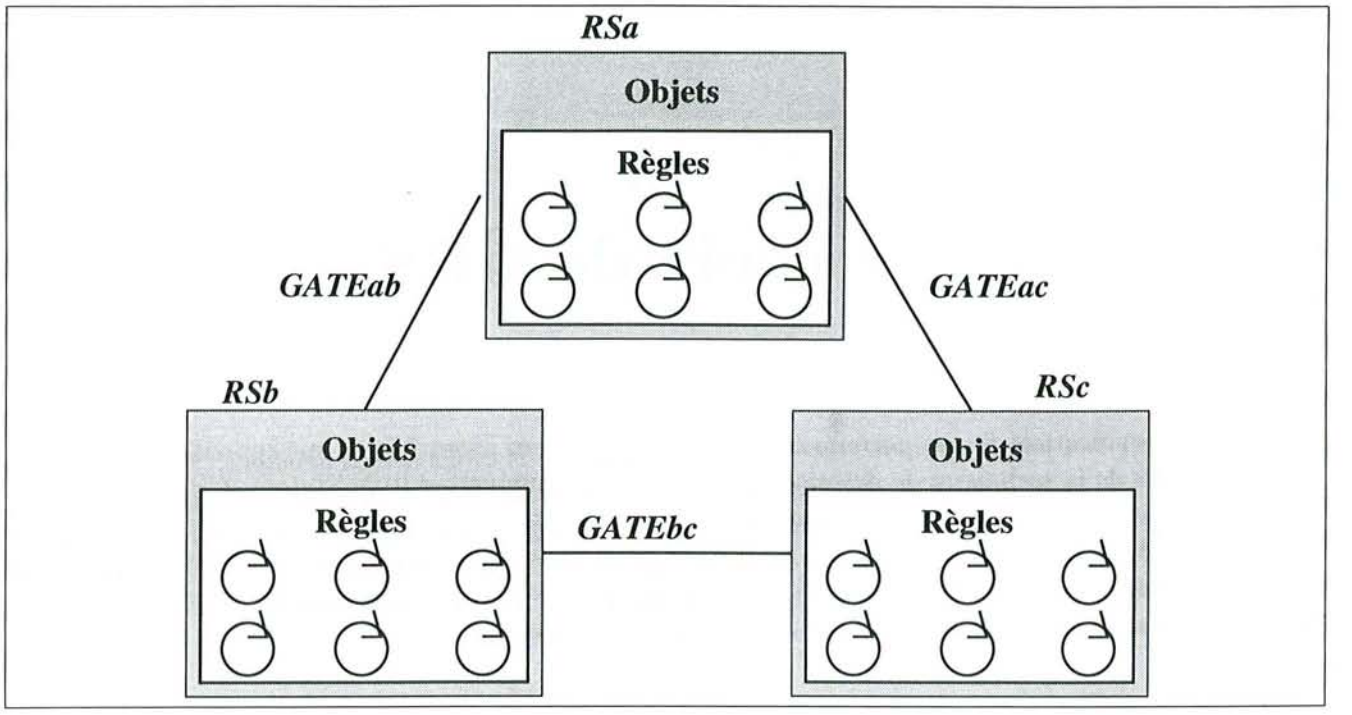

FIG. 3.1 - . *Les éléments de la technique CRS*

Les états d'un système de règles sont définis dans la clause **STATE**. Il est possible de définir des constantes utiles dans la clause **CONSTANTS**. Pour modéliser les conditions de déclenchement et de terminaison de règles, il est également possible de définir des variables intermédiaires dans la clause **DECLARATIONS** (exemple de la variable `getParam` ligne `13` l'exemple). La durée de vie de ces variables locales est celle d'une règle. La clause **DEFINITIONS** permet de spécifier des prédicats utiles employés dans des règles. Enfin, la clause **RULES** comporte la spécification de toutes les règles qui définissent le comportement associé au système de règles. La clause **SYN** contient la description de contraintes sur les règles. Ces contraintes seront présentées dans la section 3.6.

3.3 Les modèles de communication

La syntaxe associée à la spécification d'une porte de communication est celle décrite dans l'exemple de la figure 3.3. Cette spécification décrit une partie d'une interface de système de règles au travers de laquelle des notifications peuvent être émises.

Une porte de communication en CRS est définie par son nom (ici `NotificationGate`), le type de communication qu'elle supporte et l'ensemble des événements qui peuvent s'y produire. Un échange d'événements sur une porte de communication entre deux systèmes de règles se fait, soit de manière synchrone (mot-clé *SYNCHRONOUS*), soit de manière asynchrone (mot-clé *ASYNCHRONOUS*).

Un échange synchrone signifie que deux systèmes de règles peuvent s'échanger un événement par une porte de communication si et seulement si l'offre et la demande de l'événement à la porte se font au même moment. Dans le cas asynchrone, la porte de communication peut accepter une offre d'événement, la stocker dans une file d'attente, libérer le fournisseur et par la suite le sortir de la file lorsqu'un autre système désire consommer l'événement.

```
RULE_SYSTEM LogType(init: LogInitalState;
                    inRec: PotentialRecordGate;
                    outN : NotificationGate
                    manG : ManagementGate
                    RecG : RecordAccessGate);
CONSTANTS
     TR : BOOLEAN TRUE;
STATE
    administrativeState: ENUMERATED (unlocked(0),locked(1))
                    INITIALLY init.administrativeState;
     ...
DECLARATIONS
      getParam: GetParameter;
     ...
DEFINITIONS
     ...
RULES
  get;
    COND: manG.getReq(getParam)
      AND {"smi2AttributeId 31"} ∈ getParam.attributeIdList
      ...
    EFFECTS:
    IF ("smi2AttributeId 31" ∈ Get.attributeIdList)
    THEN {"smi2AttributeId 31", 'administrativeState} ∈ GetRspParam.attributeList
      ...
SYN
     ...
END_RULE_SYSTEM LogType;
```

FIG. 3.2 - . *Le schéma de spécification de système de règles*

La spécification des événements se fait par énumération de l'ensemble des paramètres véhiculés par ces événements (section **DECLARATIONS**) et par description des événements et des paramètres associés (section **EVENTS**). Par exemple, l'événement `stateChangeNotification` véhicule le paramètre `stateChangeParameter` de type `StateChangeInformation`. Naturellement, un événement peut véhiculer plusieurs paramètres, syntaxiquement séparés par une virgule, dans la liste des paramètres associés à celui-ci.

3.4 Les données

La FDT CRS comporte deux mécanismes pour la description des données. Le premier est basé sur une version étendue du langage ASN.1 et le second sur les types abstraits de données pour la description d'objets complexes.

La FDT CRS supporte l'ensemble de la syntaxe ASN.1 pour la définition des types de ses variables. Comme ASN.1 est une notation de types et que les seules opérations définies sur ces types dans les normes sont l'encodage et le décodage, il a fallu doter CRS de nouveaux opérateurs sur ASN.1 permettant de

```
GATE ASYNCHRONOUS NotificationGate;
  DECLARATIONS
     stateChangeParameter    : StateChangeInformation;
     ...
  EVENTS
     stateChangeNotification (stateChangeParameter);
     ...
END GATE incomming_notifications;
```

FIG. 3.3 - . *Le schéma de spécification d'une porte de communication*

manipuler des variables de type ASN.1 [Schneider 92]. La notation ASN.1 ainsi pourvue d'opérateurs de manipulation d'instances de variables a été dénommée eXtended ASN1 (X-ASN.1).

Les opérations définies sur les types ASN.1 sont principalement ceux d'accès aux champs d'une séquence (.), de tests de présence de champs optionnels **ISPRESENT(variable.champ)**, de manipulation d'ensembles (union, intersection, sous-ensemble, différence, cardinalité...) et de manipulation de valeurs (relations sur les entiers, opérateurs logiques sur les booléens, etc..). Des opérateurs ont été définis pour chaque type simple ASN.1 et pour chaque type structuré (ensemble, liste, choix, structure). Ces opérateurs permettent donc de manipuler toute instance de type ASN.1.

Les types de données abstraits (ADT[2]) peuvent également être spécifiés en CRS. Ces objets passifs sont basés, tout comme les systèmes de règles, sur un modèle d'automate et comportent deux parties. La première définit les états du type et la seconde les opérations sur ces états. La spécification d'une opération comporte son nom, ses paramètres, la pré-condition d'invocation et la post-condition associée. L'utilisation des ADT permet de mieux supporter l'abstraction sur les données.

3.5 Les règles

Une règle CRS est définie par son nom et par deux prédicats basés sur la logique du premier ordre étendue avec le concept d'occurence d'événements (voir figure 3.4).

```
<nom-de-regle>;
[ COND:    <condition de déclenchement>;]
[ EFFECTS: <effets du tirage de la règle sur l'état du système
                et événements offerts>;]
```

FIG. 3.4 - . *La syntaxe d'une règle CRS*

Le premier prédicat (**COND**) définit la condition de déclenchement de la règle. Il représente, d'une part l'état dans lequel doit se trouver le système, et d'autre part les événements qui doivent être offerts aux interfaces (portes) du système afin que la règle puisse être tirée.

2. . Abstract Data Type

Le second prédicat (**EFFECTS**) décrit l'état dans lequel doit se trouver le système et les événements qui doivent être offerts aux portes de communications après la terminaison de la règle.

Vis-à-vis de la communication, on ne peut trouver dans un prédicat de déclenchement de règles que des demandes d'événements alors qu'un prédicat de terminaison ne comporte que des offres. Les deux prédicats d'une règle sont considérés comme deux actions atomiques séparées. Ils ne s'exécutent pas au même moment. Pour réguler l'exécution parallèle de règles, la technique de description formelle met à disposition un formalisme de spécification de synchronisation entre règles (voir 3.6).

3.6 Autres mécanismes du langage

Deux mécanismes additionnels sont mis à disposition des concepteurs de spécifications dans CRS. Le premier permet de spécifier la synchronisation entre règles et le second de structurer les règles au sein d'une spécification.

Le mécanisme de synchronisation est nécessaire en raison de l'exécution non atomique d'une règle qui permet l'entrelacement des exécutions. Pour spécifier ces contraintes d'entrelacement, CRS fournit trois opérateurs de synchronisation:

- l'opérateur de parallélisme (`règle1 || règle2`) qui autorise l'exécution parallèle des règles `règle1` et `règle2`.
- l'opérateur de semi parallélisme (`règle1` $\vdash$ `règle2`) indique que la règle `règle1` peut être tirée durant l'exécution de la règle `règle2`. L'inverse n'est pas permis.
- l'opérateur de priorité (`règle1` $\prec$ `règle2`) indique que si les deux règles `règle1` et `règle2` sont tirables alors la `règle1` est tirée avec une certaine priorité. Cette priorité peut se définir par l'association de probabilités à chaque règle.

La spécification de synchronisations permet d'introduire le parallélisme entre les règles. En effet, si aucune synchronisation n'est spécifiée, alors toutes les règles s'exécutent de façon atomique et ne peuvent donc pas s'entrelacer.

Le second mécanisme fournit par CRS est celui de structuration de règles au sein d'un système de règles. Les deux types de structurations sont les contextes **CONTEXT** et les vues **VIEW** [Velthuys 92b].

Le mécanisme de structuration **CONTEXT** permet de regrouper des règles au sein d'une structure de blocs. Cela concerne les règles reliées par des concepts logiques comme par exemple celles qui ont des conditions et/ou effets identiques. Ce type de structuration permet d'extraire les points communs des règles reliées et de définir des variables locales au contexte.

Le mécanisme de structuration **VIEW** permet au sein d'une spécification, de séparer les différents aspects d'une règle complexe en sous ensembles compréhensifs relatifs à un aspect particulier. Par exemple, on peut séparer dans une règle la partie décrivant les influences sur les états du système de celle spécifiant les affectations des champs de PDU.

3.7 La sémantique opérationelle de CRS

La technique de description formelle CRS a une sémantique formelle qui est basée sur les systèmes de transitions étiquetés. La sémantique, élaborée dans [Schneider 90b] et présentée dans [Schneider 92], décrit la manière dont le système de transitions étiquetées associé à une spécification de systèmes de

règles communicants est construit. Le but de cette section n'est pas de présenter en détails la sémantique associée à CRS. Nous allons pour nos besoins ultérieurs, en présenter simplement quelques caractéristiques.

Un système de transitions étiquetées (LTS[3]) est constitué d'un ensemble d'états et d'une relation de transition qui, à toute étiquette, associe un ensemble d'états de départ ainsi qu'un ensemble d'états d'arrivée. Le système de transitions étiquetées associé à un système de règles est constitué de la manière suivante:

- les états du système de transitions sont obtenus par le produit cartésien des domaines des variables d'état et de l'ensemble des états d'exécution du système. Cet ensemble des états d'exécution est défini comme le produit cartésien de l'état des règles du système et des domaines des variables locales à chacune des règles.

- les étiquettes du système sont définies comme l'ensemble des interactions possibles aux interfaces du système. Une telle interaction est définie par la porte de communication à laquelle elle se produit, son nom (nom de l'interaction), ses paramètres formels, ainsi que son type. Le type d'une interaction définit l'orientation de l'interaction. Celui-ci peut être une requête (demande d'occurence de la part du système de règles), une offre (émission d'une interaction par le système de règles) ou un événement survenu à une porte externe du système. Ces étiquettes sont étendues avec l'étiquette i qui représente un type d'événement anonyme sans paramètres utilisé comme étiquette pour les transitions des règles non liées à l'occurence d'événements aux portes de communication.

- la relation de transition est définie comme suit: une transition du système est tirable s'il existe une règle du système associé qui est soit tirable (toutes les conditions sont vérifiées pour déclencher la règle) soit terminable (la post-condition d'une règle en cours d'exécution est vérifiée et on peut donc achever son exécution).

Le système de transitions étiquetées, associé à une spécification CRS contenant plusieurs systèmes de règles reliés entre eux à travers des portes de communications, est construit à partir des systèmes de transitions étiquetées des systèmes de règles qui composent cette spécification. L'état du système de transitions global est équivalent au produit cartésien de ses composants. Les étiquettes sont toutes les étiquettes des différents systèmes de règles. Les transitions sont définies comme suit: une transition peut être tirée s'il existe un système de règles qui offre un événement à une porte de communication sur laquelle un autre système a émis une requête pour cet événement et ces paramètres.

Une deuxième sémantique, basée sur les réseaux de Pétri, a également été définie pour CRS [Schneider 90b]. Dans le cadre des extensions que nous allons y apporter, seule la première, basée sur les systèmes de transitions étiquetées, sera étendue.

3.8 Les outils autour de CRS

Afin de motiver notre décision d'étudier CRS comme technique de description formelle pour la spécification du comportement des objets gérés, il nous semble intéressant de présenter brièvement l'environnement de développement associé à la technique. Cet environnement est aujourd'hui relativement complet et opérationnel. Le concept d'environnement intégré pour le développement de protocoles de communication a été proposé pour CRS dans [Schneider 90a]. Cet environnement est basé sur un noyau

3. . Labelled Transition System

composé, d'une part des spécifications CRS représentant la connaissance de l'environnement, et d'autre part d'un ensemble d'outils permettant l'élaboration et la manipulation de ces spécifications.

Les outils du noyau, présents dans l'environnement actuel, sont une machine d'inférence générique, des machines de simulation de portes de communication et un environnement d'exécution de spécification standard. Le noyau comporte également un compilateur CRS, un éditeur syntaxique, un préprocesseur et un compilateur X-ASN.1.

L'information modélisée dans le noyau de l'environnement est accessible via trois interfaces différentes. La première focalise sur les règles d'une spécification CRS. Elle permet d'accéder et de modifier toute information relative à ces règles (activation, terminaison, ...). La seconde interface est consacrée aux états des systèmes de règles (lecture d'un état, affectation, ...). La troisième permet d'accéder aux événements visibles aux portes de communications entre systèmes de règles (occurences d'événements, paramètres, ...). Ces interfaces sont utilisées par les différents outils dits applicatifs qui ont pour but de permettre une exploitation de la connaissance fournie dans une spécification concentrée dans le noyau.

Différents outils applicatifs ont été conçus et réalisés dans l'environnement. Le premier est un simulateur interactif de spécifications dans un environnement graphique multi-fenétré. Cet outil permet à un spécifieur de suivre l'exécution d'une spécification en mode pas-à-pas ou automatique. Cet outil très intéressant pour les travaux présentés dans cette thèse a été repris, porté sur UNIX et étendu dans nos activités. Nous y reviendrons dans la troisième partie de la thèse. Les autres outils applicatifs existants à ce jour sont, d'une part un outil de développement de tests TTCN[4] ([ISO-9646.3 91]) bâti sur l'interface événementielle du noyau et, d'autre part un outil de validation de tests. Ce dernier permet après une analyse syntaxique des séquences, de les confronter au comportement de la spécification formelle afin de les valider et d'obtenir ensuite des informations sur leur couverture. Ces outils sont les principaux éléments de l'environnement existant.

3.9 Les acquis de CRS

Le langage CRS a été utilisé avec succès pour la spécification de nombreux services et protocoles de différents degrés de complexité. Les principales spécifications disponibles à ce jour sont celles de X.25 [Mangold 90, Monnery 89], de la couche session [Velthuys 92a] et du traitement transactionnel normalisé [Festor 92]. Ces spécifications ont permis, d'une part de montrer l'adéquation entre le formalisme CRS et le besoin de spécification formelle des services et protocoles de communication, et d'autre part de spécifier de nouveaux mécanismes dans le langage pour faciliter la conception de spécifications complexes.

De ce point de vue, CRS est une technique de description formelle directement issue des besoins des protocoles et services de l'OSI. Ceci est particulièrement important pour notre choix. En effet, dans sa version actuelle, CRS supporte entièrement le langage ASN.1 pour la description des types de données et pour la manipulation de variables d'état de systèmes de règles grâce au formalisme X-ASN.1 mentionné précédemment. Vu le souhait que nous avons exprimé pour un support du formalisme ASN.1 dans la FDT retenue pour le comportement dans GDMO, il apparait que CRS est bien adapté pour satisfaire ces exigences.

De plus, le mécanisme de règles utilisé pour la description du comportement des systèmes de règles semble proche du raisonnement humain et assez souple pour pouvoir s'adapter à une intégration dans GDMO. C'est l'une de nos principales préoccupations.

Bien entendu, l'expérience personnelle acquise avec ce formalisme et l'ensemble des outils disponibles, notamment en matière de simulation, ont influencé le choix de CRS comme technique de description

4. . Tree and Tabular Combined Notation

formelle à étudier pour la spécification du comportement des objets gérés. Ce choix a impliqué un certain nombre de travaux sur la technique elle même, comme nous le verrons dans les sections suivantes.

3.10 CRS et la spécification objet

La technique de description formelle CRS n'est pas à proprement parler orientée-objets, du moins elle n'a pas été conçue dans ce but. Cependant, elle comporte un certain nombre de ses caractéristiques qui sont:

- **type/instance:** une spécification d'un système de règles tout comme la spécification d'une classe d'objet définit un type de système instanciable plusieurs fois.
- **activité:** une instance de système de règles est considérée au même titre qu'une instance d'objet géré dans l'approche OSI, comme un élément actif du système pouvant consommer et émettre des événements.
- **encapsulation:** un système de règles, tout comme une classe d'objet géré, encapsule un certain nombre de données (variables d'état d'un système de règles / attributs d'une classe d'objet géré) et d'opérations qui ne sont accessibles que par échange de messages à travers une interface clairement définie.

En faisant le lien entre un système de règles et un objet, on se rend compte que CRS comporte certaines caractéristiques d'une approche objet mais n'en supporte hélas pas la principale: l'héritage. En effet, CRS n'intègre pas ce concept, ni pour les systèmes de règles, ni pour la spécification des interfaces, i.e. les portes de communication.

De plus, le formalisme CRS ne permet pas de créer et de détruire des instances de systèmes de règles de manière dynamique. Ce problème a déjà été identifié et résolu dans le cadre des travaux sur la formalisation du protocole de traitement transactionnel (OSI TP) en CRS [Scheere 91]. Ces travaux ont mené à la proposition de nouvelles fonctions permettant, au sein d'une règle, de créer et de détruire des systèmes de règles, mais aussi des portes de communication.

Afin de faire le lien entre CRS et le langage de description des objets gérés GDMO, il nous faut examiner la manière dont un système de règles pourrait être construit par héritage. Par la suite, nous étudierons comment mettre en place un lien entre une spécification d'objets en GDMO et un tel système de règles.

3.11 L'idée de formalisation dans GDMO

Le langage CRS popose un certain nombre de fonctionnalités très proches des exigences retenues dès le début de cette thèse. Cependant, le but premier que nous nous sommes fixés est de doter le langage GDMO de mécanismes pour la description formelle du comportement des objets gérés et non d'étendre le formalisme CRS. Pour cela, nous nous proposons de faire le lien entre la syntaxe GDMO et celle de CRS, en établissant une correspondance entre les éléments d'une spécification d'objet géré et une description formelle d'un système de règles en CRS. Ceci nous permettra par la suite d'étendre la partie comportement de GDMO avec le mécanisme de règles proposé dans CRS.

L'idée est décrite dans la figure 3.5. Il s'agit d'introduire dans GDMO le mécanisme de règles issu des concepts de CRS pour la description du comportement, et de permettre la spécification dans GDMO

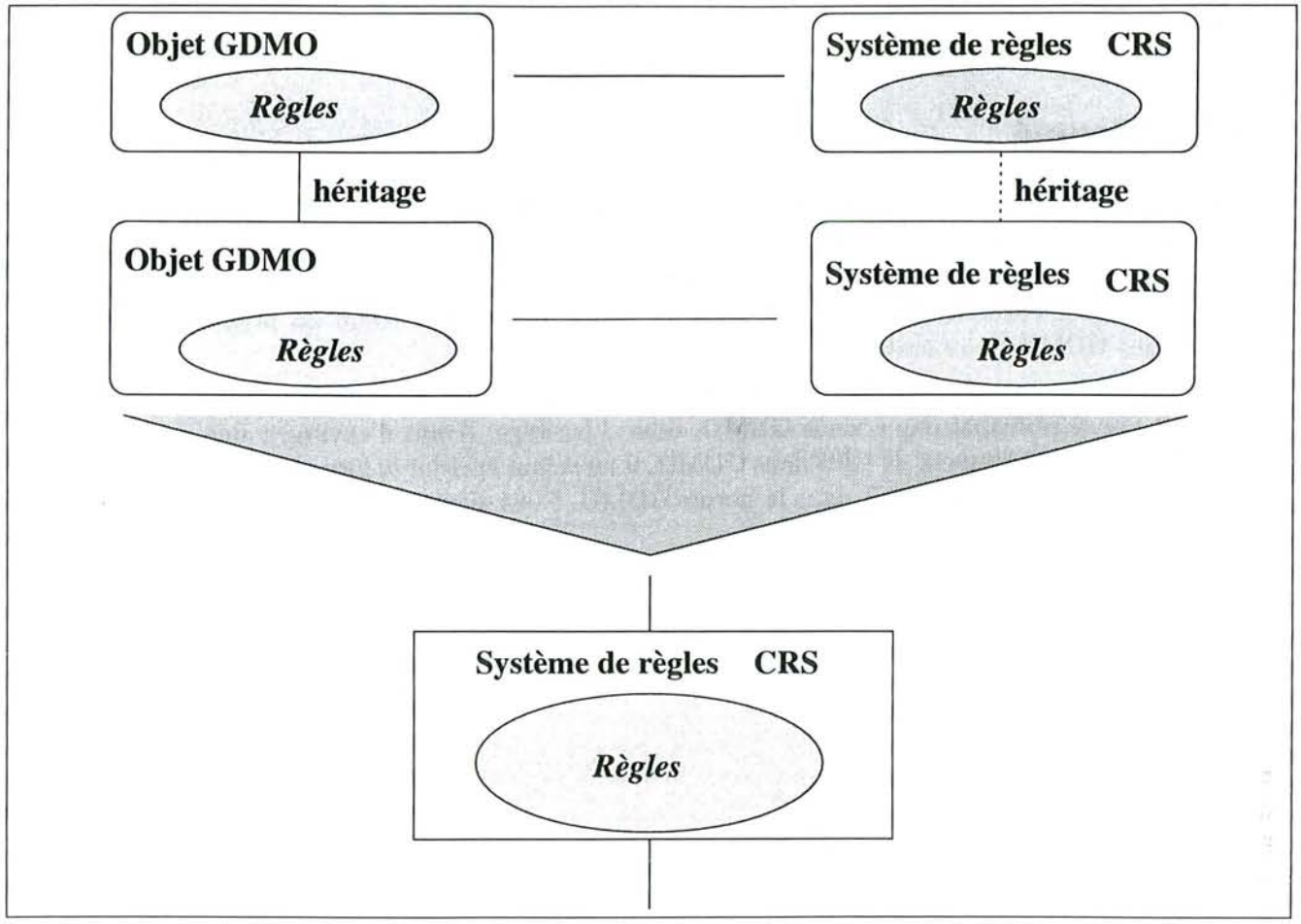

FIG. 3.5 - . *Schéma de l'intégration*

de plusieurs interfaces pour les objets (Objet GDMO étendu avec un mécanisme de règles) [Festor 93]. A partir de telles spécifications GDMO, le but est de construire un système de règles CRS équivalent, en définissant une méthode de passage d'un ensemble de spécifications GDMO à un système de règles CRS.

Pour cela, il faut dans un premier temps, faire le lien entre un objet GDMO et un système de règles CRS. A partir de ce lien, il nous faut définir de manière formelle sur la sémantique associée aux systèmes de règles et aux interfaces, le mécanisme d'héritage tel qu'il apparaît dans GDMO. Une fois l'héritage défini il nous sera possible de construire à partir de spécifications GDMO étendues, un seul système de règles contenant la description complète du comportement associé à l'objet géré considéré.

La première tâche concernant le lien entre un objet GDMO et un système de règle CRS est triviale. En effet, un objet GDMO tel qu'il est considéré dans l'approche OSI, est défini comme un objet actif accessible à travers l'invocation d'opérations à ses interfaces. C'est équivalent au concept des systèmes de règles CRS. Aussi, pouvons nous définir un objet géré comme un système de règles. Les attributs du premier forment les variables d'état du second, et le comportement de l'objet géré, les règles du système de règles associé.

Ce lien établi, il faut étendre les mécanismes de CRS avec le concept d'héritage strict. Cette extension

fait l'objet du chapitre suivant.

3.12 Résumé

Dans ce chapitre nous avons présenté le formalisme CRS, l'environnement existant ainsi que les raisons qui ont déterminé le choix de ce formalisme pour la spécification du comportement des objets gérés. Ces raisons sont principalement les acquis du langage et la souplesse de ses concepts qui permettent d'envisager une intégration dans GDMO. Dans ce cadre, nous avons établi un premier lien entre les spécifications GDMO et les systèmes de règles.

Nous avons également vu que CRS disposait d'un certain nombre de concepts orientés-objets mais ne fournissait pas le principal requis dans GDMO: celui d'héritage. Avant d'envisager une intégration des mécanismes de comportement de CRS dans GDMO, il nous faut enrichir le formalisme avec le mécanisme d'héritage strict, tel qu'il apparait dans la norme GDMO. Nous allons, dans le chapitre suivant aborder cet aspect.

4
L'extension de CRS par les mécanismes d'héritage

Pour permettre une intégration des mécanismes de CRS dans le langage GDMO et établir un lien entre les langages, il faut intégrer le mécanisme d'héritage de GDMO dans CRS. Cette intégration est présentée dans ce chapitre. Après une étude sur un exemple de système de transitions étiquetées simple, nous allons présenter les extensions apportées à CRS pour le support des concepts orientés-objets nécessaires à son intégration dans GDMO.

4.1 Contexte

La sémantique de CRS est basée sur les systèmes de transitions étiquetées (LTS[1]). Aussi, pour intégrer les mécanismes de description du comportement de la technique dans le langage GDMO, il nous faut définir de manière formelle, le concept d'héritage dans CRS.

Pour cela, nous allons procéder en deux étapes. Dans un premier temps, nous allons nous intéresser aux LTS en général, et par la suite, appliquer les résultats obtenus à la sémantique de CRS.

Définir l'héritage sur les LTS revient à établir des règles de construction qui permettent de construire entièrement un LTS à partir de différentes spécifications (sous-classe et super-classe).

Un système de transitions étiquetées est composé d'un ensemble d'états, de transitions, d'étiquettes et d'un état initial. Les règles de construction doivent donc définir:

1. comment les états d'un système peuvent être construits par héritage,
2. comment les transitions sont construites et quelles sont les contraintes à vérifier pour le respect de l'héritage strict, tel qu'il est défini dans [Meyer 88] et retenu dans l'approche OSI,
3. comment les étiquettes sont obtenues par héritage,
4. quel est le nouvel état initial du système

Pour répondre à ces quatre questions et appliquer les réponses au formalisme CRS, l'étude d'un exemple simple nous permettra de définir de manière empirique ces règles. Par la suite, nous les généraliserons aux systèmes de transitions étiquetées pour finalement les intégrer à la sémantique de CRS.

1. . Labelled Transition System

4.2 L'héritage et les systèmes de transitions étiquetées

Pour définir des règles de construction d'un LTS par héritage, nous allons procéder de la manière suivante: après une liste de définitions utiles tout au long du chapitre, nous regarderons sur un exemple simple, comment l'héritage peut être appliqué à un LTS. Cet exemple nous permettra de définir des règles de constructions réutilisées par la suite.

4.2.1 Définitions

Pour permettre une meilleure compréhension de la suite du chapitre, nous donnons quelques définitions qui clarifient la terminologie utilisée.

- **Système de transitions étiquetées:** un système de transitions étiquetées LTS est caratérisé par le modèle suivant:

$$LTS =< S, L, T, s_0 > \tag{4.1}$$

où S représente un ensemble fini non nul d'états, L un ensemble fini non-nul d'étiquettes, T un ensemble de transistions de la forme $< X, L >\rightarrow Y$ ou $X, Y \in S$ et s_0 l'état initial du système.

- **Variable d'état:** les états d'un système de transitions étiquetées sont construits à partir de l'ensemble des variables qui le composent. Une variable d'état est définie par un nom et un type. La notation $SV(A)$, où A est un système de transitions étiquetées, désigne l'ensemble des variables d'états d'un système de transitions étiquetées A.

- **Domaine d'une variable:** comme les états d'un système de transitions étiquetées sont dans notre cas construits à partir des variables qui le composent, nous appellerons domaine d'une variable, l'ensemble des valeurs qu'elle peut prendre. Pour une variable x de type T, le domaine s'écrira $DOM(x)$. Exemple:

$$v : BOOLEAN;$$
$$DOM(v) = \{TRUE, FALSE\}$$

- **Espace d'états:** l'espace d'états d'un système de transitions étiquetées est défini comme le produit cartésien de l'ensemble des domaines des variables d'état qui le composent. Cet espace est en fait l'ensemble S d'un LTS. Nous assimilerons par la suite l'espace d'états aux états eux-mêmes. Exemple:

```
variables d'état: v: BOOLEAN; w: ENUMERATED {un(1), deux(2)}
```

Espace d'états:

$$S = DOM(v) \times DOM(w) = \{(TRUE, un), (TRUE, deux), (FALSE, un), (FALSE, deux)\}$$

- **Label:** le label d'une transition t représente son étiquette. Celui-ci se notera par la suite $LABEL(t)$. Dans le cadre de notre étude, nous supposons que chaque transition a un label unique qui représente son nom et l'occurence de l'événement attendu.

- **pré- et post-condition:** la pré-condition d'une transition $< X, L >\rightarrow Y$ représente l'ensemble des états de départ de celle-ci (ici X). La pré-condition d'une transition sera dans la suite du chapitre notée $PRE(t)$ où t représente la transition concernée. La post-condition représente l'ensemble des états d'arrivée de celle-ci (ici Y). La post-condition d'une transition sera dans la suite du chapitre notée $POST(t)$.
- **Projection d'une transition d'un LTS sur un LTS étendu:** la projection d'une transition t d'un LTS A sur un LTS étendu A' est définie comme la construction de t dans A' à partir de sa spécification donnée dans A:

$$
\begin{array}{l}
SV(A) = a_{1_1}, \ \ldots \ a_{1_n} \\
SV(A') = a_{1_1}, \ \ldots \ a_{1_n}, \ a_{2_1}, \ \ldots \ a_{2_r} \\
PROJ(t, A, A') \Leftrightarrow \\
\{t' : \ LABEL(t') = LABEL(t) \\
\quad \wedge \ PRE(t') = \{< v_{1_1}, \ldots, \ v_{1_n}, w_{2_1}, \ldots \ w_{2_r} >: \\
\qquad\qquad < v_{1_1}, \ldots \ v_{1_n} > \in \ PRE(t) \\
\qquad\qquad \wedge \ w_{2_1} \in \ DOM(a_{2_1}) \ \ldots \ \wedge \ w_{2_r} \in \ DOM(a_{2_r})\} \\
\quad \wedge \ POST(t') = \{< x_{1_1}, \ldots, \ x_{1_n}, y_{2_1}, \ldots \ y_{2_r} >: \\
\qquad\qquad < x_{1_1}, \ldots \ x_{1_n} > \in \ POST(t) \\
\qquad\qquad \wedge \ y_{2_1} \in \ DOM(a_{2_1}) \ \ldots \ \wedge \ y_{2_r} \in \ DOM(a_{2_r})\} \\
\}
\end{array}
\tag{4.2}
$$

  La projection d'une transition t d'un LTS A sur un LTS A' revient donc à construire dans A' une transition de label identique à celui de t, de pré-condition égale à la projection de celle définie dans A sur A' et de post-condition égale à la projection de celle définie dans A projetée sur A'. Une illustration de cette projection est donné dans la figure 4.4.

Dans la suite de ce chapitre, nous utiliserons ces définitions pour formaliser le mécanisme d'héritage. Afin de simplifier la notation, nous utiliserons également le terme LTS pour référer un système de transitions étiquetées.

4.2.2 Un exemple de système de transitions étiquetées

Pour illustrer notre propos, considérons l'exemple très simple de la figure 4.1 ci-après. Il décrit un LTS A basé sur deux variables d'état `a` et `b`. La première étant une variable booléenne (`T`,`F`) et la seconde une restriction des entiers aux nombres `1` et `2`. L'ensemble des états du LTS est obtenu par le produit cartésien des domaines de ses deux variables d'états. Le LTS comporte une transition `tr` dont la pré-condition est l'ensemble (`<T,1>`,`<T,2>`) et la post-condition l'état `<F,2>`. L'ensemble des étiquettes du système est égal à l'ensemble des noms des transitions. L'état initial s_0 est l'état $< T, 1 >$.

Soit A' le LTS de la figure 4.2. On remarque qu'il est relativement proche de A, comme présenté dans la figure précédente. En effet, on peut le considérer comme une extension du LTS A par adjonction de la variable d'état `c` pouvant prendre les valeurs `v` et `w`. Il est également étendu, d'une part par adjonction de la transition `tr1`, et d'autre part par extension de la transition `tr`: affaiblissement des conditions sur l'ensemble des états de départ et restriction sur l'ensemble des états d'arrivée de `<F,2>` sur `<F,2,v>`.

Cette extension peut être assimilée à la définition de l'héritage strict défini dans [Meyer 88]. De ce fait, on peut définir A' comme un LTS héritant les fonctionnalités de A et étendant ses caractéristiques. Une telle définition est donnée dans la figure 4.3.

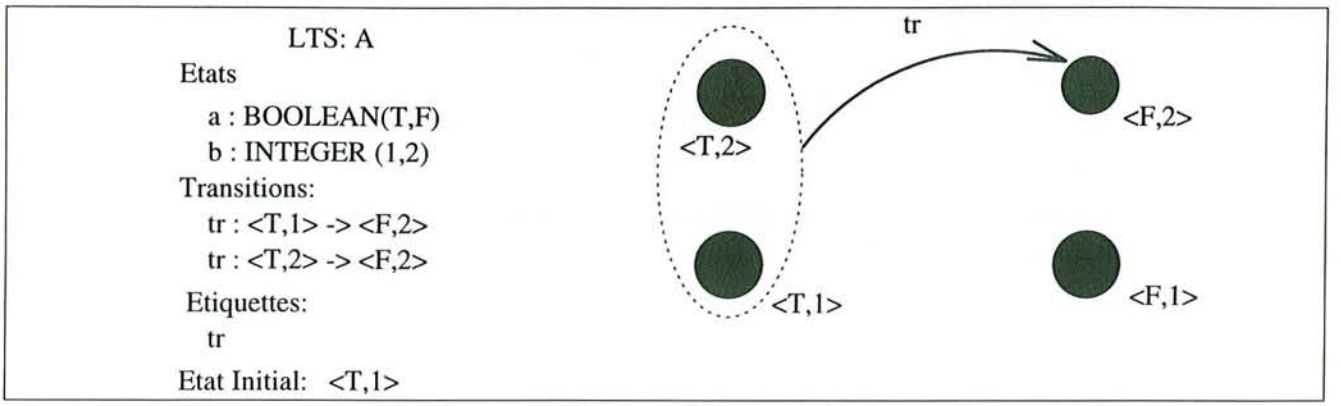

FIG. 4.1 - . *Un système de transitions étiquetées simple*

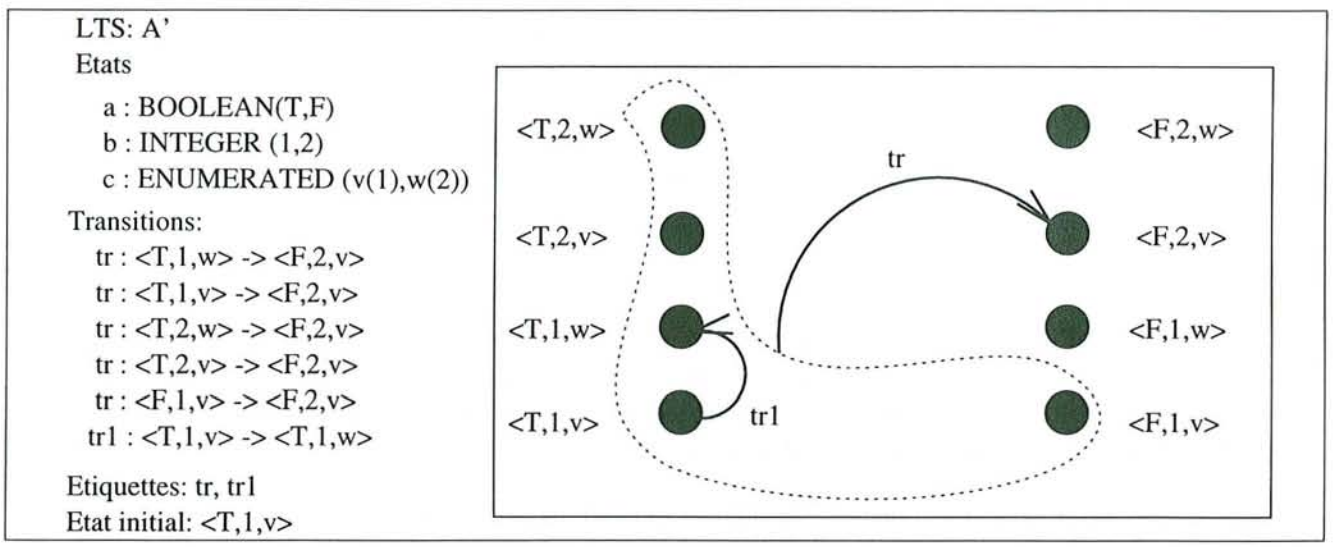

FIG. 4.2 - . *Un LTS étendu*

Cette définition est intuitive et ne spécifie pas les règles qui, à partir de cette description, permettent de reconstruire le LTS A' tel qu'il est défini dans la figure 4.2. Sur la base de cet exemple, nous définissons les règles de construction de systèmes de transitions étiquetées par héritage. Dans un premier temps, nous considérerons l'héritage simple sans contrainte puis nous regarderons la contrainte d'héritage strict.

4.2.3 Règles de construction

Cet exemple très simple nous engage à définir le concept d'héritage comme un ensemble de règles de construction entre deux LTS.

L'héritage y implique deux systèmes de transitions étiquetées A et A'. A' est construit à partir de sa définition propre et des fonctionnalités qu'il hérite de A. Pour simplifier les explications et la compréhension des règles, nous introduisons un LTS A_{res}, résultat de la construction. Ce système de transitions étiquetées représente A' après application des règles de construction.

LTS: A' herite de A
Extensions
Etats
c : ENUMERATED (v(1),w(2))
Transitions:
trl : <T,1,v> -> <T,1,w>
tr : <F,1,v> -> <F,2,v>
Etat initial:
<T,1,v>

FIG. 4.3 - . *Un LTS défini par héritage*

Héritage simple sans contraintes

Définissons les règles de construction qui permettent de construire A_{res} à partir des définitions de A (figure 4.1) et A' (figure 4.3) pour les différents éléments du LTS:

- la liste des variables d'état du LTS A_{res} est égale à l'union de celle de A et de celle de A':

$$SV(A_{res}) = SV(A) \cup SV(A') \tag{4.3}$$

En appliquant cette formule sur l'exemple de la section précédente, on obtient:

$$SV(A_{res}) = \{a, b, c\}$$

ce qui correspond bien à l'ensemble souhaité.

- l'ensemble des états $S_{A_{res}}$ de A_{res} est égal au produit cartésien du domaine de l'ensemble des variables $SV(A_{res})$:

Si on a:

$$SV(A_{res}) = a_1, a_2, \ldots a_n \tag{4.4}$$

on aura:

$$S_{A_{res}} = DOM(a_1) \times DOM(a_2) \ldots \times DOM(a_n) \tag{4.5}$$

Sur l'exemple, on obtient:

$$S_{A_{res}} = DOM(a) \times DOM(b) \times DOM(c)$$
$$=$$
$$\{< T, 1, v >; < T, 1, w >; < T, 2, v >; < T, 2, w >;$$
$$< F, 1, v >; < F, 1, w >; < F, 2, v >; < F, 2, w >\}$$

qui est bien l'ensemble des états du LTS A' entièrement défini dans la figure 4.2.

- l'ensemble des transitions $T_{A_{res}}$ du LTS résultat A_{res} est construit en trois étapes. Le résultat de chaque étape donne un sous-ensemble de transitions T_1, T_2, T_3. L'ensemble final est égal à l'union des sous-ensembles résultant de chacune de ces étapes:

$$T_{A_{res}} = T_1 \cup T_2 \cup T_3 \tag{4.6}$$

  - T_1 est l'ensemble des transitions de A' non spécifiées dans A, c'est à dire les transitions dont le label est défini dans A':

$$T_1 = \{t \mid t \in T_{A'} \wedge \forall t' \in T_A : LABEL(t) \neq LABEL(t')\} \tag{4.7}$$

    Sur l'exemple, seule la transition $tr1$ respecte cette contrainte, donc $T_1 = \{tr1\}$.
  - la seconde étape va nous permettre de construire T_2 à partir de l'ensemble des transitions définies dans A et non spécifiées dans A'. Ces transitions sont projetées sur l'espace d'états de A_{res} et ajoutées à l'ensemble T_2:

$$T_2 = \{t_{proj} \mid \exists t \in T_A, \forall t' \in T_{A'} : LABEL(t) \neq LABEL(t') \Rightarrow t_{proj} = PROJ(t, A, A')\} \tag{4.8}$$

    La transition tr est spécifiée à la fois dans A et A'. Il n'existe aucune autre transition dans A, donc $T_2 = \oslash$ dans l'exemple.
  - la dernière étape concerne le traitement des transitions définies dans A et dans A'. Pour ce type de transitions, nous définissons la règle de construction des transistions résultantes comme suit:

$$T_3 = \{t \mid t \in T_{A'} \wedge \exists t' \in T_A : LABEL(t) = LABEL(t')\} \tag{4.9}$$

    Ceci signifie que seule la partie des transitions redéfinies dans A' est conservée dans le LTS résultat A_{res}.
    Appliquée à l'exemple cette règle nous donne:

$$T3 = \{tr\}$$
$$\text{avec } tr :< F, 1, v > \rightarrow < F, 2, v >$$

    Comme on peut le voir, cette règle ne considère en fait que les redéfinitions de transitions et ne permet pas d'obtenir l'ensemble des transitions telles qu'elles sont définies dans A'. Cette règle est cependant parfaitement valable pour une définition d'héritage. Nous verrons dans la suite comment celle-ci peut être modifiée pour prendre en compte l'héritage strict qui va permettre de reconstruire A' de manière conforme à nos attentes.

On considère ici qu'une transition a un seul nom, que toutes les transitions ont un nom différent et que le calcul décrit ci-dessus ne se fait que sur la base des noms des transitions.

- l'ensemble des étiquettes $L_{A_{res}}$ du système A_{res} est égal à l'union des étiquettes du système A et de celles du système A':

$$L_{A_{res}} = L_A \cup L_{A'} \tag{4.10}$$

Les étiquettes reflètent ici les noms des transitions et aucune transition n'est supprimée par le calcul.

Sur l'exemple on obtient:

$$L_{A_{res}} = \{tr, tr1\}$$

- L'état initial du LTS résultat de la construction $s_{0_{A_{res}}}$, est celui défini dans le LTS A' ($< T, 1, v >$ dans l'exemple).

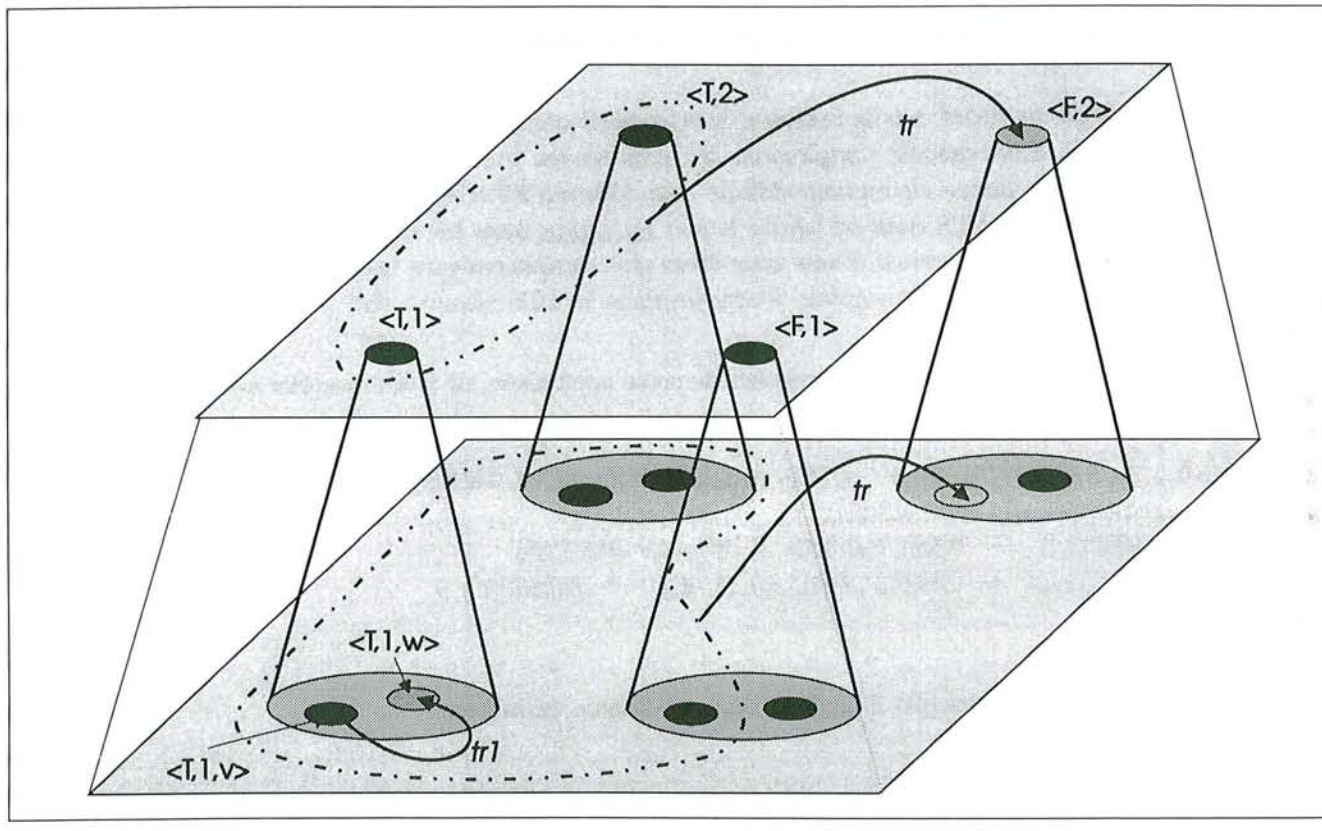

FIG. 4.4 - . *Exemple de projection d'un LTS sur un LTS étendu*

Ces règles spécifient la composition par héritage simple non strict, car elles autorisent la redéfinition d'une transition dans un système de transitions étiquetées qui hérite d'un autre LTS. En effet, l'ensemble des transitions calculées ci-dessus ne garde que celles à nouveau spécifiées dans le sous-système.

Héritage simple strict

Si nous voulons prendre en compte l'héritage strict dans la construction par héritage, il faut interdire la rédéfinition de transitions et garantir par construction, le respect des contraintes d'héritage strict. Ces contraintes ne portent à ce niveau que sur les transitions. Les deux premiers types de transistions ne posent pas problèmes. Il suffit de reconsidérer la règle de construction énoncée dans la formule 4.9.

On peut définir la contrainte d'héritage strict sur les transitions définies à la fois dans A et A' comme suit:

$$\begin{aligned}\forall\ t_{res}\ \in\ T_{A_{res}},\ \exists\ t\ \in\ T_A, \exists\ t'\ \in\ T_{A'}:\\ LABEL(t)\ =\ LABEL(t')\ =\ LABEL(t_{res})\ \Rightarrow\\ PRE(PROJ(t,A,A'))\ \in\ PRE(t_{res})\ \wedge\ POST(t_{res})\ \in\ POST(PROJ(t,A,A'))\end{aligned} \tag{4.11}$$

Cette contrainte spécifie que toute transition construite à partir d'une définition dans A, également spécifiée dans A', doit avoir un ensemble de départ au moins égal à l'ensemble de départ dans A, projeté sur A'. Son ensemble d'arrivée doit être au plus égal à la projection de l'ensemble d'arrivée dans A sur A'.

La dernière règle ne décrit pas la manière dont on doit composer par héritage les transitions définies dans les deux LTS. Elle spécifie simplement les contraintes sur les transitions résultantes pour que l'héritage garantisse la clause de rigueur définie dans [Meyer 92]. Cette règle impose en effet que toute transition tirable dans le LTS dont on hérite le soit au moins dans les mêmes conditions dans tout LTS qui en hérite. L'ensemble d'arrivée d'une transition doit également être respecté dans tout LTS étendu par héritage, c'est à dire que l'ensemble d'arrivée dans le LTS résultat soit plus restreint que dans le LTS dont on hérite.

Afin de garantir par construction le respect de cette contrainte, on peut modifier la règle de construction 4.9 de la manière suivante:

$$\begin{aligned}T_3\ =\ \{\ t_{res}\mid\ \exists\ t\ \in\ T_A,\ \exists\ t'\ \in\ T_{A'}:\ LABEL(t)\ =\ LABEL(t')\ \Rightarrow\\ LABEL(t_{res})\ =\ LABEL(t)\\ \wedge\ PRE(t_{res})\ =\ PRE(PROJ(t,A,A'))\ \cup\ PRE(t')\\ \wedge\ POST(t_{res})\ =\ POST(PROJ(t,A,A'))\ \cap\ POST(t')\ \}\end{aligned} \tag{4.12}$$

Si on calcule T_3 sur l'exemple d'après la règle ci-dessus, on obtient:

$$\begin{aligned}T_3\ =\ \{t:\ LABEL(t)\ =\ tr\\ \wedge\ PRE(t)\ =\ \{<T,1,v>;<T,1,w>;<T,2,v>;<T,2,w>\}\\ \cup\ \{<F,1,v>\}\\ \wedge\ POST(t)\ =\ \{<F,2,v>;<F,2,w>\}\ \cap\ \{<F,2,v>\}\}\end{aligned}$$

D'où:

$$\begin{aligned}T_3\ =\ \{t\ :\ LABEL(t)=tr\\ \wedge\ PRE(t)\ =\ \{<T,1,v>;<T,1,w>;<T,2,v>;<T,2,w>;<F,1,v>\}\\ \wedge\ POST(t)\ =\ \{\ <F,2,v>\}\}\end{aligned}$$

Le calcul de $T_{A_{res}} = T_1 \cup T_2 \cup T_3$ donne:

$$\begin{aligned}T_{A_{res}}\ =\ \{t:\ (LABEL(t)=tr\\ \wedge\ PRE(t)\ =\ \{<T,1,v>;<T,1,w>;<T,2,v>;<T,2,w>;\{<F,1,v>\}\\ \wedge\ POST(t)\ =\ \{\ <F,2,v>\})\\ \vee\ (LABEL(t)=tr1\\ \wedge\ PRE(t)\ =\ \{<T,1,v>\}\\ \wedge\ POST(t)\ =\ \{\ <T,1,w>\})\}\end{aligned}$$

Cela correspond bien aux transitions du LTS A' initialement défini.

Il est à noter que l'ensemble de ces règles ne sont valables que si le LTS qui hérite ne fait qu'ajouter de nouvelles variables d'état mais n'étend pas le domaine de variables définies dans le LTS dont il hérite. Comme ceci est le cas dans GDMO, si l'on considère le domaine induit par les variables permises, nous resterons sur cette hypothèse dans la suite de ce chapitre.

L'héritage multiple

Il existe plusieurs méthodes de construction pour le cas de l'héritage strict. Toutes sont définissables sur des LTS de manière analogue à celle que nous avons utilisé pour définir l'héritage simple. Pour cela, il faut, dans un premier temps, retenir une méthode de résolution des conflits (renommage, inclusion unique, ...) et, par la suite, appliquer l'héritage simple sur chacune des super-classes dont on hérite.

Dans l'approche OSI pour la modélisation des ressources, le mécanisme retenu pour la résolution des conflits d'héritage est celui de l'inclusion unique. Cela signifie que tout attribut, action ou notification hérité de plusieurs super-classes n'est instancié qu'une seule fois dans la sous-classe. L'élément hérité apparaît donc au travers de l'héritage comme une fonctionnalité héritée d'une seule super-classe.

Dans le cadre de la thèse, nous nous sommes limités à ne considérer que l'héritage simple. L'héritage multiple peut dans le cadre OSI se ramener à un héritage simple après composition des super-classes. En effet, la norme [CCITT.X.720 92] comporte des règles strictes sur la manière dont sont résolus les conflits d'héritage. Ces règles spécifient notamment qu'une même caratéristique héritée de plusieurs super-classe n'est héritée qu'une seule fois dans la sous-classe. Pour cela, il suffit de régler les conflits au niveau des super-classe en construisant d'après les règles de composition une seule super-classe à partir de toutes celles dont hérite la sous-classe considérée puis d'appliquer les règles d'héritage simple sur cette classe.

4.3 L'héritage et la sémantique de CRS

La sémantique du langage CRS est basée sur les systèmes de transitions étiquetées. Nous établissons ci-dessous, le lien entre la sémantique initiale du langage [Schneider 90b] et les extensions proposées en vue du support de l'héritage strict.

Afin de mettre en place, pour la construction du comportement d'un objet géré, un algorithme utilisant le mécanisme d'héritage défini dans l'approche OSI, il est nécessaire de spécifier comment un système de règles peut être obtenu pour une sous-classe à partir des différentes parties qui le composent et des super-classes dont il hérite. Pour cela, nous allons définir la composition d'un système de transitions étiquetées associé à un système de règles CRS par héritage.

4.3.1 Composition par héritage des états d'un objet

Basé sur les règles présentées précédemment, nous allons spécifier la manière dont l'espace d'états d'un système de règles est construit par héritage et formaliser ceci sur la base de la sémantique opérationnelle de CRS.

Comme dans l'exemple précédent, nous allons définir comment se construit un système de règles RS_{res} à partir d'un système de règles jouant le rôle d'une super-classe que nous appellerons RS_{class} et d'un système de règles qui hérite les fonctionnalités de RS_{class} et que nous appellerons $RS_{subclass}$. Ici aussi, le résultat obtenu (RS_{res}) est équivalent au système $RS_{subclass}$ après application de la construction.

L'ensemble des variables d'une sous-classe est obtenu par inclusion des attributs de la super-classe

$$SV(RS_{class}) = \{ sv_{class_1}, sv_{class_2}, ..., sv_{class_n} \} \quad (1)$$

$$SV(RS_{subclass}) = \{ sv_{subclass_1}, sv_{subclass_2}, ..., sv_{subclass_m} \} \quad (2)$$

$$SV(RS_{res}) = SV(RS_{class}) \cup SV(RS_{subclass}) \quad (3)$$

$$StatVal(RS_{class}) = DOM(PERM(sv_{class_1})) \ ... \ \times DOM(PERM(sv_{class_n})) \quad (4)$$

$$StatVal(RS_{res}) = StatVal(RS_{class}) \times StatVal(\ SV(RS_{subclass}) \setminus SV(RS_{class})) \quad (5)$$

FIG. 4.5 - . *Construction de l'espace d'états d'une sous-classe par héritage*

dans la sous-classe. Cette règle s'applique de manière récursive de la sous-classe envisagée jusqu'à la racine de l'arbre d'héritage, en l'occurence le système *top*. La seule contrainte pour l'application de cette règle est que pour tout attribut hérité d'une super-classe déjà présent dans la sous-classe, l'ensemble des valeurs permises est identique dans les deux classes.

Le calcul de l'espace d'états du système de transitions étiquetées associé à un système de règles obtenu par héritage RS_{res} est spécifié dans la figure 4.5. L'espace d'états est construit à partir des variables qui composent le système de règles. Ces variables sont notées $SV(R_x)$. L'espace d'états (noté $StatVal(RS_x)$) d'un système de règles à n variables d'état est défini par le produit cartésien des domaines des valeurs permises de toutes les variables cf. *(4)* Fig. 4.5.

L'espace d'états $StatVal(RS_{res})$ des variables du système résultat est obtenu par le produit cartésien de l'espace d'états de la super-classe $StatVal(RS_{class})$ et celui de la sous-classe restreint aux variables non définies dans la super-classe cf. *(5)* Fig. 4.5.

Ceci permet de n'inclure pour une variable donnée, qu'une seule occurence de celle-ci dans la sous-classe.

En résumé, l'ensemble des variables du système résultant $SV(RS_{res})$ est égal à l'union de l'ensemble des variables de la super-classe et celui de la sous-classe cf. *(3)* Fig. 4.5. A partir de là, l'espace d'états est égal au produit cartésien des domaines des variables qui le composent. Cet espace peut être calculé à partir de ceux des deux systèmes impliqués dans l'héritage.

Une spécification CRS comporte, en plus des variables dites d'état, un certain nombre de variables locales dont la durée de vie est celle de l'exécution d'une règle. Ces variables entrent tout naturellement en compte dans le calcul des états du système de transitions étiquetées associé à un système de règles.

L'ensemble des variables locales d'un système de règles est appelé $LV(RS_x)$. L'ensemble des variables locales du système jouant le rôle de la super-classe est donc l'ensemble $LV(RS_{class})$ cf. *(1)* Fig. 4.6. Celui de la sous-classe est noté $LV(RS_{subclass})$ cf. *(2)* Fig. 4.6.

Pour un système de règles donné, le domaine $LocVal(RS_x)$ associé aux variables locales est calculé de la même manière que le domaine des variables dites d'états, c'est à dire comme le produit cartésien des domaines de chacune des variables locales du système (voir figure 4.6). Il participe au calcul des états S du système de transitions étiquetées associé au système de règles.

L'espace d'états $LocVal(RS_{class})$ associé aux variables locales se calcule comme le produit cartésien des domaines de chacune des variables locales du système $DOM(lv_{class_1})$ *(3)*.

L'ensemble des variables locales $LV(RS_{res})$ du système résultant est égal à l'union de l'ensemble

$LV(RS_{class}) = \{ lv_{class_1}, lv_{class_2}, ..., lv_{class_m} \}$ *(1)*

$LV(RS_{subclass}) = \{ lv_{subclass_1}, lv_{subclass_2}, ..., lv_{subclass_p} \}$ *(2)*

$LocVal(RS_{class}) = DOM(lv_{class_1}) \times DOM(lv_{class_2}) \ldots \times DOM(lv_{class_m})$ *(3)*

Si la classe $RS_{subclass}$ hérite de la classe RS_{class} alors

$LV(RS_{res}) = LV(RS_{class}) \cup LV(RS_{subclass})$ *(4)*

$LocVal(RS_{res}) = LocVal(RS_{class}) \times LocVal(\ LV(RS_{subclass}) \setminus LV(RS_{class}))$ *(5)*

FIG. 4.6 - . *Composition de l'espace des variables locales d'une sous-classe par héritage*

des variables locales de la super-classe et de celui de la sous-classe *(4)*. L'espace d'états du LTS associé au système de règles RS_{res} (résultant de la construction de RS_{class} et $RS_{subclass}$), relatif aux variables locales, est obtenu en faisant le produit cartésien de l'espace d'états des variables locales de la super-classe et celui des variables locales de la sous-classe restreint aux variables non définies dans la super-classe *(5)*.

L'espace d'états d'un LTS, associé à un système, de règles est obtenu en faisant le produit cartésien de l'espace calculé pour les variables dites d'état et de celui calculé pour les variables locales.

4.3.2 Construction par héritage des transitions d'un système CRS

Le calcul des transitions du LTS associé à un système de règles CRS est plus complexe que le calcul de son espace d'états. Comme nous l'avons défini dans la section précédente, l'ensemble des transitions qui forment le comportement d'un objet est défini à partir des trois ensembles de transitions suivants:

1. ensemble des transitions héritées de la super-classe et non présentes dans la sous-classe,
2. ensemble des transitions définies dans la sous-classe et non présentes dans la super-classe,
3. ensemble qui regroupe les extension dans la sous-classe de transitions déjà existantes dans la super-classe.

Le premier ensemble décrit le cas d'héritage de transitions de la super-classe non spécialisées dans la sous-classe. Dans ce cas, l'inclusion est réalisée sans aucune restriction dans la sous-classe (ce qui correspond au calcul de l'ensemble T_1 dans la section précédente). Le second ensemble inclut la définition du comportement de transitions spécifiées dans la sous-classe sans l'être dans la super-classe. Ce cas ne présente aucune difficulté, car il suffit de garder ces transitions dans le système de transitions étiquetées associé à la sous-classe (ensemble T_2).

Le troisième ensemble décrit les transitions issues de la super-classe qui sont spécialisées dans la sous-classe (ensemble T_3). Afin de garantir les contraintes de l'héritage strict défini dans l'approche OSI, il faut garder ces transitions tout en tenant compte des spécialisations qui sont faites au niveau de la

sous-classe. Ceci peut se faire en construisant pour chacune d'elle sa condition de déclenchement comme la disjonction des conditions de déclenchement spécifiées dans la super-classe et dans la sous-classe. Les effets d'une transition sont construits par la conjonction entre les effets spécifiés dans la super-classe et ceux de la sous-classe.

$$
\begin{array}{l}
RULES(RS_{res}) = \\
\{\ r_{class} : Rule \mid r_{class} \in RULES(RS_{class}) \\
\quad \wedge\ (\forall\ r_{subclass} \in RULES(RS_{subclass}) : \\
\qquad NAME(r_{class}) \neq NAME(r_{subclass})) \qquad (1) \\
\} \\
\cup \\
\{\ r_{subclass} : Rule \mid r_{subclass} \in RULES(RS_{subclass}) \\
\quad \wedge\ (\forall\ r_{class} \in RULES(RS_{class}) : \\
\qquad NAME(r_{subclass}) \neq NAME(r_{class})) \qquad (2) \\
\} \\
\cup \\
\{\ rule : Rule \mid r_{class} \in RULES(RS_{class}),\ r_{subclass} \in RULES(RS_{subclass}) : \\
\quad NAME(r_{class}) = NAME(r_{subclass}) = NAME(rule) \\
\quad \wedge\ cond_{rule} = (cond_{rule_{class}} \vee cond_{rule_{subclass}}) \\
\quad \wedge\ effects_{rule} = (effects_{rule_{class}} \wedge effects_{rule_{subclass}}) \qquad (3) \\
\}
\end{array}
$$

FIG. 4.7 - . *Construction des transitions d'un système de règles par héritage*

Au niveau de la sémantique de CRS, les règles de construction de l'ensemble des transitions d'une sous-classe par héritage sont décrites dans la figure 4.7. Le lien d'héritage est défini sur la base des noms des transitions ($NAME(rule)$).

L'ensemble des règles du système résultant $RULES(RS_{res})$ est composé de l'union des règles de la super-classe qui ne sont pas présentes dans la sous-classe *(1)* 4.7, des règles de la sous-classe non présentes dans la super-classe *(2)* et de la composition des règles de la super-classe spécialisées dans la sous-classe *(3)*. On retrouve ici la composition des transitions à partir des trois sous-ensembles T_1, T_2 et T_3. La composition des règles de la super-classe, spécialisées dans la sous-classe, se fait de la manière suivante:

- toute règle de la super-classe spécialisée dans la sous classe voit sa pré-condition $cond_{rule}$ affaiblie par disjonction de la condition de la sous-classe ($cond_{rule_{subclass}}$) et de la condition de la super-classe ($cond_{rule_{class}}$). On retrouve à ce niveau le concept d'union des états de départ qui se traduit sur les prédicats CRS par une disjonction sur les pré-conditions.
- toute règle de la super-classe spécialisée dans la sous classe voit sa post-condition $effects_{rule}$ renforcée par conjonction de la post-condition dans la sous-classe ($effects_{rule_{subclass}}$) et de la post-condition dans la super-classe ($effects_{rule_{class}}$). On retrouve ici le concept d'intersection des ensembles d'arrivée qui se traduit sur les prédicats CRS par une conjonction des effets d'une transition.

Calculer d'après le schéma ci-dessus, les transitions d'un LTS associé à un système de règles qui hérite un certain nombre de fontionnalités, ne permet pas de garantir entièrement le respect des contraintes d'héritage strict au niveau de CRS.

Nous avons défini les règles de construction de l'héritage sur les noms des règles (ou transitions) mais pas sur les événements qui représentent en fait les opérations invoquées sur un système de règles. Or,

d'après les règles de construction, rien n'empêche de construire dans la sous-classe une règle de nom différent qui comporte dans sa pré-condition l'occurence d'une opération de la super-classe et dans sa post-condition, des états qui ne forment pas une restriction de la post-condition de l'opération dans la super-classe. Ceci enfreint la contrainte d'héritage strict car la règle de la sous-classe n'est pas composée avec celle de la super-classe. La figure 4.8 illustre un exemple d'un tel problème.

```
super-classe:
    r1;
    COND: OP.action(param)
          AND state1 = 1;
    EFFECTS: state2 = param;

sous-classe:
    r1;
    COND: OP.action(param)
          AND (state1 = 1
               OR state1 = 2 );
    EFFECTS: state2 = param;

    r2:
    COND: OP.action(param)
          AND state1 = 2;
    EFFECTS: TRUE;
```

FIG. 4.8 - . *Un exemple de violation de contrainte d'héritage strict*

Ici, une règle de la super-classe est affinée dans la sous-classe dans deux règles. L'héritage est basé sur le lien de nommage entre règles ce qui entraîne que la règle `r2` qui contrôle, tout comme la règle `r1`, l'invocation de l'opération `action()` n'hérite pas des fonctionnalités décrites dans la super-classe. De plus, la post-condition de la règle `r2` viole la contrainte de l'héritage strict en affaiblissant la post-condition liée à l'occurence.

Pour éviter l'occurence de tels problèmes il faut que l'indéterminisme associé à l'occurence d'un événement soit défini dans la super-classe. Il faut également interdire l'ajout d'indéterminisme pour un événement de la super-classe dans la sous-classe. De ce fait la situation de la figure 4.8 est interdite.

Il faut donc définir sur la spécification de règles, une contrainte qui empêche la redéfinition de comportement associé à l'occurence d'événements existants dans la super-classe sans pour autant interdire au sein de la super-classe l'indéterminisme lié à un événement. Au niveau de la sémantique de CRS, cette contrainte peut se traduire facilement de manière formelle, cf. figure 4.9.

$$\forall\ r_i \in RULES(RS_{class}),\ \nexists\ r_j \in RULES(RS_{subclass}) :$$
$$NAME(r_i) \neq NAME(r_j) \wedge ACTREQ(COND_{r_i}) = ACTREQ(COND_{r_j})$$

FIG. 4.9 - . *La contrainte sur les pré-conditions de sous-classes*

La contrainte de la figure 4.9 se traduit de manière informelle de la manière suivante: quelque soit la règle r_i de la super-classe RS_{class}, il n'existe aucune règle r_j dans la sous-classe $RS_{subclass}$ telle que r_i et r_j aient un nom différent et les requêtes de leurs pré-conditions soient indentiques ($ACTREQ(COND_{r_i}) =$

$ACTREQ(COND_{r_i})$). $ACTREQ(COND_{r_x})$ représente les événements spécifiés dans la pré-condition de la règle r_x. Sur l'exemple de la figure 4.8, $ACTREQ(COND_{r_1}) = \{< OP, setReq(param) >\}$.

4.4 L'héritage et les portes de communication

Pour permettre de généraliser le concept d'héritage à l'ensemble des composants d'une spécification, il faut également définir la sémantique de l'héritage pour les portes de communication:

- une porte hérite toutes les caractéristiques de sa ou de ses super-portes. Ces caractéristiques sont le mode de communication, les paramètres qui y sont définis ainsi que l'ensemble des événements qui peuvent s'y produire.
- une porte de communication peut étendre les fonctionnalités d'une autre classe de porte en définissant de nouveaux paramètres, en ajoutant de nouveaux événements ou en étendant des événements existants par adjonction de nouveaux paramètres.
- en aucun cas, une porte définie par héritage ne peut redéfinir le mode de communication de la porte dont elle hérite.

Dans la sémantique opérationnelle de CRS, une porte de communication est définie par son nom, le type de communication (synchrone, asynchrone) et l'ensemble des événements qui peuvent s'y produire. Cet ensemble d'événements est modélysé comme un ensemble de couples du type `(etyp,epar)` où `etyp` représente le type d'un événement (c'est à dire son nom) et `epar` le domaine de l'ensemble de ses paramètres.

$$
\begin{aligned}
inh(g,g') \Leftrightarrow\ & ETYPE(g) = {}'ETYPE(g) \cup ETYPE(g') \\
& \wedge\ \forall\ etyp \in ETYPE(g) : \\
& \quad EPAR(g,etyp) = {}'EPAR(g,etyp) \cup EPAR(g',etyp)
\end{aligned}
$$

FIG. 4.10 - . *Construction par héritage des éléments d'une porte de communication*

L'ensemble des types d'événements d'une porte de communication $ETYP(g)$ est, par héritage, obtenu par union des types spécifiques à la porte qui hérite ($'ETYPE(g)$) et des types d'événements de la porte dont elle hérite ($ETYPE(g')$). Cette construction est spécifiée dans la figure 4.10. L'ensemble des paramètres $EPAR(g, etyp)$ d'un type est obtenu par l'union des valeurs des paramètres dans la porte héritée et celles des paramètres dans la porte qui hérite.

Pour permettre la mise en place d'un mécanisme d'héritage sur les portes de communication, il faut également définir des portes de bases qui vont servir de base d'héritage pour toutes les autres. Nous avons, dans le formalisme GDMO, défini deux bases de portes de communication, l'une synchrone et l'autre asynchrone. Toutes les portes définies dans une spécification GDMO sont définies comme spécialisation d'une de ces deux portes. Nous verrons la syntaxe GDMO associée dans le chapitre suivant, mais nous pouvons dès maintenant caractériser ces portes de manière formelle.

Comme le précise la figure 4.11, les deux bases ($SyncGate, AsyncGate$) ne comportent pas de paramètres ni d'événements. En fait, chacune ne définit que la caractéristique de communication qui est soit l'échange synchrone, soit la communication asynchrone. Les portes qui en hériteront comporteront les événements et paramètres qu'elles supportent.

$$
\begin{array}{l}
ETYPE(SyncGate) = \oslash \\
EPAR(SyncGate, \oslash) = (\oslash, \oslash) \\
ETYPE(AsyncGate) = \oslash \\
EPAR(AsyncGate, \oslash) = (\oslash, \oslash)
\end{array}
$$

FIG. 4.11 - . *Formalisation des bases de portes de communication*

4.5 Premiers éléments de répartition du comportement

Nous avons jusqu'à présent étendu le formalisme CRS avec des mécanismes liés à l'héritage strict. Ceci était nécessaire pour envisager toute intégration dans GDMO des mécanismes de règles. Avant de présenter cette intégration, il reste un point délicat à traiter. Il concerne la répartition des informations de comportement sur différents formulaires d'un même objet dans GDMO. Nous avons déjà évoqué ce problème dans le chapitre 1 de cette seconde partie en énoncant le souhait que la répartition des informations relatives au comportement soit maintenue dans l'approche que nous proposons pour la formalisation.

Le but de cette section n'est pas d'imposer une répartition des descriptions de comportement dans GDMO. Elle n'a pas non plus l'intention de décrire l'algorithme de collecte qui, en fonction des répartitions effectives dans la syntaxe étendue de GDMO, permettra de construire un comportement homogène pour tout objet géré. La répartition et l'algorithme seront présentés dans le chapitre suivant. Le but est simplement d'énumérer les possibilités offertes par le mécanisme de règles de CRS pour répartir une information sur plusieurs parties.

4.5.1 Les mécanismes existants

Nous avons dans la section 3.6 du chapitre précédent, présenté les mécanismes de **contexte** et de **vue** qui permettent, pour le premier, de regrouper au sein d'une spécification des règles et informations qui décrivent des aspects communs d'un déroulement, et pour le second, de séparer une règle en groupant des informations relatives à un même sujet au sein de sous-règles.

Ces deux mécanismes très intéressants nous permettent d'envisager un même type de regroupement et de séparation dans les différents formulaires GDMO. Plus précisément, on peut regrouper des règles qui traitent une partie commune dans le comportement de modules GDMO. Un exemple de cette application est d'avoir au sein d'un module qui comporte une action, l'ensemble des règles qui en décrivent son comportement. On retrouve ici un concept analogue à celui de contexte tel qu'il apparaît dans CRS.

On peut également penser à éclater une règle sur plusieurs formulaires de comportement. Dans ce cas, chaque partie de la règle définit une caractéristique du comportement dans un contexte donné (module, action, ...). Cet éclatement est souhaitable dans plusieurs cas et permet de définir le concept de raffinement comme nous l'avons souhaité dans le chapitre 1 de cette partie.

Prenons à titre d'exemple une règle liée au comportement d'une action. Une partie du comportement est définie dans le formulaire d'action, donc on trouve la règle dans ce formulaire. Lorsque l'action est référencée dans un module, son comportement peut être affiné (lien avec les attributs du module). Donc la règle décrivant le comportement de l'action est spécialisée dans le module et seules les extensions de cette dernière y sont présentées. Il en est de même au niveau de l'objet géré où sont décrites les

interactions entre les modules. On retrouve ici un comportement à trois niveaux, chaque niveau pouvant amener des informations additionnelles sur le comportement de l'action. Ceci est une répartition proche des vues dans CRS.

4.5.2 Répartition et collecte

La souplesse des prédicats de la logique du premier ordre utilisés dans CRS nous permet d'envisager la conception d'un algorithme de collecte de comportement pour GDMO aussi simple que celui existant pour les mécanismes de contexte et de vue dans CRS. Dans les contextes, tout comme dans les vues, le couplage des différentes parties se fait par conjonction logique.

La collecte, c'est à dire la composition des règles d'un système à partir de différents formulaires, devra se faire par couplage logique (conjonction $\wedge$ ou disjonction $\vee$) des différentes parties d'une règle. Ce couplage se fera sur la base des noms de règles. Pour définir quel type de couplage devra être appliqué dans quel contexte, il nous faut d'abord définir la répartition dans GDMO, objet du chapitre suivant.

Les quelques constatations que nous avons faites dans cette section vont servir à appliquer les règles de répartition du chapitre 1 de cette partie, dans le formalisme étendu présenté dans le chapitre suivant. L'algorithme de collecte que nous développerons sur cette répartition sera basé sur les concepts présentés ici.

4.6 Résumé

Comme la technique CRS ne comporte pas le mécanisme d'héritage, nous avons dans ce chapitre défini ce mécanisme de manière formelle sur les systèmes de règles (états et transitions) et sur les portes de communication. L'approche OSI ayant retenu un mécanisme d'héritage strict pour les objets gérés, nous avons étudié les contraintes liées à celui-ci et nous les avons formalisées sur la base de la sémantique opérationnelle de CRS. La version étendue de la FDT ainsi obtenue va pouvoir servir de base à l'intégration dans GDMO.

Finalement nous avons dressé quelques lignes directrices pour la répartition d'une description de comportement basée sur des règles. Ces lignes vont nous permettre, en plus de la définition de la répartition du comportement dans les différents formulaires de GDMO, de construire un algorithme de collecte qui, à partir de différentes spécifications de comportement, permettra de construire un comportement homogène et le système de règles associé à tout objet GDMO.

Dans le chapitre suivant, nous allons montrer comment le langage de spécification GDMO peut être étendu, d'une part par des spécifications d'interfaces, et d'autre part par l'introduction des formulaires de règles Cette intégration nous permettra de répondre à l'exigence de compatibilité de notations retenue dans notre approche.

5
La formalisation du comportement dans GDMO

Dans le chapitre précédent, nous avons présenté la technique de description formelle CRS ainsi que les extensions apportées pour supporter des spécifications orientées-objets. Or, l'une des contraintes que nous nous étions fixés dès le début de ces travaux était l'intégration de la description du comportement dans le formalisme GDMO. C'est l'objet de ce chapitre.

5.1 L'organisation de la formalisation

Dans les chapitres précédents, nous avons insisté sur le besoin de formaliser le comportement en compatibilité avec le formalisme GDMO. Nous avons aussi souligné la nécessité de mettre en place un algorithme de collecte qui permet de construire à partir des différents formulaires de comportement présents dans GDMO, le comportement global associé à un objet. Pour l'approche retenue dans cette thèse, ces deux points ont été considérés. Le formalisme GDMO étendu avec les mécanismes que nous préconisons et qui satisfait les exigences fixées s'appelle **LOBSTERS** (**L**anguage for **O**bject **B**ehaviour **S**pecification based on **T**emplates and **E**xtended **R**ule **S**ystems) [Festor 94]. Il sera présenté dans ce chapitre.

La présentation introduit les différentes extensions apportées aux formulaires par LOBSTERS. Après la description du nouveau formulaire de description d'objet géré, nous introduirons la formalisation des conditions de présence des modules conditionnels. Par la suite, nous nous attacherons à motiver les différents formulaires de comportement formalisés, allant des plus isolés (les formulaires de notifications et d'action), vers les formulaires fédérateurs de comportement tels que les formulaires de comportement de module et finalement de classe d'objet géré.

La présentation de ces différents formulaires, illustrée de quelques exemples, nous permettra de concevoir l'algorithme de collecte de comportement que nous envisageons sur le formalisme pour **construire** le comportement homogène d'un objet géré à partir de ses différentes parties. La présentation de cet algorithme sera faite dans la dernière partie de ce chapitre.

5.2 Le comportement d'objet géré

Un objet géré est composé d'un ensemble de paquetages ayant chacun un comportement propre. La norme actuelle ne propose pas de description de comportement associée à un objet de gestion. Cependant, il est nécessaire de décrire les interactions entre les différents paquetages en dehors de ceux-ci. C'est pourquoi nous avons étendu le formulaire de spécification de classe d'objet géré par deux composantes

qui sont, d'une part un formulaire de comportement [1], et d'autre part un formulaire de spécification d'interface.

5.2.1 Le formulaire

Dans le premier chapitre de cette partie, nous avons présenté la spécification des conditions de présence comme l'un des facteurs d'erreurs dans les spécifications d'objets gérés. Dans l'extension de GDMO, nous avons introduit une description formelle des conditions de présence entre modules conditionnels. Cette condition formelle est intégrée au formalisme. Elle sera détaillée dans la section suivante.

```
<class-label> MANAGED OBJECT CLASS
[ DERIVED FROM <class-label> [,<class-label>]*;
]
[ CHARACTERIZED BY <package-label> [,<package-label>]*;
]
[ CONDITIONAL PACKAGES <package-label> [,<package-label>]*
                              PRESENT IF  condition
                    [,<package-label> [,<package-label>]*
                              PRESENT IF  condition]*;
]
[  INTERFACES <gate-name>: <gate-label> [<gate-name>:<gate-label>]*;
]
[  BEHAVIOUR <moc-behaviour-label>;
]
REGISTERED AS <objet-identifier>;

 condition ->  <delimited-string>
                  | "@  formal-cond @ <undelimited-string>"
```

FIG. 5.1 - . *Le formulaire étendu de classe d'objet géré*

Le formulaire de spécification d'objet géré étendu comporte en plus de l'extension des spécifications de condition de présence, deux champs supplémentaires (voir 5.1). Dans le premier (champ **INTERFACE**), les interfaces auxquelles l'objet géré accède, soit pour y émettre, soit pour y consommer des messages, sont référencées. La liste de ces interfaces définit l'ensemble des interactions possibles entre l'objet géré et son entourage.

Le second champ que LOBSTERS ajoute au formulaire de spécification de classe d'objet géré est le champ de comportement **BEHAVIOUR**. Il permet de référencer un formulaire dans lequel le comportement lié à l'objet est décrit de manière formelle, c'est à dire sous forme de règles de type CRS. Cette clause fédère les comportements de tous les modules présents dans l'objet géré et lie le comportement aux occurences d'événements. Comme ce formulaire fédère les comportements des différents modules, nous allons le présenter après ces derniers.

1. . Le choix d'introduire un formulaire de comportement au niveau même d'un objet géré résulte de la volonté de séparer les différents types de comportement (modules, objet).

Comme la norme ne prévoit pas cette possibilité, mais décrit un module obligatoire qui lui fédère le comportement de l'ensemble de l'objet, nous aurions pu formaliser le comportement d'objet au sein de ce module. Pour des raisons d'homogénéité des formulaires et de clarté des concepts, nous avons finalement défini un formulaire propre aux objets gérés

5.2.2 La formalisation des conditions de présence dans LOBSTERS

L'étude des conditions de présence de modules conditionnels dans de nombreux catalogues existants, nous a permis de constater que dans la plupart des cas (95%), la condition de présence est en liaison directe avec le support des fonctionnalités offertes par le module. Ceci dans la ressource et avec la présence ou non d'autres modules optionnels (lien d'exclusion mutuelle par exemple). Sur la base de ces constatations, nous proposons le formalisme de la figure 5.2 pour spécifier sous forme de prédicat la condition de présence d'un module conditionnel dans un objet géré. Dans le but de rester compatible avec la notation GDMO normalisée et pour pouvoir extraire ces prédicats des spécifications, les spécifications formelles des conditions de présence sont délimités par le caractère @. Lors de l'analyse des spécifications, ce délimiteur permet de distinguer la partie formelle de la partie informelle d'une spécification. Nous avons retenu le même mécanisme pour l'encapsulcation de la description du comportement dans les formulaires de GDMO.

Les variables booléennes de ce type de prédicats ne peuvent être que des références à des modules c'est à dire des <package-label> ou des clauses spéciales comme nous le verrons plus tard.

```
formal-cond ->  condition

condition -> condition1
           | ( condition )
           | condition  OR condition

condition1 -> condition2
            | condition  AND condition

condition2 -> condition3
            |  NOT condition

condition3 ->  SUPPORT
            |  PRESENT(<packagelabel>)
            |  OTHER
            |  IF condition  THEN condition  FI
            |  IF condition  THEN condition  ELSE condition  FI
```

FIG. 5.2 - . *Le modèle de formalisation des conditions de présence*

La clause **SUPPORT** permet, au sein d'une condition de présence, d'indiquer la nécessité de demander au développeur si les fonctionnalités offertes par le module le sont dans l'implantation cible. Cette clause représente donc une variable booléenne dont la valeur est à déterminer de manière interactive avec le développeur d'un objet géré lors de son implantation.

La clause **PRESENT** permet de spécifier les dépendances entre les différents modules. Cette clause est "vrai" si le module représenté par le paramètre de celle-ci est présent dans l'objet géré, sinon elle renvoie "faux".

La clause **OTHER** est une clause échappatoire qui permet au niveau de la spécification de la condition de présence de formuler l'existence d'autres conditions qui ne peuvent pas être spécifiées à l'aide des mécanismes prévus. Cette clause représente en fait un atome dont le calcul de la valeur (Vrai, Faux) est à réaliser par le concepteur de l'objet géré lors de l'implantation de celui-ci.

Tout type de prédicat peut donc être construit à partir de ces clauses de base. Nous allons dans les

lignes suivantes donner un exemple d'utilisation sur la spécification du système informatique `computer-System` présenté dans le chapitre 4 de la première partie.

La figure 5.3 nous présente un exemple de l'emploi de ce type de prédicats comme exemple de formalisation des conditions de présence de modules conditionnels. Il apparaît dans cet exemple que la formalisation est, d'une part totalement compatible avec le formalisme GDMO tel qu'il est normalisé, et d'autre part que la spécification des conditions de présence est très facile à réaliser. On remarque également sur cet exemple la référence aux interfaces `Management`, `Notification` et `External` qui permettent au système l'échange d'informations avec son entourage.

```
BasicComputerSystem MANAGED OBJECT CLASS
  DERIVED FROM top;
  CHARACTERIZED BY computerSystemPkg;
  CONDITIONAL PACKAGES
   peripheralNamePkg       PRESENT IF
        " @ SUPPORT AND NOT(PRESENT(peripheralListPkg)) @",
   peripheralListPkg       PRESENT IF
        " @ SUPPORT AND NOT(PRESENT(peripheralNamePkg)) @",
   processingEntityNamePkg PRESENT IF
        " @ SUPPORT AND NOT(PRESENT(processingEntityListPkg)) @",
   processingEntityListPkg PRESENT IF
        " @ SUPPORT AND NOT(PRESENT(processingEntityNamePkg))@",
   systemTimePkg           PRESENT IF " @ SUPPORT @",
   upTimePkg               PRESENT IF " @ SUPPORT @",
   userLabelPackage        PRESENT IF " @ SUPPORT @",
   usageStatePkg           PRESENT IF " @ SUPPORT @ usage can be detected";
 INTERFACES Management   : StandardManagementGate,
            Notification: StandardNotificationGate,
            External     : StandardExternalGate;
 BEHAVIOUR basicComputerSystemBehaviourPackage;
 REGISTERED AS { forum-objectClass  1 };
```

FIG. 5.3 - . *Un exemple de formalisation des conditions de présence*

Le but de formaliser les conditions de présence des modules optionnels est double. Il permet, lors de l'implantation d'un objet géré, d'automatiser partiellement le processus de génération de code d'un objet et de générer le code permettant la validation des requêtes de création d'objet géré dans un environnement opérationnel.

Lors de la génération de code pour un objet donné, la clause **SUPPORT** permet au système de génération de détecter le besoin de demander au développeur si le module optionnel qui la comporte est supporté dans l'implantation cible. Grâce à cette formalisation, il est donc possible de gagner en automatisation sur le processus de développement.

Une instance d'objet géré dans un système opérationnel peut être créée au travers d'une requête de création CMIP. Cette requête comporte la liste des modules optionnels à instancier. Afin de valider la requête, il est nécessaire de vérifier que la combinaison des modules optionnels soit correcte vis-à-vis de la spécification de l'objet géré. Il est donc aussi nécessaire de mettre en place une telle vérification dans la procédure de création. Comme les conditions de présence sont formelles dans la spécification, il est possible de générer automatiquement le code pour la vérification de la validité. Ici aussi, un gain d'automatisation est obtenu face à la version initiale de la norme.

Nous reviendrons plus en détails sur la génération interactive et sur le code généré dans le chapitre 2 de la troisième partie.

5.3 Le formulaire d'interface

Dans la section précédente, nous avons montré que le formulaire d'objet géré comportait un champ d'interface où sont référencées toutes les interfaces qui permettent à l'objet de fournir un service et d'accéder éventuellement aux services fournis par d'autres objets gérés. Cette section présente le formulaire qui permet de spécifier de telles interfaces ou portes de communication.

5.3.1 Le formulaire

Le formulaire d'interface permet de spécifier de manière formelle l'ensemble des caractéristiques d'une porte de communication, c'est à dire le mode de communication supporté, les événements s'y produisant et les paramètres véhiculés. Dans le but de rester fidèle au principe d'héritage, une porte de communication peut être définie comme entité unique ou comme spécialisation d'une interface existante (concept d'héritage appliqué aux interfaces). La figure 5.4 comporte la spécification de ce formulaire d'interface.

```
<gate-label> GATE
   DERIVED FROM <gate-label> [,<gate-label>]*;
[  USAGE usage;
]
[  DEFINITION <delimited-string>;
]
   DECLARATIONS [ paramName : paramType;]+
   EVENTS [ eventName([paramName [, paramName]*]);]+
[  REGISTERED AS <object-identifier>; ]
;

 usage ->   MANAGEMENT
          | RESSOURCE
          | MIB
```

FIG. 5.4 - . *Le formulaire de spécification d'interface*

Le formulaire d'interface permet de définir une porte de communication en spécifiant celles dont elle hérite (**DERIVED FROM**) et en ajoutant de nouveaux événements et paramètres. La clause **USAGE** permet de définir la visibilité d'une interface vis-à-vis de l'extérieur. En effet, une interface d'objet peut servir à la fois au système de gestion (**MANAGEMENT**), à d'autres objets gérés (**MIB**) ou à accéder à une ressource réelle (**RESSOURCE**).

La partie déclaration **DECLARATIONS** comporte la liste des paramètres des événements de la porte. La partie événement **EVENTS** comporte la liste des événements et de leurs paramètres qui peuvent transiter par la porte. La spécification d'une porte ne peut pas redéfinir le mode de communication hérité, mais permet simplement d'ajouter de nouveaux paramètres dans la clause **DECLARATIONS** et de nouveaux événements dans la clause **EVENTS**.

```
StandardNotificationGate GATE
  DERIVED FROM AsyncGate;
  USAGE MANAGEMENT
  DEFINITION " This interface is a standard interface which
                allows state change notifications to be issued";
  DECLARATIONS
        stateSet : StateSet;
  EVENTS
        stateChange(stateSet);
;
```

FIG. 5.5 - . *Un exemple de spécification d'interface*

La figure 5.5 comporte un exemple de spécification d'interface. Celle-ci définit l'interface associée à un objet de type système informatique. Au travers de cette interface de type asynchrone, appelée `StandardNotificationGate`, transitent toutes les notifications de changement d'état du système. Cette interface hérite des fonctionnalités de base de l'interface de base `AsyncGate` décrite dans la section suivante.

La notation proposée ici est proche de celle de GDMO qui permet en fait de définir des portes de communication compatibles avec les extensions que nous avons proposées à CRS.

5.3.2 Les bases d'interfaces

Dans le chapitre précédent, nous avons défini deux types de portes de communication qui servent de bases d'héritage pour toutes les autres interfaces définies dans toute spécification LOBSTERS. Ces deux portes sont:

- *SyncGate*: base des portes synchrones. Toutes les portes synchrones existantes dans une spécification de base d'informations de gestion héritent des fonctionnalités de cette porte.
- *AsyncGate*: base des portes asynchrones. Toutes les portes asynchrones d'une spécification doivent hériter de cette porte.

```
SyncGate GATE
  MODE SYNCHRONOUS;
  DEFINITION "This interface is the anchor from which all
                synchronous gates are derived";
  REGISTERED AS { smi2MSyncronousGateClass 1 };
```

FIG. 5.6 - . *La base d'interface synchrone*

Chacune de ces portes est définie en LOBSTERS à l'aide d'un formulaire de porte comme le montrent les figures 5.6 et 5.7. En fait, ces portes ne définissent que le mode de communication et sont enregistrées

dans l'arbre d'enregistrement des objets gérés. L'usage, ainsi que les événements et paramètres qui peuvent être échangés entre objets gérés, doivent être spécifiés dans des spécialisations de ces portes.

```
AsyncGate GATE
  MODE ASYNCHRONOUS;
  DEFINITION "This interface is the anchor from which all
                  asynchronous gates are derived";
  REGISTERED AS { smi2MAsyncronousGateClass 1 };
```

FIG. 5.7 - . *La base d'interface asynchrone*

Pour enregistrer toutes les interfaces dans l'arbre d'enregistrement, nous proposons l'introduction de deux nouvelles branches. L'une pour les portes synchrones (branche *synchronousGate (11)*) et l'autre pour les portes asynchrones (branche *asynchronousGate (12)*). Ces deux branches sont attachées à l'arc d'enregistrement *joint-iso-ccitt ms(9) function(2) part(x)*. Le terme x représente le numéro de la fonction de gestion dans laquelle une porte est définie (normes 10164-x). Si des portes sont définies dans le cadre des normes de gestion de systèmes, alors celles-ci seront référencées sous l'arc *joint-iso-ccitt ms(9) smi(3) part(x) typeDePorte (y)* où l'identificateur *typeDePorte* définit l'arc de porte synchrone ou asynchrone et y l'identificateur de la porte. A noter que l'enregistrement des portes de communication est optionnel.

5.4 La formalisation du formulaire de comportement des notifications

Le comportement de notification définit les conditions d'émission d'une notification ainsi que la manière dont les paramètres sont affectés. Le formulaire de comportement de notification que nous avons étendu permet de décrire de manière formelle les aspects du comportement d'une notification. Il est présenté dans la figure 5.8.

Ce formulaire de comportement d'une notification reste compatible avec la notation GDMO car les extensions sur le comportement ne se font qu'au sein de la description textuelle, telle qu'elle apparaît dans la norme. Cette dernière est divisée en deux parties: une partie formelle que nous allons détailler dans les lignes suivantes et une partie informelle <undelimited-string> qui permet de garder une description textuelle de la définition de la notification.

La partie formelle du comportement de notification comporte trois parties distinctes. La première (section **DECLARATIONS**) permet de définir des variables locales au contexte d'émission de notification. La deuxième partie (clause **DEFINITIONS**) permet de définir des prédicats paramétrés utilisés, soit dans la condition de déclenchement, soit dans les effets de l'émission de la notification. Le contexte de comportement compose la troisième partie de la formalisation du comportement d'une notification. Il associe à une étiquette de notification les conditions de déclenchement de la notification (clause **TRIGGER**) et les effets de l'émission de la notification sur les états de l'objet, ainsi que sur les paramètres de la notification (clause **ISSUANCE**).

La clause **TRIGGER** comporte un prédicat de la logique du premier ordre (*notification-trigger*) qui spécifie formellement les conditions d'émission de la notification. La clause **ISSUANCE** contient un prédicat (*notification-issuance*) qui définit, d'une part l'état de l'objet après émission de la notification, et d'autre part les valeurs des paramètres de la notification émise.

Comme nous le verrons par la suite, l'étiquette de notification associée aux deux clauses est nécessaire

```
<notification-behaviour-label> BEHAVIOUR
 DEFINED AS
"@
  [  DECLARATIONS [ var-decl]+]

  [ DEFINITIONS [ useful-predicate;]+]

  [ <notification-label>;
       TRIGGER  notification-trigger;
       ISSUANCE  notification-issuance;
  ]
 @ <undelimited-string>";

 var-decl -> <variable-label>: asn1Type;

 useful-predicate -> <predicate-name>(formalparameters*) ⇔ formula;
```

FIG. 5.8 - . *Le formulaire de comportement de notification étendu*

pour pouvoir spécialiser le comportement dans les formulaires de modules et d'objet géré.

La figure 5.9 nous montre un exemple de spécification de notification. Celle-ci est émise par un objet journal dès qu'un seuil (pourcentage de la capacité de stockage) est dépassé. A chaque fois qu'un ou plusieurs seuils de l'ensemble des seuils d'alarmes (`capacityAlarm`) sont dépassés, une notification, comportant la liste des seuils dépassés est émise.

La définition `average(size)` permet de calculer le taux de remplissage d'un journal à partir de la taille actuelle et de la taille maximale autorisée pour le journal. Celle définition est utilisée quatre fois dans les prédicats de comportement associé à la notification, d'où l'intérêt de la définir sous forme de prédicat générique réutilisable.

5.5 Le comportement d'action

Le comportement associé à une action doit spécifier les conditions dans lesquelles cette action est invocable, ainsi que les effets que l'exécution de celle-ci engendre sur les états de l'objet concerné. Dans LOBSTERS, nous avons défini un formulaire de comportement d'action étendu qui permet de formaliser ces éléments. Il est décrit dans la figure 5.10.

Le formulaire de comportement d'action est assez simple dans sa conception. Il comporte trois éléments dans sa partie formelle. Le premier permet de définir des variables locales aux règles qui vont spécifier le comportement. Ces variables locales sont définies dans la clause **DECLARATIONS**. La clause **DEFINITIONS** permet, tout comme pour le formulaire de comportement de notification, de définir des prédicats utilisés dans plusieurs règles de description de comportement d'action.

La différence avec le formulaire de spécification des notifications porte sur le troisième élément du formulaire qui est la clause **RULES**. Celle-ci comporte la définition des règles qui spécifient le comportement associé à l'action. Chaque règle a un nom, une condition de déclenchement **COND** et un prédicat (`predicate`) qui décrit la post-condition associée à la règle (**EFFECTS**). Les prédicats **COND** et **EF-**

```
capacityAlarmBehaviour BEHAVIOUR
 DEFINED AS
"@
   DECLARATIONS capacityAlarmParameter: CapacityAlarmParameter;
                size: INTEGER;

   DEFINITIONS average(size) ⇔
                 (size / maxLogSize)× 100;

  capacityAlarmNotification;
      TRIGGER maxLogSize ≠ 0
     ∧∃ x ∈ capacityAlarm:
         average('currentLogSize) <= x ∧  x> average(currentLogSize);
      ISSUANCE ∀ x ∈ capacityAlarm:
     average('currentLogSize) <= x ∧ x> average(currentLogSize) ⇒
     x ∈ capacityAlarmParameter.thresholds
     ∧ ...
 @ This notification is issued as soon as a capacity alarm threshold  is exceeded.";
```

FIG. 5.9 - . *Un exemple de formalisation du comportement de notification*

FECTS sont optionnels. Si l'invocation de l'action est valide dans tous les états du système, alors la clause **COND** est inutile. Si, par contre, l'exécution de l'action n'a aucune influence sur l'état de l'objet géré associé, alors la clause **EFFECTS** est optionnelle.

La figure 5.11 est un exemple de spécification de comportement d'action. L'action spécifiée est l'action `activate`[2] dont l'invocation a pour but de rendre opérationnelle une ressource.

Les règles liées à l'action indiquent qu'en cas de réussite la réponse est `successful` et qu'en cas d'échec la réponse est `failureResponse`. Il n'est pas possible à ce niveau de la spécification de lier les pré- et post-conditions de l'action aux attributs qui ne sont spécifiés que dans les modules.

On peut également noter que l'occurence de l'événement associé à l'action n'est pas décrit au niveau de l'action. Comme les interfaces sont référencées dans le formulaire de classe, les interactions sont spécifiées dans le comportement de l'objet.

Le fait de permettre la description du comportement d'une action dans le formulaire de celle-ci ne nous a, dans un premier temps, pas paru souhaitable car ce comportement dépend fortement du contexte dans lequel l'action est intégrée. Ce contexte représente en fait l'objet géré et les attributs qu'il supporte. Comme l'ensemble des attributs n'est connu qu'au niveau de l'objet géré, il nous semble préférable de ne décrire le comportement de l'action qu'au niveau du module dans lequel elle est instanciée, voire même au niveau de l'objet géré. Cependant, les catalogues existants le définissent bien souvent dans le formulaire de l'action. Aussi, pour ne pas imposer au niveau du langage, une méthode unique dans le processus de développement d'objets gérés, nous avons introduit ce formulaire de comportement formel d'action dans LOBSTERS.

2. . L'exemple de cette action tout comme l'ensemble des exemples présentés dans la suite de ce chapitre sont issus de la spécification d'un système informatique dont les paquetages ont été redéfinis pour permettre l'exploitation de tous les mécanismes offerts par LOBSTERS.
La spécification complète de ce système est donnée dans l'appendice 2 de la thèse.

```
<action-behaviour-label> BEHAVIOUR
 DEFINED AS
"@
  [ DECLARATIONS [ var-decl]+]
  [ DEFINITIONS [ useful-predicate;]+]
    RULES [ rule]+
@ <undelimited-string>";

 var-decl -> <variable-label>: asn1Type;

 useful-predicate -> <predicate-name>(formalparameters*) ⇔ formula ;

 rule -> <rule-name>;
          [ COND: predicate ;]
          [ EFFECTS: predicate;]
```

FIG. 5.10 - . *Le formulaire étendu du comportement d'action*

Son utilisation "sauvage " pose néanmoins un certain nombre de problèmes relatifs à la collecte de comportement et à l'algorithme de construction d'un système de règles associé. Nous verrons ces problèmes dans la dernière section de ce chapitre.

5.6 Les formulaires de comportement d'attribut et de paramètre

Le formulaire de comportement d'attribut doit, d'après la norme GDMO, comporter la définition de la manière dont s'appliquent les opérations de comparaison sur l'attribut. Comme nous utilisons X-ASN.1 dans LOBSTERS, et que tous les opérateurs de comparaison sont définis dans ce formalisme, la formalisation au sein du formulaire est rendue inutile.

Aussi, cette première version de LOBSTERS ne comporte pas de formulaire pour le comportement formel d'attribut. Si des besoins de description de contraintes entre attributs doivent être spécifiés, ceux-ci le seront dans les règles des modules comme nous le verrons dans les pages suivantes.

Tout comme celui d'attribut, le comportement associé à un paramètre est limité à la description de sa sémantique. Il ne nous a pas paru nécessaire de définir un formulaire spécifique pour ce type de description. Aussi, le formulaire de comportement d'un paramètre ne comporte qu'une partie informelle modélisée par une chaîne de caractères.

5.7 Le comportement de module

Le comportement de module a pour but de spécialiser le comportement de ses composants, c'est à dire ses attributs, actions, opérations et notifications. Il a également comme objectif de définir les interactions au niveau du comportement entre ces composants. Pour permettre une description formelle de ces contraintes et spécialisations, nous avons défini le formulaire de la figure 5.12

On retrouve au sein de ce formulaire les deux principales composantes formelles telles qu'elles existent

```
activate BEHAVIOUR
 DEFINED AS
"@
    DECLARATIONS
        response : ActivateActionReply;

    RULES
     activateSuccess;
        EFFECTS: response = successful;

     activateFailure;
        EFFECTS: response = failureResponse;
@ This is the behaviour associated with the activate action";
```

FIG. 5.11 - . *Un exemple de comportement d'action formalisé*

dans les formulaires de notification et d'action. Elles sont, d'une part les règles et les conditions/effets de déclenchement des notifications, et d'autre part les règles liées au comportement d'action. Naturellement, un spécifieur peut, au sein d'un module, déclarer de nouvelles variables locales dans la clause **DECLARATIONS** et définir de nouveaux prédicats utiles dans la clause **DEFINITIONS**.

Regardons tout d'abord la partie règles d'un comportement de module. Dans cette partie, doivent être décrites toute les influences des attributs du module sur les actions et opérations. On y retrouvera donc les règles qui décrivent le comportement de toutes les actions instanciées dans le module ainsi que le comportement associé à toutes les opérations possibles sur les attributs du module. Notons que les occurences des événements aux portes de communication ne sont pas spécifiées dans cette partie. Au sein des règles qui décrivent le comportement des actions se retrouve également la description des relations entre attributs. En effet, les combinaisons valides de valeurs sont spécifiées dans les post-conditions des actions et des opérations.

Au sein d'un module, tous les comportements des notifications peuvent être spécialisés. Sur le formalisme, cela se traduit par la possibilité de complèter la règle qui décrit le comportement d'une notification. C'est pourquoi, la partie formelle du comportement de module contient une clause **NOTIFICATIONS** dans laquelle des règles de comportement de notifications peuvent être définies.

Seules des actions ou des notifications référencées dans le module peuvent être décrites de manière formelle dans celui-ci.

L'exemple de la figure 5.13 montre l'utilisation des deux parties formelles du formulaire de comportement de module. Il spécifie le comportement associé à un module de système informatique simplifié qui ne comporte qu'un seul attribut d'état, en l'occurence l'attribut d'état opérationnel. Le comportement de l'action de mise en état `activate` est étendu par rapport à celui spécifié dans le comportement de l'action en raison de la présence dans le module de l'attribut d'état opérationnel. Aussi, l'action ne peut aboutir que dans le cas où l'état opérationnel affiche la valeur `disabled` et si l'action est un succès, cet état passe à `enabled`.

Le module comporte également la notification de changement d'état **stateChangeNotification** déclenchée dès que la valeur de l'attribut est affectée.

```
<package-behaviour-label> BEHAVIOUR
 DEFINED AS
"@
  [ DECLARATIONS  var-decl;]
  [ DEFINITIONS [ useful-predicate;]+]
  [ RULES
       [rule;]+
  ]
  [ NOTIFICATIONS
       [ <notification-label>;
       TRIGGER  notification-trigger;
       ISSUANCE  notification-issuance;
       ]+
  ]
 @ <undelimited-string>";

 var-decl -> <variable-label>: asn1Type;

 useful-predicate -> <predicate-name>(formalparameters*) ⇔ formula ;

 rule -> <rule-name>;
          [ COND: predicate;]
          [ EFFECTS: predicate;]
```

FIG. 5.12 - . *Extension du comportement de module*

5.8 Le comportement d'un objet géré

Le comportement associé à un objet géré a pour but de fédérer les comportements de ses différents composants qui sont en l'occurence les modules. De plus, le comportement d'un objet géré doit spécifier les interactions aux différentes portes de communication de l'objet.

Le formulaire de comportement de classe doit donc contenir un mécanisme de description de composition de règles entre modules et la possibilité de décrire dans des règles les interactions d'actions, d'opérations et de notifications aux interfaces. Pour répondre à ces besoins, nous avons conçu le formulaire de la figure 5.14.

Le premier élément du formulaire de comportement lié à un objet (**EVENTS**) permet de décrire les occurences des événements liés aux actions, opérations et notifications sur les portes de communications. C'est une partie de règles dans lesquelles les pré- et post-conditions ne se rapportent qu'à des occurences d'événements. Les pré-conditions des règles liées à des modules optionnels doivent contenir l'étiquette du module car elles ne peuvent être prises en compte que si le module est instancié. L'étiquette d'un module optionnel est considérée comme une variable booléenne dont la valeur est `Vrai` si le module est instancié, `Faux` dans le cas contraire. Cette étiquette peut de ce fait être intégrée dans les prédicats des règles.

Ces événements ne peuvent être que ceux liés aux actions et notifications référencées dans les modules de cet objet. En aucun cas une occurence d'action déjà définie dans une super-classe ne peut être redéfinie dans la sous-classe. Il en est de même pour les notifications.

```
BasicComputerSystemPackageBehaviour BEHAVIOUR
  DEFINED AS
  "@
    DECLARATIONS
         stateSet : StateSet;
    RULES
        activateSuccess;
         COND: operationalState = disabled;
         EFFECTS: operationalState = enabled;

        activateFailure;
         EFFECTS: operationalState = 'operationalState;

        disable;
         EFFECTS: operationalState = disabled;

    NOTIFICATIONS
        stateChange;
          TRIGGER: operationalState <> 'operationalState;
          ISSUANCE: IF (operationalState <> 'operationalState)
                       THEN MEMBEROF(stateSet,"OperationalStatë,operationalState)
                       FI;
 @ This is a formalized behaviour for a systemPackage module which contains an activate action,
a stateChange notification and an operationalState attribute ";
```

FIG. 5.13 - . *Exemple d'application de la formalisation du comportement de module*

La partie **RULES** peut comporter des descriptions de comportement liées à plusieurs modules pour une même règle. Il en est de même pour la partie **NOTIFICATIONS**. Dans ces parties, le spécifieur peut spécialiser des comportements de modules et des règles liées à des comportements hérités de la super-classe.

La figure 5.15 illustre l'utilisation du formulaire formel de comportement associé à un objet géré. Dans ce cas, il s'agit de l'objet géré système informatique de base étendu dans une sous-classe avec un module administratif et un module optionnel décrivant la détection de l'exploitation du système en nombres d'utilisateurs.

Dans la partie comportement de cet objet géré, on trouve l'utilisation des principales clauses du formulaire. Dans la clause **EVENTS**, on trouve la spécification des occurences d'événements aux interfaces de l'objet (exemple de l'invocation d'une action de verouillage `lock()` à l'interface de gestion `Management`). Les règles liées aux modules optionnels comportent bien l'étiquette du module qui conditionne le support de l'action (ex. `logout;`). La partie **RULES** comporte une spécialisation des comportements des modules (exemple de la règle `lock` qui influence l'état issu du module optionnel `UsageState`) ainsi qu'un affinage des comportements hérités de la super-classe (exemple de la règle `activateSuccess` dont la pré-condition est affaiblie et la post-condition renforcée). La clause **NOTIFICATIONS** spécialise le comportement associé à la notification de changement d'état héritée de la super-classe.

Une spécification complète du comportement du système informatique étendu est donnée en appendice 2 de la thèse.

```
<object-behaviour-label> BEHAVIOUR
 DEFINED AS
"@
  [ DECLARATIONS  var-decl;]
  [ DEFINITIONS [ useful-predicate;]+]
  [ EVENTS
       [rule;]+

       [ <notification-label>;]+
       ISSUANCE:  notification-trigger;
       ]+
  ]
  [ RULES
       [rule;]+
  ]
  [ NOTIFICATIONS
       [ <notification-label>;]+
        TRIGGER:  notification-trigger
       ISSUANCE:  notification-isssuance;
       ]+
  ]
 @ <undelimited-string>";
```

FIG. 5.14 - . *Extension du comportement d'un objet géré*

Une spécification du comportement peut donc, d'après les formulaires que nous avons présentés, être distribuée suivant des règles strictes. Nous allons maintenant nous attacher à concevoir un algorithme qui permet de construire le comportement au sein d'un objet. Puis, construire le comportement complet par héritage à partir des concepts énoncés dans le chapitre précédent.

5.9 L'algorithme de collecte de comportement

Dans le chapitre précédent, nous avons donné quelques pistes à suivre pour construire un algorithme de collecte de comportement homogène d'un objet géré à partir des informations réparties dans différents formulaires. Dans les sections précédentes, nous avons défini la répartition de l'information de comportement sur les différents formulaires de LOBSTERS. A partir de cette répartition et des idées présentées dans le chapitre 3 de cette partie, nous pouvons maintenant nous attacher à la construction de cet algorithme de collecte. Aussi allons nous, après une présentation des buts de cet algorithme, le définir sur la base des formulaires décrits précédemment et sur les concepts énoncés dans le chapitre précédent.

5.9.1 Le problème

Les sections précédentes ont défini les formulaires dans lesquels une description formelle du comportement était possible. A partir de cette répartition, nous désirons maintenant pouvoir, pour chaque objet géré, construire un ensemble homogène de règles de type CRS.

```
ExtendedComputerSystemBehaviour BEHAVIOUR
 DEFINED AS
"@
 EVENTS
   lock;
      COND: Management.lock();

   logout;
      COND: UsageStatePackage  AND External.logout();
        ....

 RULES
       activateSuccess;
         COND: administrativeState = locked;
         EFFECTS: administrativeState = unlocked;

        ....
       lock;
          EFFECTS: IF UsageStatePackage
                   THEN usageState = idle AND users = 0
                   FI;
        ....
 NOTIFICATIONS

       stateChange;
         TRIGGER: ( administrativeState <> 'administrativeState)
                        OR ( UsageStatePackage AND (usageState <> 'usageState));
         ISSUANCE: IF ( administrativeState <> 'administrativeState)
                        THEN MEMBER(stateSet,"administrativeState ",administrativeState)
                        FI
                       ....
@";
```

FIG. 5.15 - . *Un exemple de formalisation de comportement au niveau d'un objet géré*

Cette construction implique la prise en compte, d'une part des différents formulaires au sein d'un objet géré, et d'autre part du comportement hérité de la super-classe. Les différents formulaires de comportement qui entrent en compte dans le processus de collecte sont les suivants:

- **action** dans lequel une partie du comportement lié à une action est définie,
- **notification** qui définit le comportement général d'une notification, à savoir la condition de déclenchement et les effets de l'émission,
- **module** qui spécialise les comportements de ses composants (actions et notifications),
- **objet géré** qui fédère les comportements des modules et lie les règles à l'occurence des événements aux portes de communication.

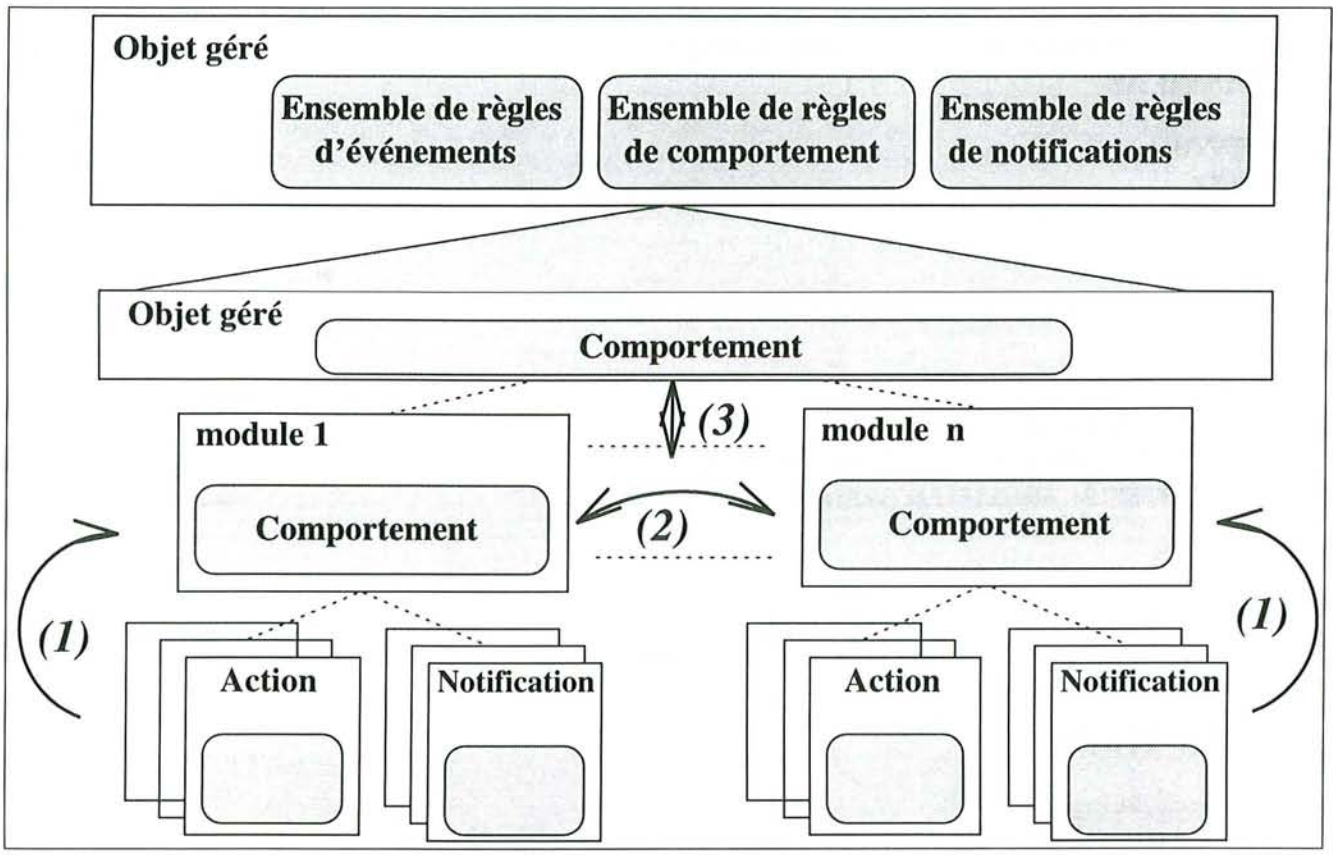

FIG. 5.16 - . *La collecte du comportement au sein d'un objet*

Pour permettre de construire le comportement d'un objet géré (c'est à dire le système de règles CRS associé), à partir des différents formulaires et super-classes, nous allons concevoir un algorithme qui va réaliser cette tâche en deux étapes distinctes.

La première étape va consister à construire, pour chaque objet géré et à partir de ses seuls formulaires (sans l'héritage), trois ensembles de règles comme le montre la figure 5.16. Le premier va contenir toutes les règles de l'objet qui décrivent des occurences d'événements. Le second va contenir un ensemble de règles qui décrivent les pré- et post-conditions de ces événements et le dernier le comportement associé aux notifications.

Dès que ces trois ensembles seront construits pour chaque objet géré, nous pourrons appliquer les concepts d'héritage tels que nous les avons énoncés dans le chapitre précédent et générer le système de règles pour l'objet géré considéré.

Dans un premier temps, nous présentons l'algorithme de construction des ensembles pour un objet donné et, par la suite, nous détaillerons la manière dont les concepts d'héritage permettent de construire, à partir de ces ensembles, le comportement CRS d'une sous-classe.

5.9.2 L'algorithme de collecte au sein d'un objet

Trois ensembles sont à contruire au sein d'un objet. Le premier que nous appelerons E comporte l'ensemble des règles spécifiées dans la clause **EVENTS** d'un objet géré. Le second et le troisième (B

et N) résulteront de la composition des règles présentes dans les formulaires d'action, de notification, de module et de la clause **RULES** de l'objet géré concerné. L'ensemble B comportera les règles relatives au comportement des actions et opérations, tandis que l'ensemble N comportera les définitions des comportements de notifications.

Afin de différencier les commentaires et justifications du texte décrivant le fonctionnement de l'algorithme, toutes les parties algorithmiques seront écrites en italique.

Construction de l'ensemble d'événements

L'ensemble E associé à un objet géré représente l'ensemble des règles de la clause **EVENTS**.

Appliqué sur l'exemple de la figure 5.15, on obtient pour E l'ensemble suivant:

$E_{ExtendedComputerSystem}$ =

```
{
  ( lock;
    COND: Management.lock();
  ),
  ( logout;
     COND:UsageStatePackage AND External.logout();
  ),
    ....    }
```

Cet ensemble E, élaboré pour chaque objet géré, va nous servir à construire le système de règles associé à un objet en appliquant l'algorithme d'héritage décrit plus loin dans ce chapitre.

Construction de l'ensemble des règles de comportement et de notification

La construction des ensembles B et N peut être divisée en trois étapes successives. La première étape consiste à intégrer les spécifications de comportement des formulaires d'actions et des notifications dans ceux des modules qui les intègrent (étape 1 de la figure 5.16). Ceci nous permet de construire les comportements de modules de manière indépendante les uns des autres.

La seconde étape consiste à composer les comportements des différents modules au sein de l'objet géré. La troisième étape va étendre les règles issues de la composition des deux premières étapes avec celles spécifiées dans la clause **RULES** du comportement formel de l'objet géré. Dans les lignes suivantes, nous allons détailler chacune de ces étapes.

Première étape

La première étape consiste pour chaque module d'un objet géré à intégrer dans son comportement, les comportements des actions et notifications qu'il instancie. Cette étape peut se décomposer en deux sous-tâches qui sont, d'une part la composition des règles relatives aux actions et opérations, et d'autre part le couplage des comportements de notifications. On obtiendra ainsi pour chaque module m de l'objet géré, deux ensembles B_m et N_m.

La première sous-tâche de cette étape construit l'ensemble B_m et fonctionne de la manière suivante:

pour chaque module de l'objet géré on va parcourir chaque règle présente dans son comportement formel. Pour chaque action référencée dans ce module, on va parcourir l'ensemble de ses règles et appliquer les opérations décrites ci-dessous. L'algorithme associé à la première étape est donc le suivant:

Pour tout module m, pour toute action a référencée dans le module m:

- *si la règle r_i du formulaire d'action n'a pas de correspondante dans le module (c'est à dire qu'il n'existe pas de règle dans le module qui a le même nom que r_i), alors cette règle est ajoutée sans modifications dans l'ensemble B_m.*
- *si par contre la règle r_i du formulaire d'action possède une règle r_j associée dans le comportement de module, alors la règle r_j du module est étendue de la manière suivante:*
  - *la pré-condition de la règle étendue est obtenue par couplage de la pré-condition de r_j et celle de r_i à l'aide du connecteur* **AND**,
  - *la post-condition de la règle étendue est obtenue par couplage de la post-condition de r_j et celle de r_i à l'aide du connecteur* **AND**.

  Le résultat de cette composition est ajouté à l'ensemble B_m.

Toute règle spécifiée dans le module, non spécifiée dans un formulaire d'action, est ajoutée sans modifications à B_m.

Cette construction appliquée à tous les modules m de l'objet géré permet de construire pour chacun d'eux un ensemble B_m de règles de comportement. La raison du couplage par le connecteur logique **AND** des pré- et des post-conditions d'actions définies dans le module et dans le formulaire d'action est la suivante: le comportement lié à une action décrit dans le formulaire d'action, définit les conditions générales nécessaires à l'invocation de celle-ci. La post-condition décrit les contraintes que doit respecter l'action dans le cadre général. Lorsqu'une action est placée dans un module, les conditions générales d'invocation sont renforcées par des conditions particulières liées aux attributs du module. Il en est de même pour la post-condition. Seul un couplage à l'aide du connecteur logique **AND** des pré-conditions (dans l'action et dans le module) et le même type de couplage entre les post-conditions permettent de garantir la contrainte que nous venons d'évoquer.

Appliqué à l'exemple des figures 5.13 et 5.11 qui définissent repectivement un comportement de module et un comportement d'action, on obtient l'ensemble $B_{BasicComputerSystemPackage}$ suivant:

$B_{BasicComputerSystemPackage}$ =

```
{ (activateSuccess;
     COND: operationalState = disabled
     EFFECTS: operationalState = enabled
              AND response = successful;
  ),
  (activateFailure;
     EFFECTS: operationalState = 'operationalState
              AND response = failureResponse;
  ),
  (disable;
     EFFECTS: operationalState = disabled;
  )
}
```

La seconde sous-tâche de cette première étape doit construire au sein de chaque module l'ensemble N_m qui regroupe les définitions de comportement des notifications recensées. Cette sous-tâche se déroule de manière analogue au traitement des actions. Le calcul est présenté ci-après.

Pour toutes les définitions d_i des formulaires de comportement des notifications n instanciées dans le module m:

- *si la définition d_i ne comprend pas de définition associée dans le module, alors elle est incluse dans l'ensemble N_m sans modifications.*
- *s'il existe une définition d_j dans le module, de même nom que la définition d_i, alors la définition d_j du module est étendue de la manière suivante:*
  - *la condition de déclenchement de la définition d_j du module est étendue par liaison de la condition de déclenchement du module et celle spécifiée dans la définition de notification (définition d_i), au travers de l'opérateur* **AND**. *Ceci entraîne un renforcement de la condition de déclenchement.*
  - *les effets du déclenchement de la définition d_j du module sont étendus par liaison des effets définis dans le comportement du module et ceux spécifiés dans la définition de notification (définition d_i) au travers de l'opérateur* **AND**. *Par de cette manipulation, les effets du déclenchement de la notification sont cumulés avec ceux du module au sein de ce dernier.*

Le résultat de cette composition est ajouté à l'ensemble N_m.

Comme pour la traitement des actions, toute définition présente dans le module et ne possédant pas de définition associée dans un formulaire de notification est ajoutée sans modifications à N_m.

La raison du choix de la manière de connecter les pré- et post-conditions est la même que celle donnée dans le cadre des actions.

Sur l'exemple de la figure 5.13, l'ensemble N ainsi obtenu ne comporte qu'un seul élément:

$N_{BasicComputerSystemPackage}$ =

```
{ (stateChange;
     TRIGGER: operationalState <> 'operationalState;
     ISSUANCE: IF (operationalState <> 'operationalState)
               THEN MEMBEROF(stateSet,"OperationalState ",operationalState)
               FI;
  )
}
```

Le traitement des variables locales et des prédicats utiles est le suivant: l'ensemble des variables locales du module est égal à l'union des variables locales définies au sein du module et de toutes celles définies dans les formulaires de comportement des actions et des notifications référencées dans le module. La démarche est identique pour les prédicats utiles.

Cette première étape nous permet d'intégrer les spécifications de comportement définies dans les formulaires d'action et de notification dans les modules et de calculer les ensembles associés (voir figure 5.17).

L'étape suivante *(2)* aura pour but de fédérer les comportements des différents modules au sein de deux ensembles temporaires B_{temp} et N_{temp} qui nous serviront à construire les ensembles finaux B et N pour l'objet géré concerné. Cette étape est décrite dans les lignes suivantes.

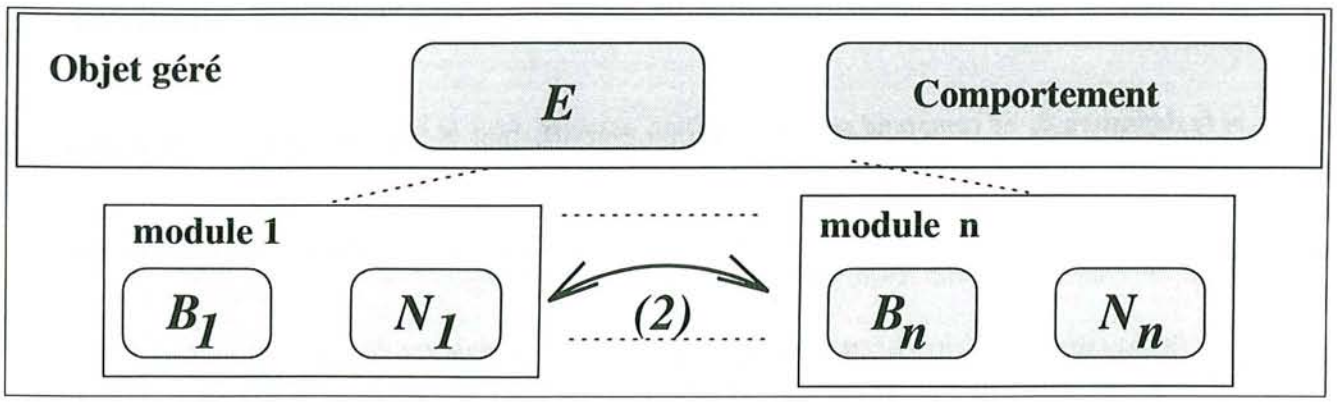

FIG. 5.17 - . *Résultat de la première étape de collecte*

Seconde étape

La seconde étape fédère les règles des différents modules pour construire les ensembles intermédiaires. Cette étape peut, comme la précédente, se décomposer en deux sous-tâches. La première traite les actions et opérations tandis que la seconde manipule les notifications. Elle se déroule comme décrit ci-dessous.

La première sous-tâche construit l'ensemble B_{temp} de la manière suivante:

Pour toute règle r issue de B_m de tous les modules m de l'objet géré:

- *si la règle r n'est présente que dans un seul module:*
  - *si le module est un module obligatoire, alors la règle r est ajoutée à B_{temp} sans modifications.*
  - *si le module est un module optionnel, la pré-condition* `pre` *devient:* `<package-label>AND pre`[3] *et le résultat est ajouté à B_{temp}.*
- *si la règle r est dispersée sur plusieurs modules, alors la règle résultante ajoutée à B_{temp} porte le même nom que r et:*
  - *si la règle est référencée dans au moins 1 module obligatoire alors sa pré-condition est égale à la conjonction des pré-conditions de celle-ci dans les modules obligatoires couplée par un* **AND** *logique avec la conjonction des pré-conditions modifiées de celle-ci dans les modules optionnels dans lesquels elle est référencée. Une pré-condition* `pre` *modifiée est du type* `si <package-label>alors pre sinon vrai fsi`.

    Sa post-condition est égale à la conjonction des post-conditions de celle-ci dans les modules obligatoires couplée par un **AND** *logique avec la conjonction des post-conditions conditionnées par la présence du module (c'est à dire:* `post` $\Rightarrow$ `si <package-label>alors post sinon vrai fsi`*) issues des modules optionnels.*
  - *si la règle n'est référencée que dans des modules optionnels alors sa pré-condition est égale à la conjonction des pré-conditions modifiées de r dans les modules couplée par un* **AND** *logique à un prédicat du type* `<package-label>OR <package-label>OR ...` *où une étiquette de paquetage représente un module optionnel dans lequel la règle est référencée. Tout comme*

3. . L'étiquette de paquetage représente l'identificateur du module optionnel. Sa présence dans la pré-condition indique que le comportement est conditionné par la présence du module.

pour le précédent cas, une pré-condition `pre` *étendue est égale à* `si <package-label>alors pre sinon vrai fsi`.
La post-condition de la règle ajoutée à B_{temp} *est égale à la conjonction de toutes les post-conditions modifiées de r dans les différents modules optionnels où r est présente. Ici également la modification d'une post-condition* `post` *transforme celle-ci en* `si <package-label>alors post sinon vrai fsi`.

Cette construction signifie que les comportements issus de différents modules et liés à une même action, s'additionnent. C'est-à-dire que les pré-conditions tout comme les post-conditions se conjugent[4].

Le fait de différencier la manière dont sont traitées les règles issues de modules optionnels et de modules obligatoires reflète le fait qu'en l'abscence du module optionnel, le comportement qui y est décrit ne doit pas être pris en compte pour l'évaluation de la règle. Le fait de conditionner les pré- et post-conditions issues de modules optionnels par un test sur la présence du module permet de prendre en compte cet aspect de la répartition et de l'agrégation du comportement.

L'algorithme pour la seconde sous-tâche est équivalent à celui de la première et permet de calculer l'ensemble N_{temp}. Il se décline comme suit:

Pour toute règle n issue de N_m *de tous les modules m de l'objet géré:*

- *si la règle n n'est présente que dans un seul module alors:*
  - *si ce module est obligatoire, elle est ajoutée à* N_{temp} *sans modifications.*
  - *si ce module est optionnel, la pré-condition* `pre` *de la règle est couplée avec l'identificateur du module optionnel dans lequel elle est définie (*`pre` ⇒ `<package-label>AND pre`*).*
- *si la règle n est dispersée sur plusieurs modules, alors la règle résultante, ajoutée à* N_{temp}*, porte le même nom que n et:*
  - *si la règle est référencée dans au moins 1 module obligatoire, alors sa pré-condition est égale à la conjonction des pré-conditions de celle-ci dans les modules obligatoires couplée par un* **AND** *logique à la conjonction des pré-conditions modifiées de celle-ci dans les modules optionnels dans lesquels elle est référencée. Une pré-condition* `pre` *modifiée est du type* `si <package-label>alors pre sinon vrai fsi`.
    Sa post-condition est égale à la conjonction des post-conditions de celle-ci dans les modules obligatoires couplée par un **AND** *logique aux post-conditions conditionnées par la présence du module (c'est à dire* `post` ⇒ `si <package-label>alors post sinon vrai fsi`*) issues des modules optionnels.*
  - *si la règle n'est référencée que dans des modules optionnels alors sa pré-condition est égale à la conjonction des pré-conditions modifiées de n dans les modules étendue par un prédicat du type* `<package-label>OR <package-label>OR` ... *où les étiquettes de paquetage représentent les modules optionnels dans lesquels la règle est référencée. Tout comme pour le cas précédent une pré-condition* `pre` *étendue est égale à* `si <package-label>alors pre sinon vrai fsi`.
    La post-condition de la règle ajoutée à B_{temp} *est égale à la conjonction de toutes les post-conditions modifiées de n dans les différents modules optionnels où n est présente. Ici également la modification d'une post-condition* `post` *transforme celle-ci en* `si <package-label>alors post sinon vrai fsi`.

4. . On peut noter à ce niveau que ce type de couplage est différent du couplage lié à l'héritage et que le fait de définir la manière dont se couplent les comportements des différents modules donne une sémantique à ces derniers qui ne peuvent donc plus être considérés comme équivalents à une extension dans une sous-classe.

Cela signifie que le comportement lié à une notification référencée dans plusieurs module se construit par renforcement de sa pré- et de sa post-condition. On remarque qu'une notification est traitée de la même manière qu'une action au niveau de son comportement et de la prise en compte de l'abscence éventuelle de module optionnel qui invalide éventuellement une partie du comportement de l'objet gérés.

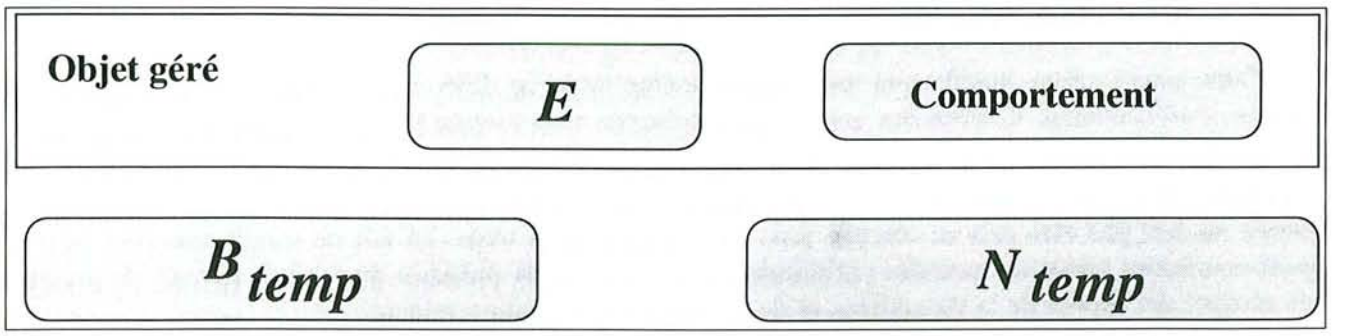

FIG. 5.18 - . *Résultat de la seconde étape de collecte*

La figure 5.18 illustre la situation du comportement obtenu après l'application de la seconde étape de l'algorithme. Pour construire les ensembles B et N de l'objet géré, il nous faut maintenant conjuger les ensembles B_{temp} et N_{temp} avec le comportement décrit dans le formulaire associé à l'objet géré. C'est l'objet de la troisième étape.

Le calcul de l'ensemble des variables locales et celui des prédicats utiles est proche du calcul défini sur ces derniers dans la première étape. Il s'agit ici de construire dans B_{temp} et N_{temp} un ensemble de variables locales et un ensemble de prédicats utiles à partir des définitions présentes dans les différents modules. Pour cela, il suffit de faire, pour obtenir chaque ensemble, l'union des définitions de tous les modules.

Troisième étape

La troisième étape a pour but de construire les ensembles B et N associés à un objet géré en couplant les ensembles B_{temp} aux règles spécifiées dans la clause **RULES** du comportement de l'objet géré et en couplant l'ensemble N_{temp} aux spécifications de la clause **NOTIFICATIONS** du comportement de l'objet géré.

L'algorithme ci-dessous permet de construire l'ensemble B.

Pour toute règle r_i définie dans la section **RULES** *du comportement de l'objet géré:*

- *s'il existe une règle r_j dans l'ensemble B_{temp} telle que r_i et r_j ont le même nom, alors on ajoute à B une règle r_k telle que:*
  - *le nom de r_k est celui de r_i,*
  - *la pré-condition de r_k est égale à la conjonction de celle de r_i et celle de r_j.*
  - *la post-condition de r_k est égale à la conjonction de celle de $r_i$et celle de r_j.*
- *les règles de la clause* **RULES** *non présentes dans B_{temp} sont ajoutées à B sans modifications.*

Les règles de l'ensemble B_{temp} qui ne sont pas spécifiées dans la clause **RULES** *de l'objet géré sont ajoutées sans modifications à B.*

Ceci signifie que le comportement général décrit par une règle de la partie **RULES** de l'objet géré, qui affine une règle d'un module, doit tenir compte du comportement décrit dans le module. C'est pourquoi les pré- et post-conditions de cette règle sont conjugées.

L'ensemble N est obtenu de manière analogue à B. L'algorithme est décrit ci-dessous.

Pour toute définition d_i de la clause **NOTIFICATIONS** *du comportement de l'objet géré:*

- *s'il existe une définition d_j dans N_{temp} telle que d_i et d_j ont le même nom, alors on ajoute à N une définition d_k telle que:*
  - *le nom de d_k est égal à celui de d_j,*
  - *la partie* **TRIGGER** *de d_k est obtenue par couplage des deux pré-conditions de d_i et de d_j par le connecteur logique* **AND**.
  - *la partie* **ISSUANCE** *de d_k est égale à la conjonction de la partie* **ISSUANCE** *de d_i et celle de d_j.*
- *si la définition d_i n'a pas de définition équivalente dans N_{temp} elle est ajoutée dans N sans autres modifications.*

Toute définition de N_{temp} non spécialisée dans la clause **RULES** *est ajoutée à N sans modifications.*

On remarque ici que les pré-conditions, respectivement post-conditions, de toutes les notifications spécialisées dans la clause **NOTIFICATIONS** du comportement de l'objet géré sont ainsi renforcées. La raison de ce choix de couplage est identique à celle invoquée dans le cas du couplage des modules et dans le cas de l'intégration des notifications dans les modules, à savoir la définition de la pré-condition en termes de conditions nécessaires d'où le couplage **AND**.

Cet algorithme nous permet de construire pour tout objet géré les trois ensembles E, B et N. Pour obtenir le comportement complet d'un objet et pouvoir en générer un système de règles CRS, il faut maintenant considérer la construction par héritage. C'est l'objet de la prochaine section.

5.9.3 La construction par héritage

Construire le comportement d'un objet géré par héritage va pouvoir se faire à partir des trois ensembles qui ont été établis lors des précédentes étapes de la construction comme le montre la figure 5.19.

Cette construction va se faire en trois étapes. La première consiste à construire les trois ensembles pour une sous-classe à partir des ensembles propres à la sous-classe et ceux de la super-classe dont elle hérite. La seconde étape permet de lier le comportement décrit dans les ensemble B_{res} et N_{res} aux occurences d'événements spécifiés dans l'ensemble E_{res}. La dernière étape consiste à coupler les définitions de notifications aux règles relatives au comportement des actions et opérations.

Première étape

Cette étape étend les ensembles E, B et N d'une sous-classe à partir de ceux définis dans la super-classe. Pour des raisons de simplicité, nous appellerons les ensembles résultants E_{res}, B_{res} et N_{res}.

L'ensemble des événements E_{res} est égal à l'union de l'ensemble des événements de la sous-classe $E_{subclass}$ et de l'ensemble des événements de la super-classe E_{class}.

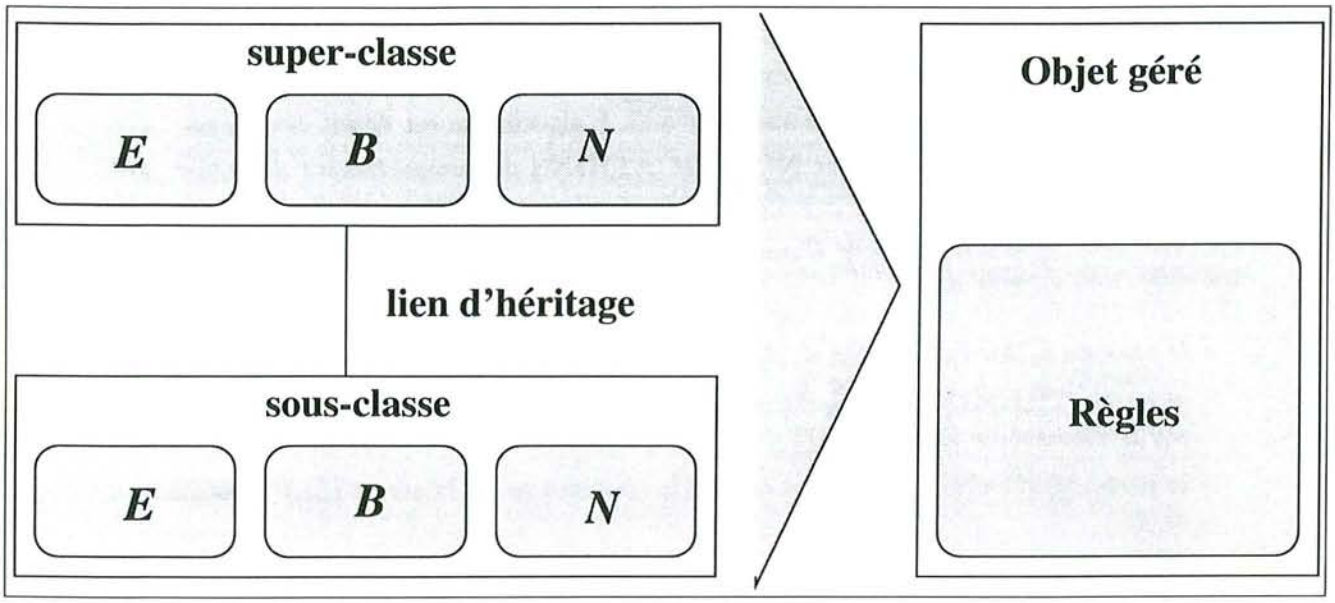

FIG. 5.19 - . *La construction par héritage*

L'ensemble des règles de comportement de l'objet résultant d'une composition par héritage B_{res} est défini d'après les règles d'héritage du chapitre précédent. C'est à dire:

- *toute règle présente dans* B_{class} *et non spécifiée dans* $B_{subclass}$ *est ajoutée sans modifications à* B_{res},
- *toute règle présente dans* $B_{subclass}$ *et non spécifiée dans* B_{class} *est ajoutée sans modifications à* B_{res},
- *toute règle spécifiée dans* $B_{subclass}$ *et* B_{class} *se voit ajoutée à* B_{res} *avec une pré-condition égale à la disjonction de sa pré-condition dans la sous-classe et celle dans la super-classe (couplage* **OR***) et une post-condition égale à la conjonction des post-conditions de la règle dans la sous-classe et dans la super-classe (couplage* **AND***).*

Ce calcul se fait de manière récursive de la sous-classe jusqu'à la racine de l'arbre d'héritage.

Le calcul de l'ensemble N_{res} se fait de manière analogue à celui de B_{res}. Il suffit pour cela de changer le calcul de la pré-condition par un calcul identique sur les conditions de déclenchement, et le calcul sur les post-conditions par un calcul identique sur les effets de l'émission.

Cette première étape permet de calculer les trois ensembles E_{res}, B_{res} et N_{res} par héritage pour un objet géré donné. Il faut maintenant combiner ces trois ensembles pour obtenir des règles homogènes. C'est le but des deux prochaines étapes.

Seconde étape

La seconde étape consiste à intégrer les occurences d'événements dans les règles du comportement liées aux actions, opérations et notifications.

Cette étape est assez simple. Elle consiste à intégrer les spécifications contenues dans E_{res} dans les ensembles B_{res} et N_{res}. Elle se déroule comme décrit ci-après.

Pour toute règle r_i spécifiée dans l'ensemble des événements, toutes les règles de l'ensemble B_{res} de même nom voient leurs pré-conditions étendues par couplage logique **AND** *avec l'occurence spécifiée dans la pré-condition de r_i et leurs post-conditions étendues par couplage logique* **AND** *avec l'occurence spécifiée dans la post-condition de r_i.*

Pour toute définition d_i spécifiée dans l'ensemble des événements, toutes les définitions de l'ensemble N_{res} de même nom voient leurs post-conditions (clause **ISSUANCE***) étendues par couplage logique* **AND** *avec l'occurence spécifiée dans la post-condition de d_i.*

Les événements ayant été liés aux règles d'actions, d'opérations et de notifications, il nous faut maintenant lier les notifications aux autres règles. C'est l'objet du paragraphe suivant.

Dernière étape

La dernière étape a pour but de coupler les règles relatives au comportement des actions et opérations (ensemble B_{res} étendu) avec les règles relatives aux notifications (ensemble N_{res} étendu). Dans ce cadre, deux stratégies sont envisageables.

La première consiste à insérer dans toute post-condition de règle issue de B_{res}, un test pour chaque notification définie. En fonction du test, les effets du déclenchement sont validés ou ne le sont pas. Cette insertion est définie dans la figure 5.20. La stratégie est simple à mettre en oeuvre.

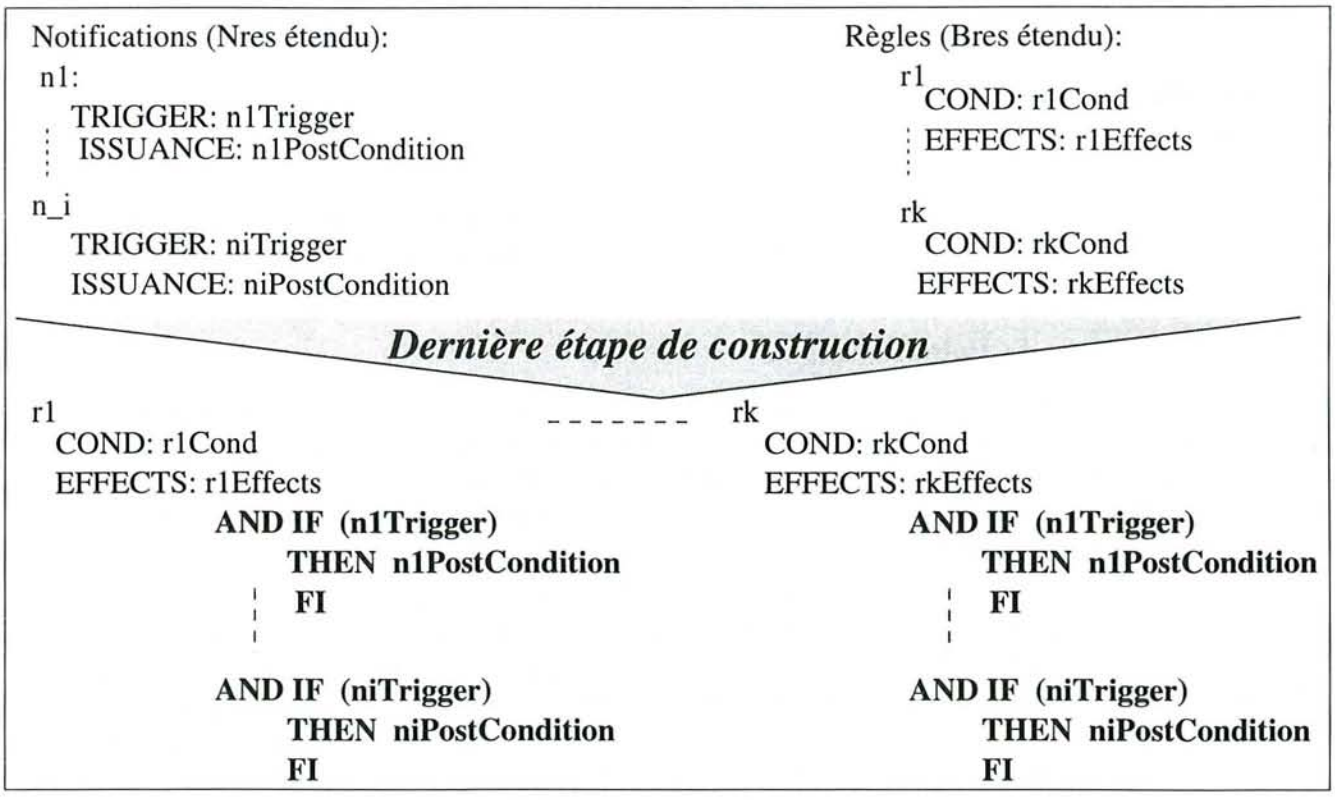

FIG. 5.20 - . *La dernière étape de la construction des règles*

La seconde stratégie consiste à n'insérer les post-conditions des émissions de notifications que dans les règles où la condition de déclenchement est ou peut être vérifiée. Ceci se fait en analysant toutes les

post-conditions des règles. Dès qu'un élément de la pré-condition de la notification y apparait, alors le test de la notification est couplé à cette règle. Cette stratégie est plus subtile que la précédente mais également plus complexe à mettre en oeuvre.

Nous avons, pour des raisons de simplicité de mise en oeuvre, retenu la première stratégie pour l'intégration du comportement relatif aux notifications dans celui des autres composants de l'objet géré.

L'application de cette dernière étape permet d'obtenir pour un objet géré donné, une description complète des règles qui régissent son comportement[5].

Génération du système de règles associé à l'objet géré

La construction du système de règles associé à un objet géré se fait à partir de la description obtenue lors de la dernière étape. Le système est composé de la manière suivante:

- le système est paramétré par les portes de communication de l'objet géré et un ensemble de variables booléennes qui décrivent le support des modules optionnels,
- les variables d'états du système sont obtenues par union de l'ensemble des attributs de l'objet géré et des étiquettes de modules optionnels,
- les variables intermédiaires du système sont identiques à celles de l'objet géré,
- les prédicats utiles du système sont ceux définis dans les différents formulaires de l'objet géré considéré,
- les règles du système sont celles obtenues après la dernière étape de construction du comportement par héritage.

Un tel système va nous permettre d'exploiter la description du comportement des objets gérés dans l'environnement de développement que nous avons conçu et réalisé autour de LOBSTERS.

5.9.4 Validation de l'algorithme

L'algorithme que nous avons présenté dans cette section a l'avantage de forcer le concepteur d'objets gérés à réfléchir à la localisation de ses informations de comportement et de ne pas tout concentrer dans une partie de la spécification. L'algorithme exploite tous les formulaires de comportement et permet, à partir des différentes spécifications, de construire une description homogène, en termes de règles, du comportement d'un objet géré.

Cet algorithme a été conçu à partir d'une étude détaillée de spécifications d'objets gérés existantes. La définition de la manière dont se connectent les différents comportements pour former celui de l'objet géré dans lequel ils sont référencés, nous a permis de donner une sémantique aux différents éléments qui composent l'objet géré et qui référencent des comportements. Dans ce cas, nous pensons surtout au concept de module ou paquetage qui ne fait l'objet d'aucune définition précise dans les normes et standards. Le comportement formel conjugé à l'algorithme de collecte donnent au module une sémantique d'ensemble regroupant des attributs, actions et notifications dont le comportement doit être décrit de manière consistante et indépendante des autres modules référencés dans l'objet. L'algorithme de collecte montre de plus, que les comportements des différents modules peuvent se conjuger, mais que

5. . Les résultats intermédiares de l'algorithme sont donnés dans l'exemple de l'appendice 2 de la thèse

cette conjugaison (couplage **AND** des pré- et post-conditions) n'est pas identique à la construction par héritage.

L'étude que nous avons entreprise avec cet algorithme a montré que les contraintes qu'il impliquait sur la localisation du comportement au sein d'un objet géré étaient en parfaite harmonie avec les spécifications existantes ainsi que celles que nous avons développées. La collecte n'entraine aucune contradiction dans le comportement construit. La construction par héritage est elle, conforme à la définition formelle de l'héritage, et donc valide de fait.

La validation de l'algorithme pour sa partie collecte au niveau d'un objet géré, en termes d'adéquation avec la vue d'un spécifieur d'objets gérés ne peut cependant se faire que par une exploitation poussée du formalisme par d'autres spécifieurs et pourra seule confirmer ou infirmer ce choix de répartition.

5.10 Résumé

Dans ce chapitre, nous avons présenté l'intégration du formalisme basé sur les concepts CRS dans le langage GDMO. Pour ce faire, nous avons présenté les extensions proposées à la norme au travers des différents formulaires de comportement.

La présentation de ces extensions nous a permis de mieux comprendre l'intérêt d'un formalisme unique et l'avantage de la compatibilité avec GDMO. Compatibilité pseudo-parfaite en réalité car nous avons dû introduire dans le formalisme deux nouveaux formulaires qui sont le formulaire d'interface et le formulaire de comportement d'objet géré, non présents dans la norme.

Nous avons finalement décrit l'algorithme de collecte du comportement qui, à partir des différents formulaires de comportement, permet de construire, d'une part un comportement homogène, et d'autre part le système de règles CRS associé. Nous avons également vu que l'algorithme imposait fortement la manière dont le formalisme devait être utilisé et que cela pouvait être considéré comme un avantage à ce niveau, mais pouvait également se montrer désavantageux à terme.

La mise à disposition d'un formalisme étendu pour la description du comportement des objets gérés dans GDMO ne couvre pas l'ensemble des exigences que nous nous étions fixées au début de cette thèse. Nous avons en effet, émis le besoin de mettre en place des mécanismes qui sont capables d'exploiter cette information supplémentaire au sein d'un environnement de développement homogène. Ces mécanismes font l'objet des travaux développés dans la troisième partie de cette thèse.

PARTIE III

L'utilisation des extensions dans le développement d'agents de gestion

«Il faut créer ce que l'on cherche.»

René Magritte.

Cette partie est consacrée aux environnements de développement d'applications de gestion et plus particulièrement aux applications qui peuvent tirer profit des extensions que nous avons apportées au formalisme GDMO dans la seconde partie de ce document.
Après une brève introduction aux environnements existants pour le développement d'applications de gestion, nous présenterons l'environnement MODE que nous avons développé. Dans le cadre de ce dernier, nous présenterons plus particulièrement l'environnement de simulation que nous utilisons pour valider des scénarios de bases d'informations de gestion.

Chapitres

1. **Un environnement de développement homogène**
2. **Une approche simulative pour la validation d'interfaces de gestion**

1
Un environnement de développement homogène

Dans le chapitre précédent, nous avons présenté le formalisme LOBSTERS qui repose sur une extension de GDMO avec des mécanismes de description du comportement issus de la FDT CRS. Avec ce formalisme, nous avons atteint le premier objectif de la thèse: mettre à disposition un langage compatible à GDMO qui supporte la formalisation du comportement des objets gérés. Le second objectif que nous nous étions fixés dès le début, était le besoin de mettre à disposition des concepteurs d'objets gérés, un environnement logiciel qui facilite le développement et tire profit des extensions apportées à GDMO. Ce chapitre prévoit l'étude du processus de développement des objets gérés et des bases d'informations de gestion. Dans ce cadre il présente les environnements existants et introduit l'environnement que nous avons conçu autour de LOBSTERS.

1.1 Le processus de développement d'un agent de gestion

Mettre en place un environnement de développement de bases d'informations nécessite une étude préalable du processus de développement. Cela implique l'identification de toutes les étapes du développement, mais aussi la définition des liens entre ces différentes étapes en termes de validation et de traitement de l'information. Nous allons tout d'abord définir le processus de développement, puis à partir de cette définition, extraire un certain nombre d'exigences envers tout environnement de développement.

1.1.1 Le processus

Il existe dans la littérature de nombreuses définitions de processus de développement basés sur une approche formelle. Dans le cadre de LOBSTERS, nous avons retenu le processus défini pour CRS dans [Schneider 92, Velthuys 92a], car il comporte les principales fonctionnalités que l'on trouve dans la plupart des définitions.

Le processus de développement d'une base d'informations est relativement proche de celui fréquemment retenu dans le cadre de l'ingénierie des protocoles. En effet, comme il est également construit sur l'utilisation d'une méthode formelle, on y retrouve les différentes phases du passage de l'énoncé des besoins à une première spécification formelle (formalisation), les étapes de spécialisation et d'affinages successifs, et puis finalement le passage de la spécification à l'implantation (étape de réalisation).

Le schéma de la figure 1.1 illustre le processus de développement de bases d'informations de gestion. La principale différence avec le processus de développement de protocoles repose sur la réutilisation de spécifications existantes enregistrées dans l'arbre d'héritage des objets gérés. Après une définition plus

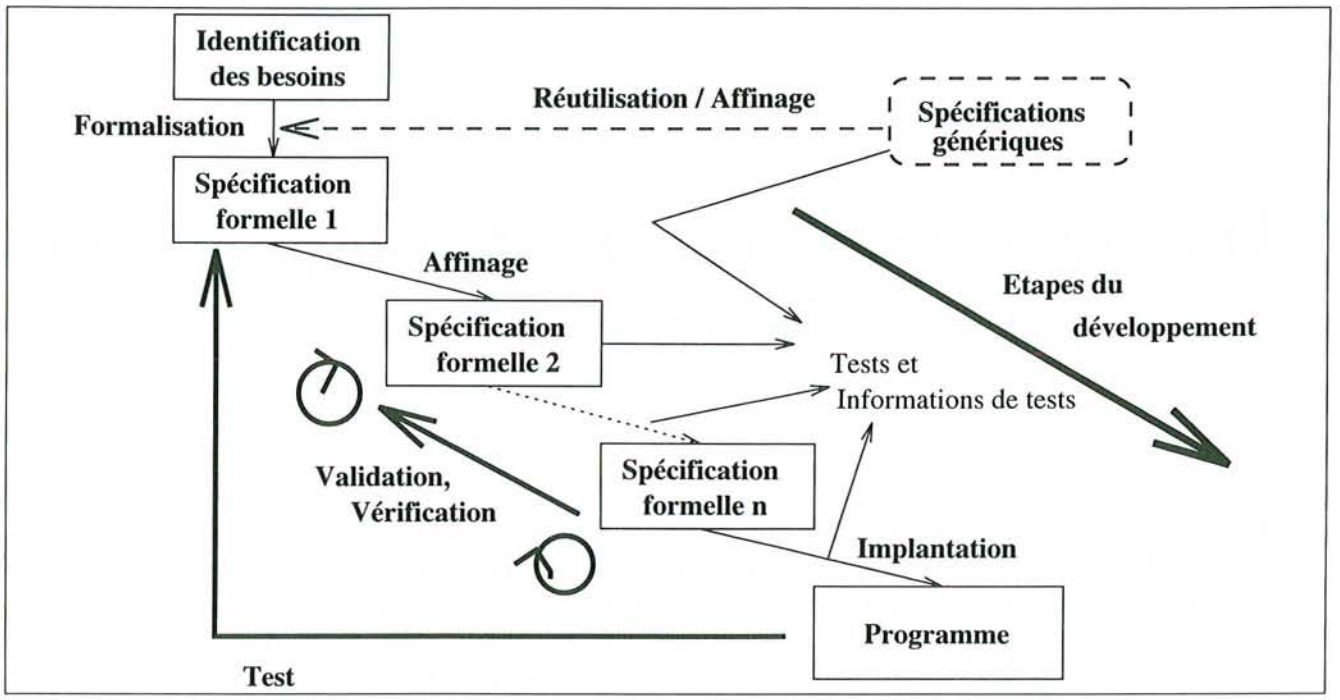

FIG. 1.1 - . *Le processus de développement d'agent de gestion*

détaillée de ces différentes étapes, nous allons définir les besoins réels en termes de support logiciel du processus de développement.

1.1.2 Les étapes de développement

Dans les lignes suivantes, nous allons définir de manière précise les actions attachées à chaque étape du développement d'une base d'informations de gestion. Ces actions formeront la base des tâches à supporter par un outil logiciel.

Analyse et expression des besoins

La première étape de tout développement d'une base d'informations de gestion est celle de l'analyse et de l'expression informelle des besoins. Cette étape consiste à étudier le système pour lequel une base d'informations de gestion doit être développée. Ceci revient à extraire, de manière concertée avec l'utilisateur, de la documentation du système, les caractéristiques intéressantes pour la gestion. Ces caractéristiques forment alors le premier modèle informel du système à gérer.

Cette étape doit se faire de manière concertée avec les personnes qui vont gérer le système afin de bien capter leurs besoins. Elle est basée sur une approche entièrement informelle et ne requiert pas d'outils logiciels spécifiques dans le processus.

Formalisation et réutilisation

L'étape qui succède à la définition des besoins de l'utilisateur est celle de formalisation du modèle informel. Cette formalisation consiste à construire la définition des objets gérés en GDMO à partir du

modèle informel défini lors de la première étape. Cette étape comporte à la fois un processus de conception (définition de nouvelles classes d'objets gérés) et une activité de réutilisation (reprise de classes d'objets gérés déja définis dans des normes ou dans d'autres bases d'informations de gestion). Ces deux tâches de formalisation et de réutilisation peuvent être combinées lors de la réutilisation et de la spécialisation d'une classe existante.

La définition formelle du modèle ainsi obtenue devra servir de base à la négociation entre le concepteur de la base d'informations et le futur utilisateur (le gestionnaire du système). La négociation pourra éventuellement amener les partenaires à revenir sur la première étape de modélisation.

Cette étape comporte deux sous-tâches qui peuvent être facilitées par l'utilisation d'outils logiciels. Ces tâches sont, d'une part le développement de spécifications GDMO, et d'autre part le choix de classes existantes à réutiliser. Les outils logiciels intéressants dans ce cadre sont pour la première étape des analyseurs syntaxiques et sémantique statique des spécifications GDMO et ASN.1, et pour la seconde, des outils d'aide à la recherche et à la sélection de classes existantes. Ces derniers ont pour but de permettre de construire à partir de spécifications isolées de classes d'objets gérés, des architectures de MIBs cohérentes[1].

Spécialisation

L'étape précédente du processus de développement a permis d'établir une première version formelle du modèle du système. Or celle-ci ne comporte qu'une description de l'interface de gestion de la base. Le but de l'étape de spécialisation (étape itérée) est donc de partir de sa base et, dans un environnement formel, de la spécialiser pour l'amener vers une implantation. Cette spécialisation comporte un certain nombre de tâches qui sont pour chaque extension apportée à la spécification formelle, une vérification de la correction de la syntaxe et sémantique statique du résultat, ainsi qu'une validation de la nouvelle spécification par rapport à la précédente.

Le travail de vérification (preuve de propriétés de la spécification), tel qu'il est défini dans cette approche, est considéré comme une sous-tâche de l'activité de validation [Velthuys 92a] qui englobe l'ensemble des tâches qui permettent de vérifier que la spécification est conforme aux besoins exprimés.

Différents outils logiciels peuvent faciliter l'étape de spécialisation. Ils sont:

- des analyseurs de la syntaxe et de la sémantique statique de GDMO et d'ASN.1 pour les extensions apportées aux spécifications d'objets gérés.
- des outils de vérification et de validation des spécifications formelles à chaque étape de la spécialisation.
- des outils de génération d'informations de tests qui prennent en compte les décisions prises à chaque étape de la spécialisation.
- des outils de génération de tests qui permettent de générer des tests pour l'implantation à partir de la dernière version de la spécification formelle et qui prennent en compte les informations de tests fournies à chaque étape du processus de développement.

1. . On distingue à ce niveau la cohérence d'une spécification isolée d'un objet géré et la cohérence d'une spécification de MIB. La première (objet géré pris en isolation) implique l'existence des spécifications de toutes les parties de l'objet géré (paquetages, attributs, types ASN.1 correspondants, ...). Le second type de cohérence concerne une MIB complète, c'est à dire l'organisation d'une MIB par rapport aux spécifications de ses objets et des corrélations de noms qui les lient. Dans ce cas il faut bien sûr que toutes les spécifications des objets gérés impliqués soient complètes, mais également que pour chaque objet géré impliqué, il existe au moins une corrélation de noms et un objet géré englobant qui permettent d'instancier un objet de cette classe. Il existe le même type de contraintes de cohérence pour la prise en compte des relations dans une MIB.

Dans le cadre des outils de validation que nous mettons en oeuvre dans notre environnement, nous désirons valider des propriétés particulières sur un système de gestion et sur des MIB. Ces propriétés sont:

- sur les services de gestion:
  - complétude: les fonctionnalités attendues du service sont-elles réalisables par le service?
  - minimalité: toutes les primitives et opérations définies pour accéder et manipuler une MIB spécifique sont-elles utilisées dans l'activité de gestion?
  - correction: les opérations sont-elles complètement décrites et comportent-elles tous les paramètres nécessaires?
- sur les objets gérés pris en isolation:
  - les opérations sont-elles répercutées sur les attributs?
  - les réponses aux invocations sont-elles correctes?
  - les opérations interdites sont elles bien spécifiées?
  - toutes les notifications sont-elles émises?
- sur les MIBs:
  - les opérations définies sur un objet ont-elles des répercussions (attendues ou non) sur d'autres instances?
  - les dépendances entre objets gérés sont-elles vérifiées dans une exécution?
  - les interactions entre instances d'objets gérés sont-elles toutes correctes?

Dans le processus de développement et dans les outils proposés, nous chercherons à aider le concepteur d'une MIB avec le formalisme LOBSTERS à répondre à ces questions.

Etape de réalisation

L'étape de réalisation part d'une spécification formelle complète et assez détaillée pour construire une implantation de la base d'informations de gestion. Les outils existants généralement utilisés pour la réalisation de cette étape sont des compilateurs qui fournissent des fonctions de génération de code vers un environnement cible donné.

Dans le cas de notre approche, cette étape est limitée, car nous verrons que l'ensemble du code d'une implantation ne peut être généré automatiquement dans l'outil actuel[2]. Ceci implique donc la nécessité de considérer une étape supplémentaire qui est celle du test.

De plus, le passage d'une description formelle GDMO vers une implantation comporte, tout comme toute étape de spécialisation, un certain nombre de choix et de prises de décisions qui vont influencer les tests éventuels à faire sur l'implantation. En effet, lorsqu'un développeur de base d'informations de gestion va implanter des objets sur un système, il va prendre un certain nombre de décisions en ce qui concerne les modules optionnels supportés (support ou non), faire un certain nombre d'arbitrages sur des faits qui ne sont pas explicites dans les normes et être guidé par des impératifs du système cible.

2. . Il n'existe d'ailleurs à ce jour, aucun compilateur GDMO capable de générer tout le code pour une MIB. Ceci est bien sûr dû à l'abscence dans la norme de formalismes pour la description du comportement

Pour permettre de tester une implantation pour laquelle de telles décisions ont été prises, il faut dans le processus de développement, c'est à dire au moment où le développeur va faire ses choix, pouvoir enregistrer ces choix sous une forme normalisée afin d'exploiter cette information dans la phase de test. Nous appelons cette information de l'information de tests car elle permet de sélectionner les tests à effectuer sur l'implantation. Il existe une représentation normalisée de cette information. Celle-ci peut être intégrée dans le processus de développement comme nous le verrons plus tard.

Test de l'implantation

Le test de l'implantation du système a pour but de valider l'implantation vis-à-vis des besoins exprimés lors de la première phase du développement. Cette dernière est nécessaire car la spécification formelle ne prend pas en compte l'ensemble des facteurs de l'environnement d'accueil du code généré. Cette étape est d'autant plus nécessaire que la totalité du code d'une implantation est rarement générée automatiquement à partir de la spécification formelle.

Les outils qui tournent autour du test de l'implantation sont principalement ceux de génération, d'interprétation et de déroulement de test dans un environnement donné. Ils doivent être capables de prendre en entrée des séquences de test, de les exécuter vis-à-vis de l'agent de gestion sur sa base d'informations de gestion, et doivent finalement fournir des verdicts sur le déroulement des tests. La principale contrainte placée sur ces outils se situe au niveau du formalisme de description des tests fournis en entrée. En effet, il est impératif de se baser sur la notation normalisée TTCN.

1.2 Les exigences envers un environnement de développement

D'après le processus de développement présenté dans les pages précedentes, nous pouvons exprimer un certain nombre d'exigences envers un environnement de développement de bases d'informations de gestion, exigences exprimées en termes de besoins logiciels dont certaines sont données dans [Lecerf 93].

1.2.1 Les outils

Pour faciliter le développement de bases d'informations de gestion, il faut des outils qui fournissent un support pour les tâches suivantes:

- formalisation et spécification: analyse syntaxique et sémantique de nouvelles spécifications mais également gestion de spécifications existantes (bibliothèques d'objet gérés).
- élaboration de tests et recensement d'informations de tests: génération automatique ou semi-automatique de tests et support de génération d'informations de tests sur la base des choix de spécialisations opérés lors du développement.
- génération de code dans un environnement cible.

Un autre aspect important des besoins logiciels concerne la structure d'accueil d'une implantation de bases d'informations de gestion, car celle-ci n'est opérationnelle que si elle est couplée à un agent et que celui-ci communique au travers de CMIP. Il faut donc également que l'environnement d'accueil fournisse une structure assez souple pour permettre l'intégration de tout type de bases d'informations de gestion dans son système. La considération de cette exigence va fortement influencer les deux dernières étapes du développement que sont la réalisation et le test.

1.2.2 Les formalismes

Pour mettre à disposition des développeurs de bases d'informations de gestion les outils signalés dans les pages précédentes, ceux-ci doivent s'appuyer sur une technique de description formelle donnée. Dans le cadre de cette thèse, la FDT retenue sera le langage LOBSTERS. Il faut donc un support pour ASN.1 ainsi que pour les extensions que nous avons apportées à GDMO.

Dans le but de sélectionner un environnement de développement de bases d'informations de gestion, nous allons regarder les environnements existant à ce jour et les confronter aux exigences précédemment émises.

1.3 Les environnements existants

La gestion des réseaux hétérogènes par une approche OSI est une activité en plein essor et fort prometteuse. Aussi, les principaux constructeurs proposent déjà des environnements de développement de bases d'informations de gestion qu'ils intègrent dans leur plate-forme de gestion OSI. Quelques publications à ce jour présentent également un environnement de développement d'agents de gestion et de bases d'informations de gestion ([Dossogne 93, Desai 93, Pfeiler 93]).

L'étude de ces différents environnements montre qu'ils comportent en général les mêmes fonctionnalités et ont pour la plupart les mêmes limites. Dans les lignes suivantes, nous passons en revue ces outils et leurs limites.

1.3.1 Les fonctions supportées par les environnements existants

Certains environnements de développement d'agents de gestion existants proposent trois outils de base qui sont:

- un analyseur syntaxique GDMO et ASN.1,
- un outil de gestion de bibliothèques de spécifications GDMO et,
- un générateur de code pour GDMO et ASN.1.

Le premier outil permet à tout développeur de vérifier la syntaxe et la sémantique statique de ses définitions GDMO et ASN.1. Le second outil permet de manipuler des définitions GDMO, de gèrer des bibliothèques d'objets, de se déplacer dans ces bibliothèques, d'y ajouter de nouveaux objets ou d'y détruire des objets qui ne sont plus nécessaires.

Le dernier outil est un générateur de code qui permet de générer, d'une part le code relatif aux définitions ASN.1, et d'autre part un squelette pour les spécifications d'objets gérés en GDMO. Les structures cibles de la génération de code sont bien souvent des interfaces standardisées telles que XOM/XMP[3] ou CORBA/IDL[4].

Cependant, ces outils ne sont pas assez puissants pour permettre un support logiciel sur l'ensemble du processus de développement. En effet, ils ne supportent en général que deux étapes du processus qui sont l'élaboration de description GDMO et la génération (partielle) de code.

3. . X.Open Object Management/X.Open Management Protocol: Interfaces normalisées par X.Open

4. . Common Object Request Broker Architecture/Interface Description Language de l'OMG (Object Management Group)

1.3.2 Les lacunes des environnements actuels

Les environnements de développement d'agents de gestion actuels ne comportent pas d'outils pour la validation des spécifications, pour la génération d'informations de tests et de séquences de tests et ne contribuent pas à la mise en place d'un développement rigoureux de bases d'informations de gestion.

L'absence de ces outils est principalement due aux lacunes de la norme GDMO, qui par l'utilisation exclusive de la description informelle, ne permet pas la conception de tels outils de développement. Aussi, comme nous avons étendu le formalisme GDMO, nous pouvons considérér le développement d'outils plus puissants. Malheureusement, la non disponibilité du code source des outils existants, nous oblige également à développer les outils de base qui sont les analyseurs et le gestionnaire de bibliothèque de spécifications.

1.4 L'environnement MODE

L'environnement que nous avons développé s'appelle **MODE** (**M**anaged **O**bject **D**evelopment **E**nvironment). Il comporte quatre outils de manipulation et d'exécution de spécifications d'objets gérés. Cet environnement supporte les étapes de spécification et de validation de bases d'informations de gestion. L'outil est structuré sur trois niveaux comme le montre l'architecture de la figure 1.2.

Au niveau le plus bas sont présents des outils de manipulation (gestion) de spécifications d'objets gérés. On y retrouve principalement des analyseurs et des fonctions de chargement de spécifications. Ces outils de base forment le noyau de l'environnement MODE. Ils sont décrits dans la section suivante.

Les informations fournies par le noyau sont accessibles par une interface de programmation d'applications (API[5]). Nous avons, dans le cadre de cette thèse, développé deux applications qui exploitent cette interface de programmation: l'outil d'aide à la construction de bases d'informations de gestion et l'outil de génération de scénarios CRUSADE. Ces applications forment la base des outils d'exploitation des spécifications formelles en LOBSTERS. Ils seront décrits dans les sections suivantes.

1.4.1 Le noyau MODE

Le noyau MODE fournit les fonctions de base pour la manipulation de spécifications LOBSTERS et ASN.1. Il comporte un ensemble d'outils de chargement, analyse syntaxique et manipulation de spécifications LOBSTERS et ASN.1. Cet outil fournit aussi, au travers d'une interface de programmation d'applications, toutes les fonctions nécessaires pour accéder à l'information contenue dans les spécifications.

Le noyau comporte deux outils de base qui sont des analyseurs de spécifications. Le premier permet de vérifier la syntaxe de spécifications LOBSTERS et le second la syntaxe de spécifications ASN.1. Du point de vue de l'implantation, nous avons développé l'analyseur LOBSTERS dans le cadre d'un projet industriel avec trois étudiants de l'ESIAL [Abdelkrim 94]. L'analyseur ASN.1 est quant à lui basé sur le compilateur SNACC développé à l'université de British Columbia de Vancouver [Sample 93]. L'analyseur LOBSTERS dispose d'une interface graphique pour simplifier sa manipulation.

L'interface de programmation d'application met à disposition des outils applicatifs un ensemble de fonctions qui permettent d'extraire des spécifications précédemment analysées toute l'information nécessaire à leur traitement (liste des classes, type des attributs, comportement d'un objet, ...). Cette interface a été développée en C++.

5. . Application Programming Interface

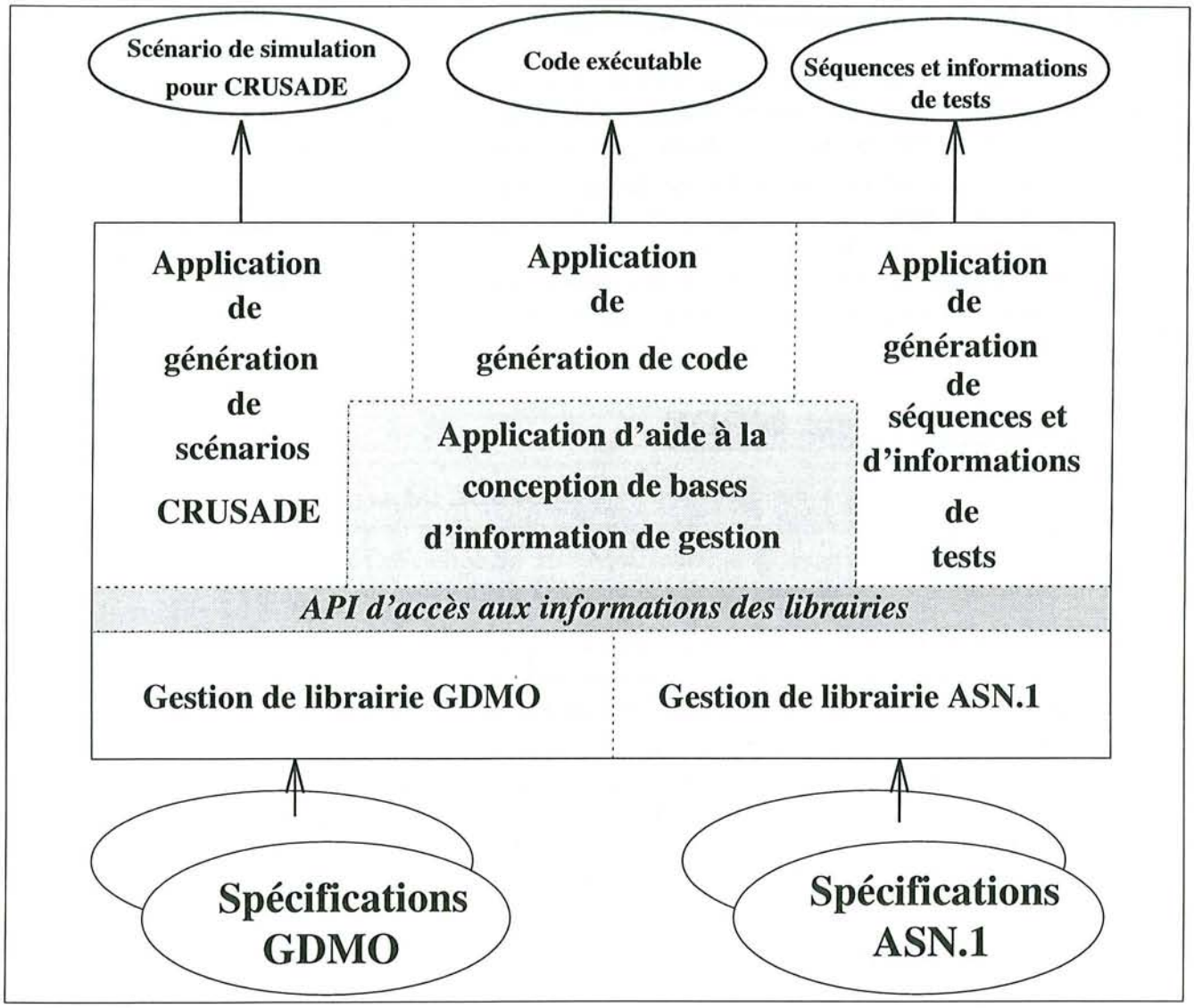

FIG. 1.2 - . *Architecture générale de l'environnement MODE*

1.4.2 Les outils applicatifs

Le noyau met à disposition des outils applicatifs toute l'information contenue dans une spécification LOBSTERS au travers de l'interface de programmation d'applications. Nous avons envisagé quatre applications qui exploitent cette information. Ces quatre applications, illustrées dans la figure 1.2, sont:

- un outil d'aide à la conception de bases d'informations de gestion,
- un générateur de scénarios pour la simulation interactive,
- un outil de génération de code,
- un outil de génération de tests et d'informations de conformité.

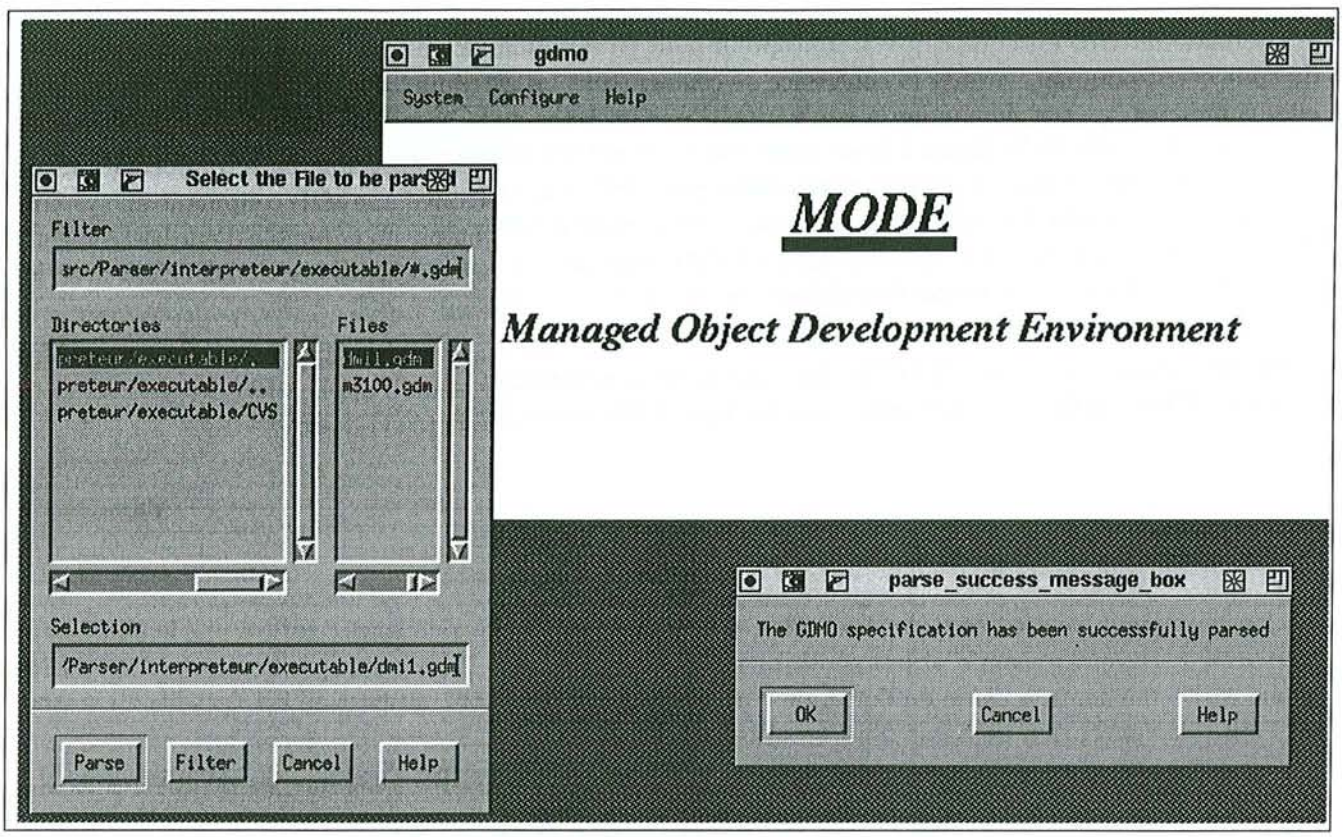

FIG. 1.3 - . *Le noyau de l'environnement MODE*

Ces quatre applications forment la base des outils de développement de l'environnement MODE. Dans le cadre de la thèse, nous avons entrepris le développement des deux premières applications de la liste c'est à dire l'application d'aide à la conception de MIBs et le simulateur interactif.

L'outil de conception de bases d'informations de gestion

L'outil d'aide à la conception de bases d'informations de gestion est une application graphique qui permet de construire, à partir des spécifications existantes et de manière interactive, des organisations hiérarchiques de bases d'informations de gestion. Cet outil a pour but de faciliter le travail de recherche et de réutilisation de spécifications d'objets gérés déjà définis dans le système et de fournir le support maximal pour la construction de bases d'informations de gestion. Une base d'informations de gestion constituée à partir de cet outil, servira de donnée aux autres applications de l'environnement.

La figure 1.4 illustre l'interface de l'outil de conception de bases d'informations de gestion.

L'outil exploite les informations fournies par l'analyse et le chargement de fichiers de spécifications LOBSTERS ou GDMO standard et met à disposition du concepteur de MIB, la liste des classes d'objets gérés disponibles, l'ensemble des corrélations de noms, les relations et les corrélations de relations. A partir de ces informations, l'utilisateur peut composer l'arbre de nommage et la hiérarchie de la base d'informations qu'il désire construire. A tout moment, il peut éditer une définition, ajouter ou retirer un couple (classe, corrélation) ou (relation, corrélation) dans la MIB et vérifier la cohérence de la base. La structure hierarchique de la MIB en construction est toujours visible sur la partie gauche de l'interface.

Lorsqu'une MIB est construite, le concepteur peut en sélectionner une partie, demander la génération du code correspondant, vérifier la cohérence de celle-ci (tous les attributs sont-ils définis, les classes sont-elles complètes, ...) et demander la génération d'une instance de la base dans le but de simuler celle-ci.

Dans l'exemple de la figure 1.4, on peut voir la structure hiérarchique d'une base d'informations de gestion construite d'après les classes normalisées par l'ISO. On y retrouve une MIB contenant un système qui lui même contient les objets de type journal. Un journal peut contenir un ensemble d'enregistrements. L'outil accepte à la fois des spécifications GDMO standard et des spécifications LOBSTERS avec les schémas de relation, corrélation de relations et comportement formel.

Cet environnement de construction est implanté en C++, sur une machine SUN. L'interface graphique a été développée sous UIL[6]/MOTIF. Au niveau de la génération, seule l'application de génération de scénarios CRUSADE a été envisagée dans la thèse. Celle-ci est présentée ci-dessous.

L'outil de génération de scénarios CRUSADE

Le générateur de scénarios de simulation permet de générer le code source utilisé par le simulateur interactif de comportement d'objets gérés. La génération se fait sur la base des spécifications LOBSTERS d'une MIB construite à l'aide de l'outil de conception de MIB. Cette application est appelée au travers de l'outil de conception de bases d'informations.

Le générateur prend en entrée une série de définitions LOBSTERS ainsi que les définitions ASN.1 associées, et génère les systèmes de règles associés aux objets gérés sélectionnés, les portes de communication spécifiées dans ces objets, ainsi que les tables ASN.1 nécessaires au traitement des attributs et variables locales. Le code CRS ainsi généré servira d'entrée au simulateur décrit dans le chapitre suivant.

L'algorithme de cet outil est basé sur la fonction de collecte de comportement que nous avons définie dans le chapitre précédent.

L'outil de génération de code

Le générateur de code de l'environnement MODE possède un double rôle. Il a comme premier but de générer du code pour les objets gérés dans un environnement d'accueil donné. Sa seconde tâche est de générer les informations qui seront exploitées par l'outil de génération de tests. Nous nous sommes intéressés à ce niveau aux apports que pouvait fournir LOBSTERS pour la génération d'informations de tests et pour l'extension de l'interactivité entre l'outil de génération et le concepteur.

Sur ce point, il est apparu que l'on pouvait gagner en interactivité dans le processus de développement grâce à la formalisation des conditions de présence des modules optionnels. Cette formalisation permet également de générer automatiquement ou semi-automatiquement les informations de conformité présentes dans un MOCS[7]. Il est le formulaire normalisé à fournir avec toute implantation en vue d'appliquer des tests de conformité.

Ceci dit, nos idées dans ce domaine n'en sont qu'au stade embryonnaire et des recherches additionelles doivent être entreprises à ce niveau pour intégrer ces aspects dans l'environnement. Aussi, aucune implantation de ce type n'est actuellement disponible sous MODE.

---

6. . User Interface Language
7. . Managed Object Conformance Statements

L'outil de génération de tests

Une autre application de support de développement, susceptible de tirer profit des extensions fournies par LOBSTERS, est la génération de tests. En effet, différents travaux ont déjà été entrepris sur la génération de tests pour des systèmes orientés-objets à partir de représentations du comportement sous forme de systèmes de transitions étiquetés [Yao 93].

Sur le plan des outils de génération de tests pour les objets gérés, des travaux sont actuellement en cours dans notre groupe [Kaboré 94a, Kaboré 94b]. Des applications de génération de tests prévues dans le cadre de ces travaux sont envisagées sur la base de l'interface de programmation de MODE.

D'autres recherches qui exploitent la description du comportement offerte par LOBSTERS sont également entreprises à l'ENC [Bär 93]. Aucune implantation n'est prévue dans ce cadre.

1.5 Résumé

Nous avons dans ce chapitre fixé les bases pour la conception d'un environnement de développement d'agents de gestion, qui supporte le concepteur d'objets gérés de la spécification jusqu'à l'implantation de son système. L'étude du processus de développement basé sur une approche formelle nous a permis d'identifier, dans les différentes phases, les outils logiciels qui peuvent aider le concepteur d'une base d'informations de gestion dans son développement.

Au cours du survol des environnements existants, nous avons mis l'accent sur la disponibilité restreinte d'outils de développement. Cette absence de support est principalement due aux lacunes du formalisme GDMO en matière de spécification du comportement et des conditions de présence des modules optionnels. Nous avons également remarqué que tous les environnements existants proposent des outils pour les deux étapes extrêmes du processus qui sont la formalisation des besoins et la génération de code.

Comme nous avons étendu le langage GDMO avec des mécanismes formels pour le comportement et les modules optionnels, nous pouvons étendre les fonctionnalités des environnements de base. Dans ce cadre, nous avons proposé notre propre environnement de développement appelé MODE. Au travers de ces différents outils, l'environnement MODE a pour but de soutenir le concepteur de bases d'informations de gestion durant tout le cycle de développement. Il comporte des outils exploitables à toutes les étapes.

De par son architecture modulaire, l'environnement MODE se veut ouvert et permet la conception et le développement de nouvelles applications utiles au développement. Ces applications peuvent extraire l'ensemble des informations contenues dans le noyau au travers de l'interface de programmation conçue à cet effet. A ce jour, l'environnement est encore au stade de prototype et toutes les applications envisagées n'y sont pas implantées.

Dans le chapitre suivant, nous allons présenter l'application principale que nous exploitons dans le processus de développement. Cette application est un simulateur permettant de simuler de manière interactive des spécifications CRS générées à partir de descriptions formelles LOBSTERS.

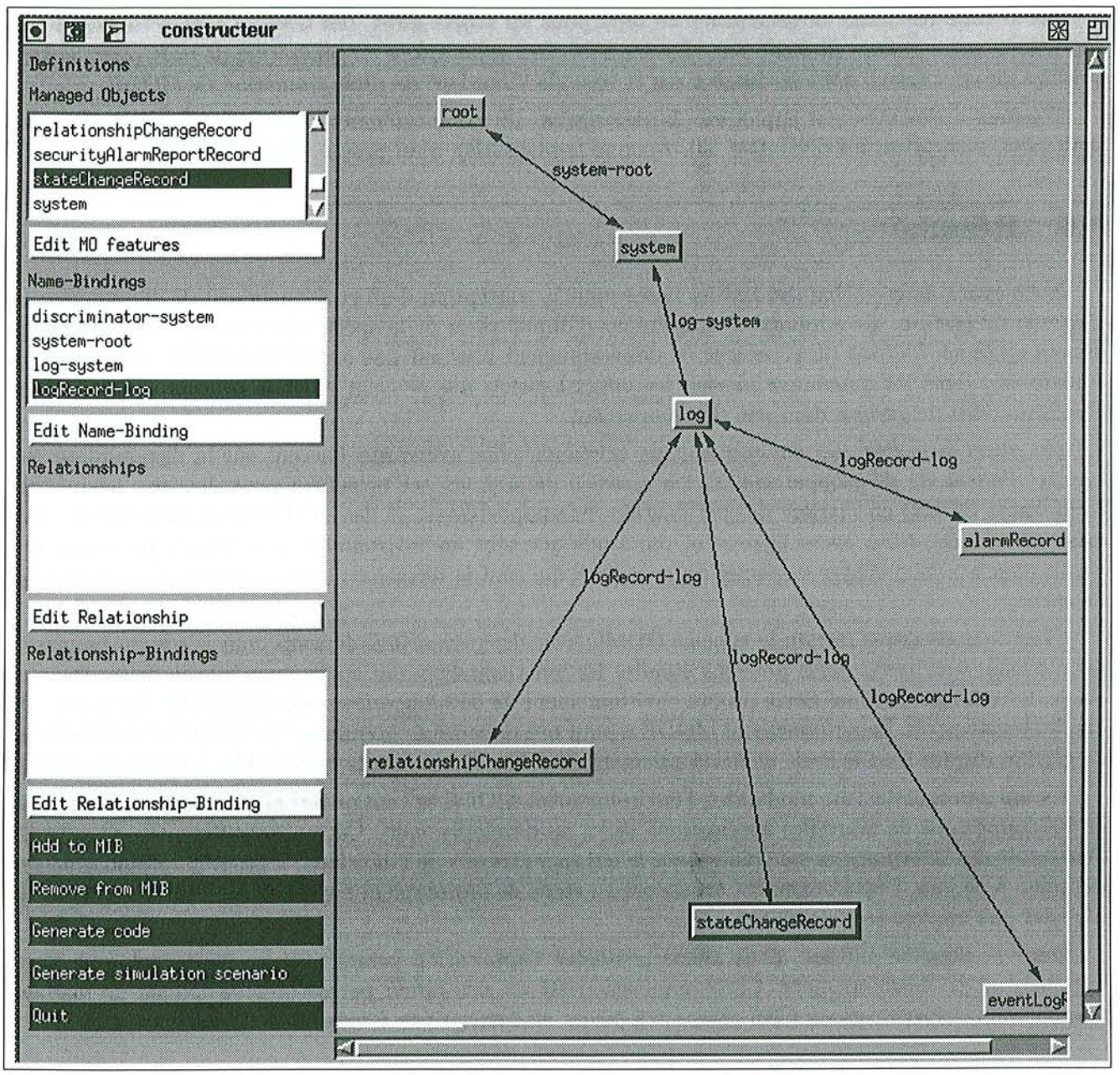

FIG. 1.4 - . *L'interface de construction de bases d'informations de gestion*

2
Une approche simulative pour la validation d'interfaces

La première application basée sur le formalisme LOBSTERS que nous avons envisagée dans le cadre de l'environnement MODE, et plus généralement pour le support de conception d'applications de gestion de réseaux, est une application de simulation de base d'informations de gestion. Ce chapitre en fait la présentation.

2.1 Introduction

Dans le cadre des travaux entrepris à l'ENC sur un environnement de développement de protocoles, une approche homogène pour la validation de spécifications de protocoles en CRS a été envisagée. Très vite, les recherches se sont orientées vers la conception d'un outil de simulation interactif pour les raisons suivantes:

1. un tel outil permet au concepteur d'analyser un modèle de son système dès les premières phases de sa conception et ceci jusqu'à la phase finale d'implantation.

2. il permet aussi de se focaliser sur différents niveaux de détails allant des interactions entre le modèle et son environnement jusqu'à l'étude très détaillée de son état interne et du contenu des données transmises lors des interactions.

Dans l'environnement CRS, l'outil qui permet cette analyse s'appelle CRUSADE[1]. Nous l'avons retenu comme base pour le support logiciel des phases de spécialisation du processus de conception de bases d'informations de gestion.

Après une présentation des fonctionnalités de l'outil, nous détaillons les extensions que nous y avons apportées ainsi que la manière dont l'outil peut être intégré dans l'environnement de développement présenté dans le chapitre précédent. Dans le cadre d'un projet de recherche sur les réseaux privés virtuels, l'outil ainsi que les extensions que nous proposons à GDMO ont été appliqués pour la validation des bases d'informations de gestion et des interfaces. Nous présentons rapidement cette mise en oeuvre et les expériences acquises sur cet exemple.

1. . Communicating RUle Systems Automated Development Environment

2.2 L'outil CRUSADE

L'outil CRUSADE a été initialement développé à l'ENC, par Wilko Eschebach [Eschebach 91]. Nous avons repris et étendu l'outil en vue de l'utiliser dans le cadre de l'environnement MODE. Après une description des fonctionnalités de base de l'outil, nous détaillons les extensions que nous y avons apportées.

2.2.1 Les fonctionnalités

Dans sa version initiale développée à l'ENC le simulateur CRUSADE offrait les fonctionnalités suivantes:

- visualisation des valeurs des variables d'état d'instances de systèmes de règles
- visualisation de règles déclenchables (resp. terminables) de systèmes de règles
- visualisation des occurences d'événements
- déclenchement (resp. terminaison) de règles au sein d'un ou plusieurs systèmes

Ce sont les fonctions de base pour l'exploitation d'une spécification CRS. Comme nous utilisons l'outil dans le cadre des applications de gestion, un certain nombre de limites sont rapidement apparues lors de sa mise en oeuvre sur des spécifications complexes de la couche appliciation du modèle OSI. Ces limites concernaient principalement le support des types ASN.1, restreints aux types simples (entiers, booléens, chaines de caractères). Comme l'ensemble des types ASN.1 sont autorisés dans GDMO et donc dans LOBSTERS, il a été nécessaire d'intervenir à ce niveau sur le simulateur.

2.2.2 Les extensions apportées

Pour permettre sa mise en oeuvre dans le processus de développement d'agents de gestion et dans notre environnement, il a fallu, d'une part porter l'outil d'un environnement OS/2 PM[2] sur un environnement UNIX, et d'autre part étendre certaines de ses fonctionnalités.

La première tâche a été réalisée sous mon autorité dans le cadre d'un projet industriel par quatre élèves de l'ESIAL [Frot 93]. Ce travail a permis, d'une part de porter l'ensemble de l'outil sur UNIX, et d'autre part de redéfinir entièrement l'interface utilisateur. Cette première activité sur le simulateur a également abouti à sa première utilisation dans le cadre de la gestion de réseaux à travers la spécification formelle en CRS et la simulation d'une version simplifiée du protocole CMIP.

La figure 2.1 présente le résultat de la première activité que nous avons entreprise sur l'outil. Dans cet exemple, une simulation d'une entité d'application, fournissant le service CMIS, est simulée. On retrouve l'architecture de deux entités d'application comportant chacune une entité ACSE pour la gestion de l'association d'application, une entité CMIP qui utilise les services d'une entité ROSE.

Cette simulation nous a permis de valider la nouvelle interface graphique et en même temps de mettre à jour un grand nombre de problèmes liés au support restreint de la notation ASN.1 dans l'environnement CRUSADE.

Or, le langage GDMO utilise ASN.1 comme langage de types et ceci de manière complète. Pour permettre une utilisation ultérieure de l'outil pour la simulation des bases d'informations de gestion, il

2. . Operating System/2 Presentation Manager

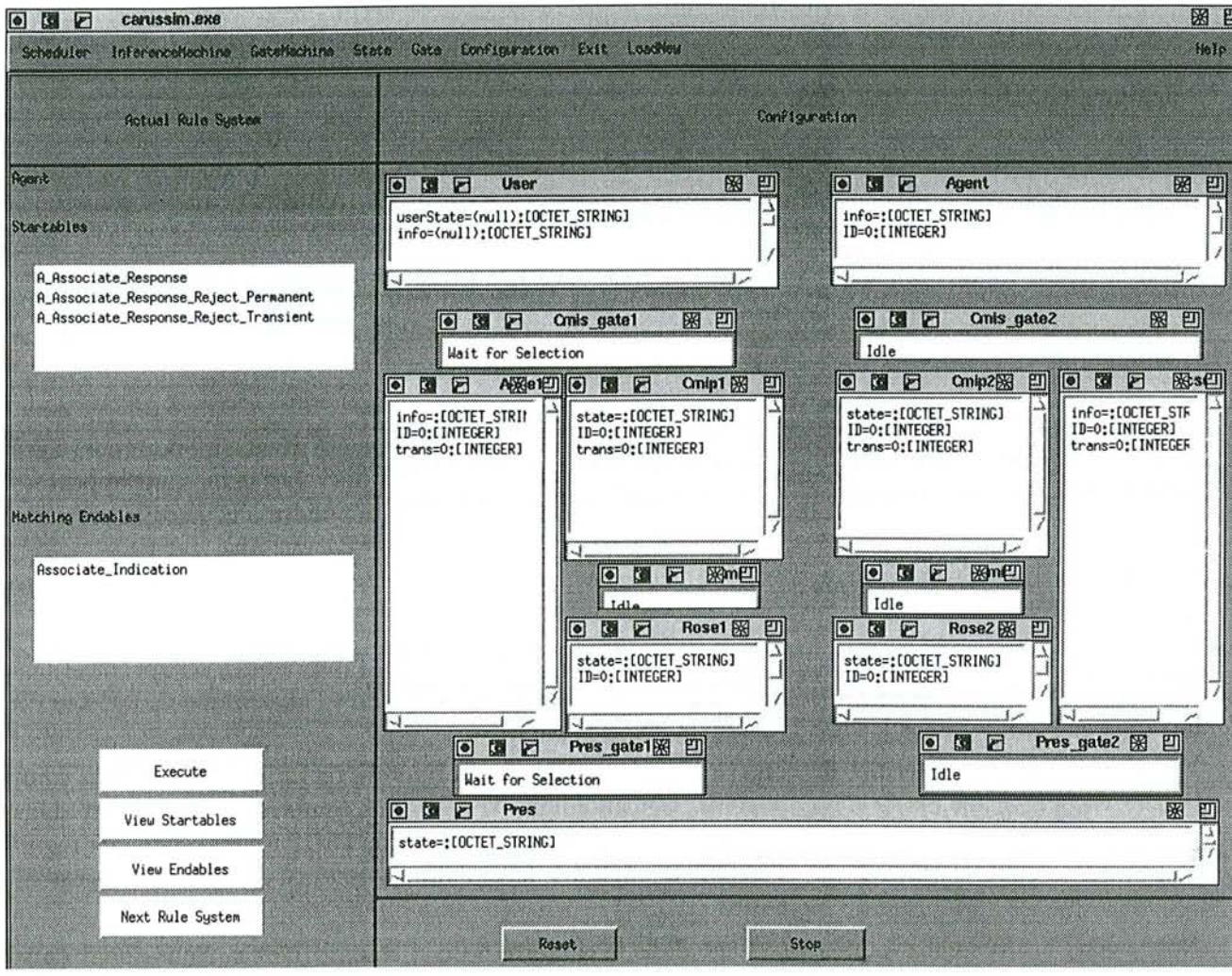

FIG. 2.1 - . *La validation du service CMIS*

a fallu reconsidérer entièrement la partie ASN.1. Ceci a été réalisé sous mon contrôle par David Orain lors d'un stage à l'ENC [Orain 93] aucours duquel, le compilateur ASN.1 a été entièrement changé et les fonctions de manipulation des données de l'interpréteur étendues.

Pour la partie analyse syntaxique d'ASN.1, l'outil SNACC[3] a été retenu. Après une étude détaillée de la représentation interne des types ASN.1, un générateur de tables à été développé en "back-end" sur SNACC. Ce générateur permet de construire l'ensemble des tables de types dont a besoin le simulateur CRUSADE.

Les extensions apportées par ce développement portent sur:

- l'analyse syntaxique: celle-ci est bien plus complète avec le nouveau compilateur, notamment au travers du support de la norme de 1990 et de la prise en compte du concept de module ASN.1 non supporté dans le compilateur initial de CRUSADE,
- les types structurés: ceux-ci sont désormais supportés par le compilateur et le simulateur. Notam-

3. . Sample Neufeld ASN.1 to C/C++ Compiler

ment, les types `CHOICE` et `ANY` couramment utilisés dans des spécifications GDMO. Le nouveau compilateur fournit également un support pour le type identificateur d'objet utilisé dans les spécifications GDMO et LOBSTERS.

- le sous-typage qui est supporté dans la nouvelle version de l'environnement. Il concerne les clauses de taille d'éléments (SIZE), les contraintes de présence PRESENT, ABSENT sur les sous-types ainsi que la clause de sous-typage WITH-COMPONENTS.

- les notation valuées qui sont également supportées par le nouveau compilateur et des tables associées sont générées pour le simulateur. Ces tables ne sont cependant pas exploitées à ce jour par le simulateur.

L'intégration d'un nouvel outil pour la manipulation des types ASN.1 a été nécessaire pour envisager l'exploitation de l'environnement dans le cadre des spécifications LOBSTERS. Les extensions apportées à l'outil SNACC et son intégration dans CRUSADE nous ont permis de répondre à ce besoin.

2.2.3 Le lien LOBSTERS-CRS

Le simulateur CRUSADE permet la simulation de spécifications CRS. Pour obtenir ces spécifications CRS à partir des spécifications LOBSTERS, nous appliquons à ces dernières l'algorithme de collecte du comportement présenté dans le chapitre 5 de la seconde partie.

En effet, l'algorithme permet de construire un ensemble homogène de règles qui décrivent le comportement associé à une classe d'objets gérés. Cet ensemble de règles est conforme à celui formé dans un système de règles. Pour compléter le passage d'une spécification LOBSTERS à un système de règles CRS, il faut construire en plus de cet ensemble de règles, le formulaire de système de règles, les variables d'états ainsi que les variables intermédiaires.

Nous avons brièvement évoqué ce passage dans le chapitre 5 de la seconde partie. Nous allons dans les lignes suivantes présenter plus en détail les correspondances entre une spécification d'un objet géré à laquelle l'algorithme de collecte a été appliqué et le système de règles CRS correspondant.

La construction est relativement simple. Pour tout objet géré sélectionné dans le constructeur de MIB, un système de règles est généré. Dans un premier temps, l'algorithme de collecte du chapitre précédent est appliqué à chaque objet. Une fois cet algorithme appliqué, la génération du système de règles associé est entreprise comme suit:

- le système de règles a le même nom que l'objet géré associé. Ses paramètres sont, d'une part les interfaces auxquelles l'objet accède ,et d'autre part une liste de variables booléennes qui ont chacune le nom d'un module optionnel de l'objet géré.

- les variables d'état du système de règles (clause **STATES**) sont les attributs instanciés de l'objet géré ainsi que tous les attribust hérités, et un ensemble de variables booléennes qui représentent la présence ou l'abscence de modules optionnels.

- les variables locales du système de règles (clause **DECLARATIONS**) sont les mêmes que celles du comportement de l'objet géré associé et comportent aussi toutes les variables locales héritées. Il en est de même pour les prédicats utiles qui sont instanciés dans la clause **DEFINITIONS**..

- les règles de comportement sont celles de l'objet géré après application de l'algorithme de collecte.

Les interfaces sont transformées en portes de communication **GATE** d'après les schémas d'interface présentés dans le chapitre 5 de la seconde partie en appliquant les règles d'héritage sur les interfaces définies dans le chaiptre 4 de la seconde partie.

Comme nous l'avons déjà mentionné dans le chapitre précédent, cet algorithme qui constitue en fait l'outil de génération de scénario pour CRUSADE est en cours d'implantation. Aussi, l'ensemble des transformations LOBSTERS vers CRS que nous avons effectuées à ce jour l'ont été à la main.

2.3 La simulation des systèmes de gestion des réseaux privés virtuels

Un des projets en cours dans l'équipe "Administration de Systèmes et Réseaux" de l'ENC porte sur la conception et la réalisation d'un système de gestion de réseaux privés virtuels (VPN[4]) [Schneider 93a]. Nous avons retenu ce cadre pour mettre en oeuvre le formalisme LOBSTERS et l'outil de simulation CRUSADE. Ceci nous a permis d'acquérir une première expérience sur un développement concret et important de base d'informations de gestion complète. Après une brève présentation du cadre de l'application, nous allons présenter quelques spécifications et simulations réalisées avec LOBSTERS et CRUSADE.

2.3.1 Le cadre de l'application

Le projet a pour but de définir un ensemble de services de gestion ainsi que les bases d'informations de gestion correspondantes pour permettre la mise en place d'un réseau privé virtuel entre deux réseaux locaux interconnectés au travers d'un réseau public constitué d'un sous réseau DQDB et d'un sous réseau ATM [Schneider 93b].

Dans ce but, des services et trois bases d'informations ont été conçus. La complexité de celles-ci, ainsi que la volonté des personnes ayant travaillé sur ce projet, nous ont permis de mettre en oeuvre le formalisme et l'outil de simulation pour valider ce système.

2.3.2 Les résultats obtenus

L'approche LOBSTERS a été utilisée pour spécifier les objets intervenants dans les différentes bases d'informations de gestion, c'est à dire la MIB du gestionnaire privé et celle du système de gestion du réseau public. Le simulateur a été utilisé pour la validation des interfaces de communication et pour la validation de la base d'informations de gestion du gestionnaire du réseau public.

La validation du modèle entrepris par Thomas Preuss et Juergen Schneider (concepteurs du modèle) [Preuss 93, Schneider 93b] a été divisée en trois phases distinctes:

1. **validation des objets gérés:** test d'objet géré isolé. Cette étape se concentre sur la validation des interfaces d'un objet géré isolé de son entourage. Elle permet de valider le comportement de cet objet.

2. **validation de la MIB:** simulation d'un scénario de la MIB. Cette étape consiste a simuler une base d'informations de gestion comportant tous les objets spécifiés dans l'application. Elle permet de valider la consistance de la MIB en observant l'ensemble des interactions entre la MIB et l'agent, ainsi que les interactions entre les objets au sein de la MIB.

4. . Virtual Private Network

3. **validation des interfaces de gestion:** simulation des interactions entre les différents intervenants dans la gestion des réseaux privés virtuels. Cette dernière étape a permis de valider les services de gestions offerts par les systèmes de gestion des réseaux des clients et ceux du gestionnaire du réseau public.

Dans les pages suivantes, nous allons présenter de manière plus détaillée ces trois phases.

Validation des objets gérés

Cette première étape du processus de validation des objets gérés se concentre sur le comportement d'un objet géré isolé de son entourage. Pour ce faire, on considère une instance de l'objet que l'on stimule en y invoquant des opérations de lecture, d'affectation tout en observant les notifications émises.

Dans le cadre du projet, l'objet VPN a été retenu pour une validation isolée. Ce dernier est l'un des plus importants d'une base d'informations de gestion du système de gestion du réseau public. Il modélise un réseau privé virtuel alloué à un utilisateur. Il est dérivé de l'objet géré `opNetwork` du forum qui modélise un réseau. Les attributs de l'objet géré VPN sont:

- les états opérationnels et administratifs (attributs hérités),
- la liste des services offerts par le réseau (attribut hérité),
- les attributs de largeur de bande (`maxBandwidth` largeur maximale allouable sur le réseau, `usedBandwidth` largeur de bande allouée),
- des attributs qui identifient les liens et les points d'accès des clients sur le réseau.

Les actions autorisées sur cet objet sont: l'ajout, la suppression et la modification de points d'accès, l'établissement et la terminaison de liens, la modification de la largeur de bande.

Pour permettre sa validation, l'objet géré VPN a été spécifié en LOBSTERS, puis traduit à la main en CRS. Cette spécification a été utilisée pour la simulation de l'objet dans CRUSADE. La figure 2.2 comporte une partie du système de règles associé à la spécification de l'objet géré: le comportement sur une requête de lecture d'attribut et la requête de réservation de bande passante.

La simulation de l'objet a permis de vérifier la validité des conditions et effets associés à chaque occurence d'événement au niveau de l'objet géré.

Validation de la MIB

La validation des bases d'informations de gestion compose la deuxième étape du processus de validation. Elle a pour but de simuler une base d'informations complète, c'est à dire comprenant l'ensemble des objets susceptibles d'y apparaître. Dans le cadre du projet, la base d'informations de gestion du gestionnaire public a été retenue comme candidate pour cette tâche.

Dans la figure 2.3, on retrouve une illustration de l'exécution de la simulation des objets d'une MIB du réseau public impliqués dans le processus de réservation de lien privé. Les objets impliqués dans cette simulation sont le fournisseur de service du réseau privé virtuel `VpnServiceOsf` qui interagit avec l'objet géré réseau virtuel `Vpn`. Cet objet contient des objets liens `link` qui modélisent un lien entre deux utilisateurs sur le réseau privé virtuel. Un lien se fait toujours au travers de deux points d'accès au service d'où la présence des objets `Cap` qui modélisent les points d'accès des utilisateurs. Les objets `VpnSrcxx` modélisent des services établis sur un lien.

```
RULE_SYSTEM Vpn(id : INTEGER;
                VG : vpn_gate;
                VLG: vpn_link_gate;
                BG : vpn_bw_reservation_gate;
                NOT: vpn_notification_gate);

STATE
   allocatedBw          : REAL INITIALLY 0;
   administrativeState: AdministrativeState INITIALLY unlocked;
   operationalState    : OperationalState INITIALLY enabled;
   status              : VpnStatus INITIALLY idle;
   ...

RULES
  GetBw;
   COND: VG.Get(allocatedBwId);
   EFFECTS: VG.GetRsp('allocatedBw);

 incommingRequestBwAccepted;
   COND: VG.AC_ReserveBw(reserveBwInfo)
     AND administrativeState = unlocked
     AND operationalState    = enabled
     AND status = idle
     ....
   EFFECTS: VLG.RequestBwOnLink(VpnLink.Id,
                                reserveBwInfo.bw,
                                reserveBwInfo.qos)
     AND status = bwReservationInProgress;
```

FIG. 2.2 - . *Spécification du comportement de l'objet VPN*

La simulation consiste à déclencher toutes les opérations suivant un séquencement préétabli sur l'objet `Vpn` et de vérifier que les objets de la MIB restent consistants après l'exécution (valeurs des attributs) et que les interactions entre instances d'objets gérés soient bien celles qui sont souhaitées. Notez qu'à tout moment de la simulation, l'ensemble des attributs des objets gérés sont visibles.

Validation des interfaces

La validation des interfaces représente la dernière étape du processus de validation. Celle-ci est très importante car elle permet de visualiser les échanges entre les différents intervenants (dans notre cas des agents et des gestionnaires) et de vérifier les propriétés de minimalité, correction, et complétude présentées au début de ce chapitre.

Dans le cadre des réseaux privés virtuels, les interfaces entre le gestionnaire public et les gestionnaires des deux réseaux privés (clients) ont été spécifiées en CRS et des simulations sur le processus de réservation de réseau privé virtuel ont été entreprises.

La figure 2.4 est une copie de l'interface du simulateur durant une simulation des services de réservation. On y trouve deux utilisateurs `user1`, `user2` situés sur deux réseaux locaux. Un utilisateur peut éta-

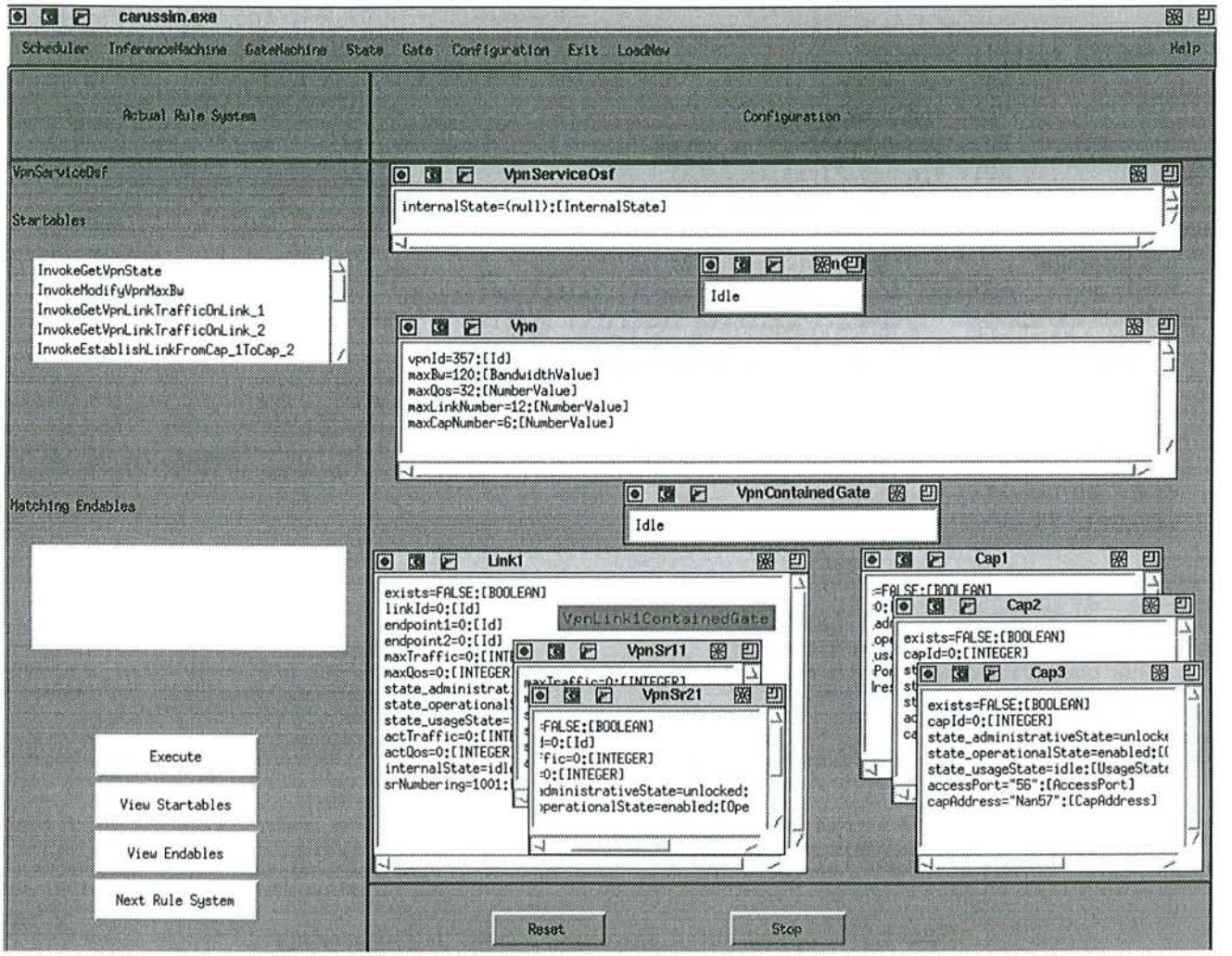

FIG. 2.3 - . *La validation de la MIB du gestionnaire public*

blir un réseau privé virtuel avec son correspondant en s'adressant à son gestionnaire local `CpnServiceOs1` qui, après consultation de son réseau local `Cpn1OS`, va propager la demande au gestionnaire du réseau public `VpnServiceOsf` et au gestionnaire du réseau local du correspondant `CpnServiceOs2`. Le gestionnaire public va faire la réservation sur le réseau métropolitain `ManOs` et sur le réseau grande distance `WanOs` dont il a la charge. Le gestionnaire du réseau local du correspondant va faire la réservation sur son réseau local. Lorsque toutes les réservations ont été effectuées et que le lien devient actif, les deux utilisateurs peuvent alors faire transiter des données sur le réseau privé virtuel nouvellement établi.

La conception de cette simulation des interfaces a été très intéressante car elle a permis une première visualisation de l'exécution complète du processus de réservation. Elle a aussi servi de base aux discussions sur la validité du modèle.

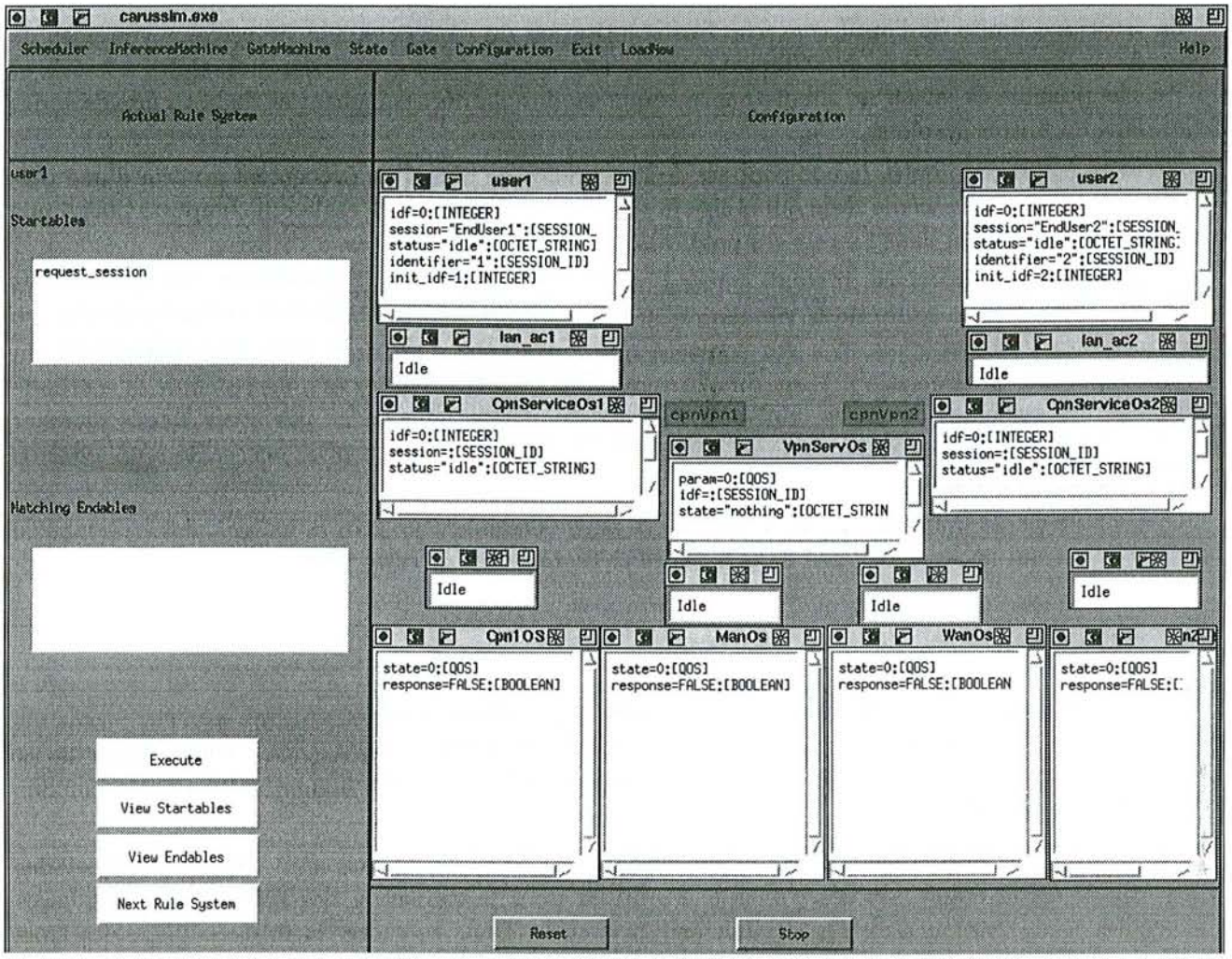

FIG. 2.4 - . *La validation du service de réservation des réseaux privés virtuels*

2.4 Limites actuelles du simulateur

Le simulateur possède encore un certain nombre de limitations qui empêchent partiellement son utilisation comme simulateur puissant d'objets gérés.

La limite principale de celui-ci concerne le non-support de la création et de la destruction dynamique d'instances de systèmes de règles, ce qui oblige le concepteur à spécifier dans son scénario un nombre donné d'instances et une fonction d'activation (resp. désactivation) qui joue le rôle de créatrice (resp. destructrice) d'instance.

La seconde limite du simulateur est l'impossibilité d'accéder à des portes de communication ouvertes. En effet, aucun mécanisme n'est fourni dans l'outil actuel pour créer des occurences d'événements de manière interactive à une porte de communication. Aussi, pour tester le comportement d'un objet géré (ici un système de règles), il est nécessaire de définir au préalable dans un autre système de règles la séquence des opérations que l'on désire invoquer et de coupler ce système de règles aux interfaces du système que l'on désire tester.

La troisième limite de l'outil concerne l'automatisation du comportement de certains systèmes de règles. Actuellement, on ne peut traiter de manière interactive qu'un seul système de règles à la fois et il n'est pas possible de laisser au simulateur la responsabilité du déclenchement et de la terminaison des règles dans les autres systèmes.

Dernière lacune de l'outil, le non-support de l'occurence de plus d'un événement au sein d'une pré- ou post-condition d'une même règle qui oblige le spécifieur à éclater toute règle qui comporte plus d'une occurence d'événements dans l'un de ces prédicats.

Du point de vue interface, il serait intéressant de reconsidérer la représentation des systèmes de règles. En effet, dans le cadre de la simulation de bases d'informations de gestion comportant plus de vingt instances d'objets gérés, il a été constaté que l'utilisateur perdait le sens de l'architecture du système en cours de simulation. Pour cela, il serait certainement intéressant de redéfinir la méthode d'affichage des systèmes de règles impliqués dans la simulation, afin de ne pas avoir à tout moment une vue sur l'ensemble des systèmes mais seulement sur ceux désirés. On peut penser ici à un système d'ouverture à la demande.

L'ensemble de ces limites peuvent être abrogées mais demandent un gros investissement en temps de développement qui ne se justifie pas dans le cadre et les objectifs de cette thèse.

2.5 Résumé

Nous avons dans ce chapitre présenté la première application rendue possible par l'extension de GDMO avec les mécanismes de règles issus de CRS. Cette application permet à toute étape de la conception d'une base d'informations de gestion de valider celle-ci par une simulation interactive à l'aide du simulateur CRUSADE.

Après avoir été portée sur UNIX, l'application a été étendue en vue d'un support complet des types ASN.1. Cette étape achevée, l'outil a été utilisé pour la validation des spécifications des bases d'informations de gestion pour les réseaux privés virtuels. Dans ce cadre, la mise en place des trois phases de validation a permis d'aborder de manière rigoureuse le processus.

Cette mise en oeuvre nous a permis, en coopération avec les personnes ayant spécifié les bases d'informations de gestion, d'acquérir une première expérience sur un cas réel de développement d'application de gestion. Les résultats obtenus dans ce cadre sont satisfaisants pour les développeurs de l'application de réservation de réseaux privés virtuels et pour nous. L'approche a permis aux concepteurs d'avoir une première expérience dynamique avec leur système. Cette expérience a révélé l'intérêt d'utiliser le simulateur, mais a également permis d'identifier un certain nombre de limites de l'outil.

Conclusion générale et perspectives

«L'avenir ne se prévoit bien qu'après coup.»

Robert Coeuillet

Le processus de développement d'agents et d'objets gérés pour les activités de gestion de réseaux hétérogènes dans le cadre des normes OSI s'appuie sur deux formalismes normalisés qui sont ASN.1 pour la description des données et GDMO pour la spécification des objets. Le but des travaux présentés dans cette thèse était d'étendre ces formalismes afin de permettre l'élaboration de spécifications formelles des systèmes qu'ils modélisent. De telles extensions sont souhaitables car elles permettent, d'une part d'ajouter de la rigueur au processus de développement d'agents de gestion, et d'autre part de mettre à la disposition des concepteurs de systèmes des outils puissants qui assistent le développement.

Résumé des résultats obtenus

L'analyse des approches de gestion et des formalismes existants

Dans cette thèse, nous avons dans un premier temps étudié différentes approches existantes pour la gestion des réseaux hétérogènes. Cette étude a été bénéfique à plusieurs niveaux. Elle nous a permis de bien comprendre l'intérêt de poursuivre sur les bases de l'approche OSI qui est, à ce jour, la solution la mieux adaptée, aussi bien pour la modélisation des réseaux complexes que pour l'implantation optimale et évolutive de systèmes de gestion de réseaux. L'étude détaillée du formalisme GDMO et de nombreux catalogues d'objets gérés nous a permis de bien identifier les problèmes que ce formalisme entraînait au niveau de la description du comportement. L'analyse de ces problèmes nous a, par la suite, facilité l'énumération de contraintes à placer sur la description du comportement et la définition d'exigences envers des techniques de description formelles, candidates à la formalisation du comportement des objets gérés.

La confrontation de nos contraintes avec les différentes techniques candidates à ce jour a clairement fait ressortir qu'aucune ne satisfaisait l'ensemble des exigences, ce qui nous a poussé à rechercher une alternative. Cette dernière devait certes permettre de décrire formellement le comportement des objets gérés, mais ceci en parfaite harmonie avec le formalisme GDMO qui est, rappelons le, normalisé et largement utilisé dans le monde. L'élaboration de cette FDT a été le premier but des travaux de la thèse.

Dès le début des travaux, nous avons identifié le besoin de mettre à disposition des concepteurs un environnement de développement d'agents de gestion qui tire profit des extensions que nous allions proposer à la norme. Cet environnement fut le second objectif de notre travail.

Le formalisme LOBSTERS

Pour mettre en place une formalisation du comportement des objets gérés dans GDMO, nous avons retenu les mécanismes de règles issus du formalisme CRS. Pour permettre l'utilisation de ces règles dans une approche orientée-objets, nous avons défini de manière formelle les concepts de l'héritage strict tel qu'il est imposé dans GDMO et nous les avons inclus dans CRS. Ceci nous a mené à une version orientée-objets de la FDT qui a pu, par la suite, être intégrée dans la notation GDMO.

De cette intégration est né le formalisme LOBSTERS. Celui-ci représente l'aboutissement du premier but que nous nous étions fixé dans cette thèse, à savoir la mise à disposition d'une FDT, compatible à GDMO et qui intègre les aspects liés au comportement des objets. En effet, le formalisme permet de garder la partie description statique des données de GDMO et de décrire le comportement de manière formelle grâce aux règles et interfaces introduites dans la technique.

L'algorithme de collecte lié à une localisation stricte des informations de comportement, que nous avons proposé permet de construire de manière rigoureuse les spécifications LOBSTERS. Cet algorithme permet aussi de construire un comportement homogène à partir des différents formulaires contenus dans un objet géré. Il constitue également une première étape vers la génération de systèmes de règles CRS à partir de spécifications LOBSTERS.

L'environnement de développement

Basé sur le formalisme LOBSTERS, nous avons conçu un environnement de développement d'objets gérés afin de finaliser le second but fixé dans la thèse. Cet environnement MODE fournit un ensemble de services qui eux permettent d'exploiter les descriptions formelles du comportement tout au long du processus de développement de bases d'informations de gestion.

Contrairement aux environnements de développements disponibles actuellement, la conception de MODE prévoit un certain nombre de tâches additionnelles non présentes dans les environnements standard.

Ces tâches additionelles concernent la validation interactive de spécifications formelles à différentes étapes de la conception, le support de génération d'informations de tests et de conformité ainsi qu'une interface de programmation pour l'intégration de nouveaux outils de développement.

L'outil de manipulation de définition, une application d'aide à la conception de bases d'informations de gestion, l'interface de programmation d'applications ainsi que l'environnement de simulation sont à ce niveau disponibles. Dans ce cadre, des travaux sont actuellement en cours dans notre groupe pour la conception d'outils de génération de tests.

Implantation et premières expériences

Les limites identifiées

Nous avons appliqué le formalisme LOBSTERS à la spécification de quelques fonctions de gestion qui présentent les objets gérés d'un système informatique, à une partie de l'objet de journal ainsi qu'aux objets gérés utilisés pour la modélisation des réseaux privés virtuels.

Le formalisme LOBSTERS permet de décrire formellement le comportement des objets gérés mais ne fournit actuellement aucun support pour les fonctions de création et destruction dynamiques d'instances d'objets. Des études préliminiaires ont été menées sur cet aspect, mais n'ont pas été prises en compte à ce jour dans le formalisme.

Le second aspect qui n'est pas supporté par LOBSTERS pour le moment, concerne l'extension des domaines de valeur pour un attribut donné. En effet, l'approche ne considère, dans sa version actuelle, que l'héritage simple strict et ne prend pas en compte l'extension des domaines sur des attributs hérités. Ces problèmes peuvent cependant être résolus assez facilement dans le formalisme.

Un autre problème lié au formalisme concerne l'ordonnancement des règles. Pour permettre cette spécification, il faut actuellement définir pour chaque objet géré des attributs internes qui maintiennent l'historique du comportement de l'objet. Ceci implique la définition d'attributs spécifiques qui ne concernent en rien ceux définis naturellement dans l'objet. Aussi, serait-il préférable d'éviter cet ajout et de fournir un autre mécanisme pour la description de l'ordonnancement.

L'environnement MODE

L'environnement MODE est encore dans une phase précoce du développement. Cependant, un certain nombre de fonctionnalités sont disponibles à ce jour. Nous disposons d'une base solide pour l'analyse statique des spécifications LOBSTERS (analyseur syntaxique et sémantique statique). Cet outil est stable et permet de charger tout type de spécifications. L'interface de programmation d'application construite sur l'analyseur, est également réalisée et permet l'accès aux informations d'une spécification. Sur l'interface de programmation d'application, nous avons conçu un outil d'aide à la conception de base d'informations. Il met à disposition du concepteur de la MIB une interface conviviale, lui permettant de construire de manière interactive et simple l'architecture de la base d'informations de gestion qu'il désire créer à partir des classes d'objet gerés existants, des corrélations de noms, des relations et des corrélations de relations disponibles dans la base des spécifications. Cet outil est partiellement opérationnel et intégré dans l'environnement MODE.

En plus des applications mentionnées ci-dessus, nous avons travaillé sur le simulateur CRUSADE et sur le support ASN.1 de l'outil. Celui-ci supporte désormais la plupart des types ASN.1 dans les spécifications et dispose d'une implantation sous UNIX bien plus stable que la version précédente.

Nous travaillons actuellement à l'implantation de l'algorithme de génération de systèmes de règles CRS pour permettre une intégration de l'environnement de simulation dans MODE. Ce travail est actuellement encore au stade embryonnaire en ce qui concerne l'implantation.

Les applications actuelles

Le cadre dans lequel nous avons pu considérer le processus de développement avec LOBSTERS, a été celui des travaux entrepris à l'ENC sur les réseaux privés virtuels. Ce travail d'utilisation du formalisme pour le comportement des objets gérés, ainsi que la simulation à l'aide de CRUSADE de plusieurs spécifications, nous ont permis de prouver l'intérêt de l'utilisation d'une FDT dans le développement de bases d'informations de gestion et de services complexes. Ils nous ont également permis d'identifier un certain nombre de limites du formalisme et du simulateur.

Perspectives et travaux futurs

Sur les formalismes normalisés

De nombreux travaux sont actuellement en cours sur les formalismes utilisés à l'ISO. On note notamment un foisonnement de travaux sur les approches objets comme dans l'ODP ou ASN.1 par exemple. De plus, les formalismes normalisés évoluent et proposent des mécanismes de description des objets comme c'est notamment le cas pour la nouvelle version d'ASN.1 qui n'est plus en harmonie avec les normes uti-

lisées pour le service et le protocole de gestion CMIP. Au niveau des techniques de description formelle normalisées, les choses évoluent également avec la normalisation prochaine des FDT VDM et Z.

Donc, du point de vue des notations et formalismes utilisés à l'ISO, une des tâches primordiales à envisager (et avant toute autre) est la mise à plat de tous ces formalismes et l'harmonisation de ceux-ci. Comme l'approche objet semble percer définitivement dans l'OSI, on pourrait envisager une fusion des travaux avec ce qui se fait dans le cadre de l'ODP. Ceci est cependant tout à fait illusoire et utopique. La seule issue envisageable à ce niveau est l'intégration d'ASN.1 comme langage de description de types dans les FDTs normalisées.

Sur le formalisme LOBSTERS

Le formalisme LOBSTERS que nous avons conçu sur la base de GDMO et de CRS permet de décrire le comportement des objets gérés mais aucune recherche n'a été entreprise sur la description du comportement des relations entre objets gérés. Dans MODE nous avons retenu un schéma de relations étendu pour la spécification des liens entre objets gérés. Un des axes principaux de recherche à suivre dans ce domaine sera celui de la spécification du comportement des relations et l'incidence du comportement des relations sur les instances d'objets gérés qui y participent. Nous envisageons fortement de poursuivre des recherches dans ce domaine afin de prendre en compte les toutes dernières normes en vigueur sur le modèle relationnel (GRM).

D'autres points sont également à éclaircir et des extensions sont à proposer dans le formalisme. Ces points sont, principalement, la création et la destruction d'instances ainsi que la proposition d'un mécanisme basé sur la logique temporelle pour la spécification des ordonnancements des règles, sans passer par des attributs spécifiques comme cela est le cas actuellement.

Sur l'environnement MODE

L'environnement MODE est, dans sa version actuelle, au stade de prototype. Il supporte entièrement le langage LOBSTERS mais la partie ASN.1 est basée sur la version 90 de la norme. Une première extension de l'environnement à envisager sera le passage de la norme 1990 à celle de 1992. Des travaux sur le compilateur ASN.1 ont déjà été entrepris à l'université de Nancy 1, mais l'intégration dans l'environnement, ainsi que les conséquences sur le formalisme LOBSTERS, n'ont pas été étudiées à ce jour.

La partie génération de code de l'environnement MODE est également un domaine à explorer. En effet, nous avons dans le cadre de la thèse simplement émis quelques remarques sur l'intérêt de la formalisation du comportement dans un tel processus, mais les aspects spécifiques à la génération de code n'ont pas été étudiés.

Les extensions à envisager pour l'environnement MODE concernent principalement l'étude et la réalisation de nouveaux outils de support. Dans ce cadre, nous pensons notamment à la réalisation d'un outil de génération de tests et à l'intégration de l'outil dans un environnement d'accueil, telle qu'une plate-forme de gestion. Mais avant cela, une phase de validation de l'outil par différents utilisateurs est nécessaire.

Sur l'approche en général

Les travaux à entreprendre sur la méthode de spécification sont probablement ceux qui sont les plus intéressants. En effet, nous avons proposé dans cette thèse des concepts qui ont l'avantage de permettre de garder un formalisme unique tout au long du développement. Or, ces concepts (telles que

la répartition et la collecte) sont indépendants du formalisme et on peut facilement les appliquer à d'autres FDT candidates à la description du comportement des objets gérés. Comme CRS n'est pas une FDT normalisée, il serait intéressant d'appliquer les concepts sur une notation normalisée ou en phase de l'être. Vis-à-vis de nos concepts, on peut penser à Z ou VDM pour une telle application.

C'est donc principalement dans cette direction que nous envisageons d'appliquer les résultats de cette thèse et de nous investir dans des recherches futures.

Bibliographie

[Abdelkrim 94] A. Abdelkrim, L. Busteau et E. Duval. *"Validation de spécifications GDMO"*. 1994. rapport de projet industriel, Ecole Supérieure d'Informatique et Applications de Lorraine, 1994.

[Bär 93] B. Bär. *"A Conformance Testing Approach for Managed Objects"*. 1993. Fourth International Workshop on Distributed Systems: Operations and Management, October 5-5, 1993, Long Branch, New-Jersey, USA.

[Bartocci 93] A. Bartocci, G. Larini et C. Romellini. *"A first attempt to combine GDMO and SDL techniques"*. O. Faergemand et A. Sarma, éditeurs, *SDL '93: Using Objects*, pages 333–345. Elsiever Science Publishers B.V., 1993.

[Black 89] S. Black. *"Objects and LOTOS"*. S.T. Vuong, éditeur, *Proc. 2nd Int. Conf. on Formal Description Techniques for Communication Protocols and Distributed Systems*, pages 285–297. North-Holland, 1989.

[Bochmann 89] G.v. Bochmann et M. Deslauriers. *"Combining ASN.1 support with the LOTOS language"*. *Protocol Specification, Testing and Verification IX*. North-Holland Publ., 1989. Proc. IFIP Int. Symp. on Protocol Specification, Testing and Verification, 1989.

[Bochmann 91a] G.v. Bochmann. *"Object-oriented specifications in OSI and distributed processing"*. *Tutorials of the 11th Int. Conf. on Protocol Specification, Testing and Verification*, pages 93–123, 1991.

[Bochmann 91b] G.v. Bochmann, P. Mondain-Monval et L. Lecomte. *"Formal Description of Network Management Issues"*. I. Krishnan et W. Zimmer, éditeurs, *Integrated Network Management, II*, pages 77–92. Elsiever Science Publishers B.V. (North-Holland), 1991. Proc. of the IFIP TC6/WG6.6 2nd. Int. Symp. on Integrated Network Management, Crystal City, Washington, D.C., 1-5 April, 1991.

[Carrington 89] D. Carrington, D. Duke, R. Duke, P. King, G. Rose et G. Smith. *"Object-Z: An Object-Oriented Extension to Z"*. S.T. Vuong, éditeur, *Proc. 2nd Int. Conf. on Formal Description Techniques for Communication Protocols and Distributed Systems*, pages 401–420. North-Holland, 1989.

[Case 88] J. Case, M. Fedor, M. Schoffstall et J. Davin, *"Simple Network Management Protocol"*, RFC1067, SNMP Research Inc., Performance Systems International, MIT Laboratory for Computer Science, August 1988.

[Case 90] J. Case, M. Fedor, M. Schoffstall et J. Davin, *"A Simple Network Management Protocol"*, RFC1157, SNMP Research Inc., Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[Case 93a] J. Case, K. McCloghrie, M. Rose et S. Waldbusser, *"Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2)"*, RFC1450, SNMP Research Inc., Hughes LAN Systems Inc., Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

[Case 93b] J. Case, K. McCloghrie, M. Rose et S. Waldbusser, *"Management-to-Manager Management Information Base"*, RFC1450, SNMP Research Inc., Hughes LAN Systems Inc., Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

[Case 93c] J. Case, K. McCloghrie, M. Rose et S. Waldbusser, *"Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)"*, RFC1448, SNMP Research Inc., Hughes LAN Systems Inc., Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

[Case 93d] J. Case, K. McCloghrie, M. Rose et S. Waldbusser, *"Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)"*, RFC1442, SNMP Research Inc., Hughes LAN Systems Inc., Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

[Case 93e] J.D. Case, K. McCloghrie, M.T. Rose et S. Waldbusser. *"An Introduction to the Simple Management Protocol"*. H.G. Hegering et Y. Yemini, éditeurs, *Integrated Network Management, III (C-12)*, pages 261–272. Elsiever Science Publishers B.V. (North-Holland), 1993. Proc. of the IFIP TC6/WG6.6 3rd. Int. Symp. on Integrated Network Management, San francisco, Ca., 18-23 April, 1993.

[CCITT-Z-100-92 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Specification and Description Language (SDL) 1992"*, Norme Internationale, CCITT-Z-100-92, Janvier 1992.

[CCITT.M.3100 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Generic Network Information Model"*, Draft International Standard, CCITT.M.3100, Janvier 1992.

[CCITT.X.701 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Technologies de l'Information - Interconnexion de Systèmes Ouverts: Aperçu Général de la Gestion des Systèmes"*, Norme Internationale, CCITT.X.701, Janvier 1992, [ISO-10040 92].

[CCITT.X.720 92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Technologies de l'Information - Interconnexion de Systèmes Ouverts - Structure des Informations de Gestion: Modèle d'Information de Gestion"*, Norme Internationale, CCITT.X.720, Janvier 1992, [ISO-10165.1 92].

[CCITT.Z.100-104 84] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"Specification and Description Language (SDL)"*, Norme Internationale, CCITT.Z.100-104, January 1984.

[CCITT.Z.100-A 88] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *"SDL User gudelines"*, Norme Internationale, CCITT.Z.100-A, Janvier 1988, "Appendix to Z.100".

[Clemm 93a] A. Clemm. *"Incorporating Relationships into OSI Management Information"*. 1993. 2nd IEEE Network Management and Control Workshop, Tarrytown, NY, 21-23 September 1993.

[Clemm 93b] A. Clemm et O. Festor. *"Behavior, Documentation and Konwledge: An Approach for the Treatment of OSI-Behavior"*. *Fourth International Workshop on Distributed Systems: Operations and Management*, 1993. October 5-6, 1993, Long Branch, New-Jersey, USA.

[Collins 89] W. Collins. *"OSI Management Service Elements, Protocols and Application Layer Structure"*. B. Meandzija et J. Westcost, éditeurs, *Integrated Network Management, I*, pages 119–131. Elsiever Science Publishers B.V. (North-Holland), 1989. Proc. of the IFIP TC6/WG6.6 1st. Int. Symp. on Integrated Network Management, 1-5 April, 1989.

[Davin 92] J. Davin, J. Galvin et K. McCloghrie, *"SNMP Security Protocol"*, RFC1352, MIT Laboratory for Computer Science, Trusted Information Systems Inc., Hughes LAN Systems Inc., MIT Laboratory for Computer Science, July 1992.

[Desai 93] S. Desai, S. Joglekar et P.B. Westerfield. *"An Object Management and Communications Platform for Building TMN Compliant Operations Systems"*. 1993. Fourth International Workshop on Distributed Systems: Operations and Management, October 5-5, 1993, Long Branch, New-Jersey, USA.

[Dijkerman 89] E.M. Dijkerman, W.H.P. van Hulzen et P.A.J. Tilanus. "Object Oriented Specification Style in LOTOS". ri no. 917-RNL-89, PTT Research, "Sint Paulusstraat 4, P.O. Box 421, 2260 AK Leidshendam, The Netherlands", 1989.

[Dossogne 93] F. Dossogne et M.P. Dupont. *"A software architecture for Management Information Model definition, implementation and validation"*. H.G. Hegering et Y. Yemini, éditeurs, *Integrated Network Management, III (C-12)*, pages 593–604. Elsiever Science Publishers B.V. (North-Holland), 1993. Proc. of the IFIP TC6/WG6.6 3rd. Int. Symp. on Integrated Network Management, San francisco, Ca., 18-23 April, 1993.

[Duke 90] R. Duke, G. Rose et A. Lee. *"Object-oriented protocol specification"*. L. Logrippo, R.L. Probert et Ural H., éditeurs, *Proc. 10th Int. Conf. on Protocol Specification, Testing and Verification*, pages 323–339, 1990.

[Eschebach 91] Wilko Eschebach. *"Interpretative Ausfuehrung kommunizierender Regelsysteme"*. 1991. Master Thesis, University of Kaiserslautern, Germany, 1991.

[Festor 92] O. Festor, J.M. Schneider et G. Zörntlein. "Formal Description of the OSI Distributed Transaction Processing Protocol". Technical Report, IBM European Networking Center, Heidelberg, Germany, 1992.

[Festor 93] O. Festor et G. Zoerntlein. *"Formal Description of Managed Object Behavior - A Rule Based Approach"*. H.G. HEGERING et Y. YEMINI, éditeurs, *Integrated Network Management, III (C-12)*, pages 45–58. Elsiever Science Publishers B.V. (North-Holland), 1993. Proceedings of the IFIP TC6/WG6.6 Third International Symposium on Integrated Network Management, San francisco, California, USA, 18-23 April, 1993.

[Festor 94] O. Festor. *"OSI Managed Objects Development with LOBSTERS"*. 1994. Fifth International Workshop on Distributed Systems: Operations and Management, 12-16 Septembre 1994, Toulouse, France.

[Fischer 93] J. Fischer et R. Schroeder. *"Combined Specification Using SDL and ASN.1"*. O. Faergemand et A. Sarma, éditeurs, *SDL '93: Using Objects*, pages 293–304. Elsiever Science Publishers B.V., 1993.

[Fisher 93] A. Fisher, M. Herpers, D. Holden et S. Sievert. *"The DOMAINS Management Language"*. H.G. Hegering et Y. Yemini, éditeurs, *Integrated Network Management, III (C-12)*, pages 181–192. Elsiever Science Publishers B.V. (North-Holland), 1993. Proc. IFIP TC6/WG6.6 3rd. Int. Symp. on Integrated Network Management, San francisco, Ca., 18-23 April, 1993.

[Frot 93] J.P. Frot, H. Lecorguillé, J. Lefranc et D. Orain. *"CRUSADE: un environnement de développement de protocoles"*. 1993. Industrial Project Report, Ecole Supérieure d'Informatique et Applications de Lorraine, 1993.

[Galvin 93] J. Galvin et K. McCloghrie, *"Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2)"*, RFC1445, Trusted Information Systems, Hughes LAN Systems Inc., April 1993.

[Hegering 93] Heinz-Gerd Hegering et Sebastian Abeck. *"Integriertes Netz- und Systemmanagement"*. Addison-Wesley, 1993.

[ISO-10026.2 92] International Organization for Standardization (ISO), *"Distributed Transaction Processing - Part 1: OSI TP Service"*, Final Text, ISO-10026.2, April 1992.

[ISO-10040 92] International Organization for Standardization (ISO), *"System Management Overview"*, Interim Final Text, ISO-10040, January 1992.

[ISO-10164.1 92] International Organization for Standardization (ISO), *"System Management - Part 1: Object Management Function"*, International Standard, ISO-10164.1, January 1992.

[ISO-10164.2 92] International Organization for Standardization (ISO), *"System Management - Part 2: State Management Function"*, International Standard, ISO-10164.2, January 1992.

[ISO-10164.7 92] International Organization for Standardization (ISO), *"System Management - Part 7: Security Alarm Reporting Function"*, International Standard, ISO-10164.7, January 1992.

[ISO-10165.1 92] International Organization for Standardization (ISO), *"Structure of Management Information - Part 1: Management Information Model"*, International Standard, ISO-10165.1, January 1992.

[ISO-10165.2 92] International Organization for Standardization (ISO), *"Structure of Management Information - Part 2: Definition of Management Information"*, International Standard, ISO-10165.2, January 1992.

[ISO-10165.4 92] International Organization for Standardization (ISO), *"Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects"*, International Standard, ISO-10165.4, January 1992.

[ISO-10165.5 92] International Organization for Standardization (ISO), *"Structure of Management Information - Part 5: Generic Management Information"*, International Standard, ISO-10165.5, January 1992.

[ISO-7498.4 89] International Organization for Standardization (ISO), *"Basic Reference Model - Part 4: Management framework"*, International Standard, ISO-7498.4, November 1989.

[ISO-8649 88] International Organization for Standardization (ISO), *"Service definition for the Association Control Service Element"*, International Standard, ISO-8649, December 1988.

[ISO-8807 87] International Organization for Standardization (ISO), *"LOTOS: A Formal Description Technique based on the Temporal Ordering of Observable Behaviour"*, Draft International Standard, ISO-8807, June 1987.

[ISO-8824 90] International Organization for Standardization (ISO), *"Specification of Abstract Syntax Notation Number One (ASN.1)"*, International Standard, ISO-8824, December 1990.

[ISO-8825 87] International Organization for Standardization (ISO), *"Specification of Basic Encoding Rule for Abstract Syntax Notation Number One (ASN.1)"*, International Standard, ISO-8825, December 1987.

[ISO-9072.1 89] International Organization for Standardization (ISO), *"Remote Operations - Part 1: Model, notation and service definition"*, International Standard, ISO-9072.1, November 1989.

[ISO-9074 87] International Organization for Standardization (ISO), *"ESTELLE: A Formal Description Technique based on Extended State Transition Model"*, Draft International Standard, ISO-9074, June 1987.

[ISO-9545 89] International Organization for Standardization (ISO), *"Application Layer Structure"*, Draft International Standard, ISO-9545, March 1989.

[ISO-9595 91] International Organization for Standardization (ISO), *"Common Management Information Service Definition"*, International Standard, ISO-9595, June 1991.

[ISO-9596 91] International Organization for Standardization (ISO), *"Common Management Information Protocol Specification"*, International Standard, ISO-9596, June 1991.

[ISO-9646.3 91] International Organization for Standardization (ISO), *"OSI Conformance Testing and Framework - Part 3: The Tree andTabular Combined Notation (TTCN)"*, International Standard, ISO-9646.3, June 1991.

[ISO-9804 90] International Organization for Standardization (ISO), *"Commitment Concurrency and Recovery (CCR): Service Definition"*, Final Text, ISO-9804, February 1990.

[Jones 90] C.B. Jones. *"Systematic Software Development Using VDM (second edition)"*. Prentice Hall, 1990.

[Kaboré 94a] P. Kaboré et A. Schaff. *Du test des systèmes d'administration de réseaux. Submitted to CARI'94*, Ouagadougou, Burkina-Faso, October 1994.

[Kaboré 94b] P. Kaboré et A. Schaff. *Objets de gestion de réseaux: une approche de test de conformité. JISI'94*, Tunis, Tunisie, May 1994.

[Karner 93] G. Karner. *"Semantic Integration of ASN.1 into SDL"*. O. Faergemand et A. Sarma, éditeurs, *SDL '93: Using Objects*, pages 305–315. Elsiever Science Publishers B.V., 1993.

[Kilov 92] H. Kilov. *"Understand→Specify→Reuse: Precise Specification of Behaviour and Relationships."*. 1992. IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, Munich, October 1992.

[Kouyzer 92] A.J. Kouyzer et A.K. van den Boogaart. *"The LOTOS framework for OSI Systems Management"*. I. Krishnan et W. Zimmer, éditeurs, *Integrated Network Management, II*, pages 147–156. Elsiever Science Publishers B.V. (North-Holland), 1992. Proc. of the IFIP TC6/WG6.6 2nd. Int. Symp. on Integrated Network Management, Crystal City, Washington, D.C., 1-5 April, 1991.

[Lecerf 93] C. Lecerf et D. Chomel. *"Les Normes de Gestion de Réseau à l'ISO"*. Masson, 1993.

[Mackert 87] L.F. Mackert et I.B. Neumeier-Mackert. *"Communicating Rule Systems"*. pages 77–88, 1987. Proc. 7th. Int. Symp. on Protocol Specification, Testing and Verification, H. Rudin, C.H. West (editors), North-Holland 1987.

[Mangold 90] T. Mangold. *"Formal Specification of the X.25 Packet Layer in CRS"*. 1990. D.E.S.S. Thesis, University of Nancy, France, 1990.

[Marshall 92] L.S. Marshall et Simon L. *"Using VDM to Specifiy Managed Object Relationships"*. 1992. FORTE'92, Lannion, Fr., 13-16 Oct, 1992.

[Mazaher 93] S. Mazaher et B. Moller-Pedersen. *"On the use of SDL-92 for the Specification of Behaviour in OSI Network Management Objects"*. O. Faergemand et A. Sarma, éditeurs, *SDL '93: Using Objects*, pages 317–331. Elsiever Science Publishers B.V., 1993.

[McCloghrie 90] K. McCloghrie et M Rose, *"Management Information Base for Network Management of TCP/IP-based Internets"*, RFC1156, Hughes LAN Systems Inc., Performance Systems International, May 1990.

[McCloghrie 91] K. McCloghrie et M Rose, *"Management Information Base for Network Management of TCP/IP-based Internets: MIB-II"*, RFC1213, Hughes LAN Systems Inc., Performance Systems International, March 1991.

[Meyer 88] Bertrand Meyer. *"Object-oriented Software Construction"*. Prentice Hall International Series in Computer Science, 1988.

[Meyer 92] B. Meyer. *"Applying Design by Contract"*. *Computer Magazine*, pages 40–50, 1992. 0018-9162/92/1000-0040 IEEE.

[Moller-Pedersen 90] B. Moller-Pedersen. *"Introduction to Object-Oriented SDL"*. J. Quemada, J. Manas et E. Vazquez, éditeurs, *Tutorials of the 3rd Int. Conf. on Formal Description Techniques*, pages 7–66, 1990.

[Monnery 89] J. Monnery. *"A Rule-based Executable Specification of the X.25 Network layer"*. 1989. Master Thesis, Ecole Nationale Supérieure des Télécommunications de Bretagne, 1989.

[N109 89] International Organization for Standardization (ISO), *"Modelling techniques and their use in ODP"*, Working Draft, ISO-JTC1 SC21 WG7 N109, April 1989.

[Nakai 92] S. Nakai et M.L. Liu. *"Concurrency Control for Network Management Transactions based on OSI's Systems Management and Transaction Protocols"*. 1992. IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, Munich, October 1992.

[Neumeier-Mackert 88a] I.B. Neumeier-Mackert. *"Modellierung und Implementierung kommunizierender Systeme mit hilfe von Regeln"*. 1988. Doctoral Dissertation, University of Erlangen-Nürnberg (Germany).

[Neumeier-Mackert 88b] I.B. Neumeier-Mackert, L.F. Mackert et A. Fleischmann. *"A knowledge based Protocol Engineering Environment"*. 1988. Proc. 1st Int. Conf. on Networking in Open Systems, Washington, 1988.

[NMF 92] Network Management Forum, *"Forum Library Volume 1: OMNIpoint 1 Definitions Forum 001"*, NMF, July 1992.

[Orain 93] D. Orain. *"A New ASN.1 Compiler for the CRUSADE Environment"*. 1993. Master Thesis, Ecole Supérieure d'Informatique et Applications de Lorraine, 1993.

[Pfeiler 93] M. Pfeiler et M. Wittig. *"Distributed Modelling and Administration of OSI Management Schema Knowledge"*. 1993. GMD FOKUS, Hardenbergplatz 2, 1000 Berlin 12, Germany.

[Postel 80] D. Postel, *"User Datagram Protocol"*, RFC768, September 1980.

[Preuss 93] Thomas Preuss. *"Management von virtuellen privaten Netzen"*. 1993. Master Thesis, University of Magdeburg,1993.

[Rose 89] Marshall T. Rose. *"The Open Book: A practical perspective on OSI"*. Prentice Hall, 1989.

[Rose 90] M. Rose et K. McCloghrie, *"Structure and Identification of Management Information for TCP/IP-based Internets"*, RFC1155, Performance Systems International,Hughes LAN Systems Inc., May 1990.

[Rose 91] M Rose et K. McCloghrie, *"Concise MIB Definitions"*, RFC1212, Performance Systems International,Hughes LAN Systems Inc., March 1991.

[Rudkin 91] H. Rudkin. *"On the use of Z to specify OSI Managed-Objects behaviour"*. 1991. Proc. Int. Workshop on Open Distributed Processing, 8-11 October, 1991, Berlin, Germany.

[Rumbaugh 92] R. Rumbaugh. *"Object-oriented modelling techniques"*. Prentice Hall, 1992.

[Sample 93] M. Sample. *"SNACC 1.1: A High Performance ASN.1 to C/C++ Compiler"*. 1993. Master Thesis, University of British Columbia.

[Scheere 91] U. Scheere. *"Formale Spezification des OSI-TP-Standards mit der regelbasierten Technik CRS"*. 1991. Master Thesis, University of Karlsruhe (Germany), 1991.

[Schneider 90a] J.M. Schneider, L.F. Mackert, G. Zoerntlein, R.J. Velthuys et U. Baer. *"An integrated environment for Developing Communicating Rule Systems"*. 1990. in Computer networks and ISDN systems, 1990.

[Schneider 90b] J.M. Schneider, H. Mueller, L.F. Mackert, G. Zoerntlein et R.J. Velthuys. *"Transition System Semantics for Communicating Rule Systems"*. 1990. Technical Report No. 43.9005, IBM European Networking Center, Vangerowstrasse 18, D-6900 Heidelberg, Germany, 1990.

[Schneider 92] J.M. Schneider. *"Protocol Engineering: A Rule-based Approach"*. Vieweg, 1992.

[Schneider 93a] J.M. Schneider et W. Donnely. *"Co-operative Management of Integrated Broadband Communication Services over the CCITT X Interface"*. 1993. Fourth International Workshop on Distributed Systems: Operations and Management, October 5-5, 1993, Long Branch, New-Jersey, USA.

[Schneider 93b] J.M. Schneider, T. Preuss et P.S. Nielsen. *"Management of Virtual Private Networks for Integrated Broadband Communication"*. 1993. Proc. ACM SIGCOMM '93, San Francisco, CA, september 1993.

[Schott 92] B. Schott, A. Clemm et U. Hollberg. *"An ISO/OSI based Approach for Modeling Heterogeneous Networks"*. 1992. 4th. IFIP Int. Conf. on Information Networks and Data Communication, Espoo, Finland, march, 1992.

[Sibilla 93] M. Sibilla, D. Marquie et Y. Raynaud. *"Management Information Base: Guidelines leading from Generic Specification of Managed Object Relationships to Consistent Implementation"*. *Fourth International Workshop on Distributed Systems: Operations and Management*, 1993. October 5-7, 1993, Long Branch, New-Jersey, USA.

[Sijelmassi 89] R. Sijelmassi et P. Gaudette. *"An Object-Oriented Model for ESTELLE Formal Description Technique"*. K.J. Turner, éditeur, *Proc. 1st Int. Conf. on Formal Description Techniques for Communication Protocols and Distributed Systems*, pages 91–105. North-Holland, 1989.

[Simon 92] L. Simon et S. Marshall. *"Using VDM to Specifiy OSI Managed Objects"*. 1992. Bell-Northern Research Ltd., P.O. Box 3511 Station C, Ottawa, Ontario, Canada.

[Spivey 89] J.M. Spivey. *"The Z Notation"*. Prentice Hall, 1989.

[Sylor 93] M. Sylor. *"Junction Objects: A solution to a problem in naming and locating OSI managed Objects"*. H.G. Hegering et Y. Yemini, éditeurs, *Integrated Network Management, III (C-12)*, pages 161–173. Elsiever Science Publishers B.V. (North-Holland), 1993. Proc. IFIP TC6/WG6.6 3rd. Int. Symp. on Integrated Network Management, San Francisco, Ca., 18-23 April, 1993.

[Thomas 92] M. Thomas. *"An ASN.1 to LOTOS translator"*. 1992. FORTE'92, Lannion, Fr., 13-16 October 1992.

[Turner 93] K.J. Turner. *"Using Formal Description Techniques"*. Wiley, 1993.

[Velthuys 92a] R.J. Velthuys. *"Conformance testing based on formal system models"*. 1992. Ph.D. Dissertation, University of Bern (Switzerland), 1992.

[Velthuys 92b] R.J. Velthuys, L.F. Mackert, J.M. Schneider et G. Zoerntlein. *"Structuring mechanisms for the Formal CRS"*. 1992. Technical Report, IBM European Networking Center, Heidelberg (Germany), 1992.

[Waldbusser 91] S. Waldbusser, *"Remote Network Monitoring Management Information Base"*, RFC1271, Carnegie Mellon University, November 1991.

[Wester 93] J.J. Wester et M.L.J. van Everdingen. *"Distributed Management Transactions"*. 1993. Fourth International Workshop on Distributed Systems: Operations and Management, October 5-5, 1993, Long Branch, New-Jersey, USA.

[Yao 93] M. Yao et G. Bochmann. *"Testing for a conformance relation based on acceptance"*. 1993. Proc. of the TAPSOFT'93, 1993.

Appendice A
Les formulaires additionnels de GDMO

Dans le chapitre 4 de la première partie, nous avons présenté les formulaires de GDMO qui sont à la base des extensions proposées dans cette thèse. Il est cependant nécessaire de présenter les autres formulaires de la notation et ceci pour plusieurs raisons. La première est la volonté d'être exhaustif dans le présentation de GDMO. La seconde est que ces formulaires sont retenus tels quels dans le formalisme LOBSTERS et qu'il convient donc à ce titre de les décrire de manière détaillée.

A.1 Les formulaires inchangés

Les formulaires de GDMO que nous n'avons pas encore présentés et qui sont repris sans modifications dans notre formalisme sont au nombre de trois. Il s'agit du formulaire d'attribut, de groupe d'attributs et de paramètre. Tout comme les autres formulaires présentés précédemment, nous allons les décrire de façon détaillée. Pour les illuster, nous utiliserons des spécifications existantes normalisées.

Chaque section de cet appendice décrit l'un de ces formulaires.

A.2 Les attributs

Les attributs représentent l'espace des données d'un objet géré. Un attribut est défini par une étiquette, un type et par les opérations de comparaison qu'il supporte. Les opérations d'accès à l'attribut étant spécifiques au module dans lequel celui-ci est intégré, celles-ci sont spécifiées dans la liste des propriétés de l'attribut lors de son intégration dans le module.

A.2.1 Le formulaire d'attribut

Le formulaire d'attributs de GDMO permet de définir un attribut de deux manières différentes (voir figure A.1). En effet, un attribut peut être défini comme extension d'un attribut existant `<attribute-label>` via la clause **DERIVED FROM**, ou comme attribut de type indépendant via la clause **WITH ATTRIBUTE SYNTAX**. Dans le premier cas, le type est celui de l'attribut dont on hérite et dans le second cas, le type est obtenu par référence à une structure ASN.1.

Le type de l'attribut influence les opérations de comparaison autorisées sur toute instance de cet attribut. La clause **MATCHES FOR** permet de fixer la liste de ces opérations. Elles sont au nombre de cinq. L'option **EQUALITY** indique que la valeur de l'attribut peut faire l'objet d'un test d'égalité avec une valeur d'un type compatible. L'option **ORDERING** indique l'existence d'une relation d'ordre sur le type de l'attribut. Celui-ci peut donc être testé sur l'ordre par rapport à une valeur donnée. L'option

```
<attribute-label> ATTRIBUTE
  derived-or-with-syntax-choice ;
 [ MATCHES FOR  qualifier [, qualifier]* ;
 ]
 [ BEHAVIOUR <behaviour-label> [,<behaviour-label>]* ;
 ]
 [ PARAMETERS <parameter-label> [,<parameter-label>]* ;
 ]
[ REGISTERED AS <object-identifier>] ;

 derived-or-with-syntax-choice -> DERIVED FROM <attribute-label>
                                | WITH ATTRIBUTE SYNTAX reference-type

 qualifier -> EQUALITY | ORDERING | SUBSTRINGS
            | SET-COMPARISON | SET-INTERSECTION
```

FIG. A.1 - . *Le formulaire de spécification d'attribut*

SUBSTRINGS est applicable à des attributs de type chaînes (caractères, bits, etc..). Elle indique la possibilité de rechercher des motifs dans une instance de l'attribut. Les options **SET-COMPARISON** et **SET-INTERSECTION** s'appliquent à des attributs de type ensemble et indiquent respectivement la possibilité de comparer la valeur de l'attribut à celle d'un ensemble donné et de pouvoir appliquer des opérations d'intersection sur une instance de cet attribut.

Le champ comportement **BEHAVIOUR** contient une définition de la sémantique de l'attribut. Les paramètres associés à un attribut à l'aide de la clause **PARAMETERS** sont utilisés pour représenter des informations d'erreurs de traitement relatives à l'attribut dans le cas d'une notification d'erreurs.

Bien que le formulaire GDMO d'attribut comporte une clause de définition de comportement, celle-ci n'est pas exploitée en LOBSTERS et ne comporte donc pas de mécanismes formels.

A.2.2 Un exemple

L'attribut de statut de disponibilité `availabilityStatus` est normalisé par l'ISO dans le cadre de la fonction de gestion des états. Sa définition en GDMO est donnée dans la figure A.2. Il est de type ensemble et représente à tout moment l'état de disponibilité de la ressource modélisée par l'objet dans lequel il est contenu. Si l'ensemble des valeurs de l'attribut est vide, cela signifie que la ressource est disponible. Si par contre l'attribut contient un ou plusieurs éléments, cela signifie que la ressource n'est pas disponible. Dans ce dernier cas, les valeurs des éléments de l'ensemble indiquent les raisons de l'indisponibilité de la ressource.

L'attribut `availabilityStatus` est défini comme attribut indépendant. Il n'est dérivé d'aucun attribut existant. Sa syntaxe est définie dans le module `ASN.1 Attribute-ASN1Module` et les opérations de comparaison autorisées sont le test d'égalité **EQUALITY**, la comparaison **SET-COMPARISON** et l'intersection **SET-INTERSECTION** avec d'autres ensembles de même type. Il est utilisé dans de nombreuses définitions d'objets gérés.

```
availabilityStatus ATTRIBUTE
  WITH ATTRIBUTE SYNTAX Attribute-ASN1Module.AvailabilityStatus;
  MATCHES FOR EQUALITY, SET-COMPARISON, SET-INTERSECTION;
 REGISTERED AS { joint-iso-ccitt ms(9) smi(3) part2(2) attribute(7) 33};
```

FIG. A.2 - . *La spécification de l'attribut de disponibilité de ressource*

A.3 Les groupes d'attributs

Définir un groupe d'attributs permet de mettre en relation des attributs dont l'évolution est liée au sein d'une classe d'objets gérés. Une telle définition autorise également la récupération et l'affectation au travers d'une seule opération de l'ensemble des valeurs des différents composants du groupe.

A.3.1 Le formulaire

Un groupe d'attributs (voir figure A.3) est défini par une étiquette <group-label> et un identificateur d'objet dans l'arbre d'identification via la clause **REGISTERED AS**. Sa définition comporte la liste des attributs qu'il regroupe **GROUP ELEMENTS** ainsi qu'une description textuelle de la sémantique du groupe dans le champ **DESCRIPTION**.

```
<group-label> ATTRIBUTE GROUP
[ GROUP ELEMENTS <attribute-label> [,<attribute-label>]*;
]
[ FIXED;]
[ DESCRIPTION delimited-string;]
 REGISTERED AS <object-identifier>;
```

FIG. A.3 - . *Le formulaire de description de groupes*

Le champ **FIXED** est optionnel. Sa présence indique que le groupe n'est pas extensible par adjonction de nouveaux attributs lors de son inclusion dans un module. Lorsque ce champ n'est pas présent, le groupe peut être étendu.

A.3.2 Un exemple

Le groupe `state` spécifié dans la figure A.4 est normalisé par l'ISO dans la fonction de gestion des états. Il a pour but de lier l'ensemble des attributs d'état présents dans un module. Comme tous les attributs d'état existants ne sont pas forcément implantés dans tous les modules, le groupe `state` est vide par défaut. Les éléments qui le composent sont spécifiés dans le module dans lequel le groupe est instancié.

Comme le module est extensible, la clause **FIXED** n'est pas présente dans la définition de celui-ci. Un exemple d'instanciation de ce groupe est donné dans la définition du module relatif à un système informatique dans la section 4.3.2.

```
state     ATTRIBUTE GROUP
          DESCRIPTION "This is defined as an empty attribute group.
The elements of this group are composed of state attributes in the
managed object. The state attributes may include those specified in
CCITT Rec. X.731 | ISO/IEC 10164-2 and others that are specific
to the managed object class.";

 REGISTERED AS    { joint-iso-ccitt ms(9) smi(3) part2(2) attributeGroup(8) 1};
```

FIG. A.4 - . *La définition du groupe d'attributs d'état*

A.4 Les paramètres

Les paramètres sont principalement utilisés pour la spécification d'actions ou de notifications. Ils peuvent également être utilisés dans la spécification d'attributs auquel cas ils servent à représenter l'information associée à une erreur de traitement de l'attribut.

A.4.1 Le formulaire

Le formulaire de paramètre permet de définir pour chaque paramètre: sa syntaxe, le contexte dans lequel il est utilisé, ainsi que le comportement associé (voir figure A.5). La clause **CONTEXT** définit le champ de la PDU CMIP qui va contenir la valeur du paramètre. Lorsque le paramètre est associé à une action, les contextes possibles sont **ACTION-INFO** (le paramètre est utilisé lors de l'appel à l'action), **ACTION-REPLY** (le paramètre est utilisé dans une réponse d'action), `type-reference.identifier` (la valeur du paramètre est transmise dans le champ **identifier** de la partie de type `type-reference` de la PDU CMIP) et **SPECIFIC-ERROR** (le champ d'erreur spécifique dans CMIS/P). Dans le cas d'une notification, les valeurs possibles du contexte sont: **EVENT-INFO** utilisée dans une émission de notification, **EVENT-REPLY** utilisée dans une réponse de notification, `type-reference.identifier` dont la valeur est reportée sur un champ spécifique ou **SPECIFIC-ERROR** qui est un champ d'une notification d'erreurs de traitement. Pour les paramètres associés à des attributs, des fonctions de création ou de destruction, le contexte ne peut être que **SPECIFIC ERROR** (paramètre d'information d'erreur de traitement).

La syntaxe d'un paramètre est soit directement un type ASN.1, soit obtenue à partir de celle d'un attribut. Dans le premier cas **WITH SYNTAX**, la syntaxe *syntax-or-attribute-choice*) est obtenue à l'aide d'une référence à un type ASN.1 <`reference-type`>. Dans le second cas **ATTRIBUTE**, la syntaxe est identique à celle de l'attribut référencé <`attribute-label`>. Le champ **BEHAVIOUR** contient soit une référence de comportement<`behaviour-label`>, soit directement la description textuelle de la sémantique du paramètre.

A.4.2 Un exemple

La figure A.6 comporte la spécification d'un paramètre. Le paramètre `transportDisconnectCause`, spécifié dans cette figure, est utilisé dans la notification d'effacement d'objet géré modélisant une connexion de transport.

```
<parameter-label> PARAMETER
   CONTEXT context-type;
   syntax-or-attribute-choice;
  [ BEHAVIOUR <behaviour-label> [,<behaviour-label>]*;
  ]
[ REGISTERED AS <object-identifier>];

 context-type -> context-keyword | ACTION-INFO | ACTION-REPLY
               | EVENT-INFO | EVENT-REPLY | SPECIFIC-ERROR

 context-keyword -> type-reference.<identifier>

 syntax-or-attribute-choice -> WITH SYNTAX type-reference
                             | ATTRIBUTE <attribute-label>
```

FIG. A.5 - . *Le formulaire de paramètre*

```
transportDisconnectCause PARAMETER
   CONTEXT EVENT-INFO;
   WITH SYNTAX SYNTAX-1.Cause;
   BEHAVIOUR causeBehaviour;
REGISTERED AS { forum-parameter 1};
```

FIG. A.6 - . *Un exemple de spécification de paramètre*

Ce paramètre comporte des informations sur la cause de rupture d'une connexion de transport. Il est utilisé comme information dans le cadre d'une notification (contexte **EVENT-INFO**). Son type ASN.1 (Cause) comporte deux champs. Le premier indique la source de la rupture (inconnue, utilisateur, fournisseur) et le second la cause (inconnue, temps d'inactivité excessif, nombre de retransmissions excessives).

A.5 Résumé

Dans cette annexe, nous avons complété la description des formulaires de spécification du langage GDMO. Pour ce faire et en suivant le même principe que pour les formulaires modifiés, nous avons présenté chacun d'eux en deux étapes: les composants du formulaire et un exemple d'utilisation.

Ces formulaires sont tous intégrés dans LOBSTERS et ne comportent pas de description formelle du comportement.

Appendice B
L'exemple du système informatique

Dans les chapitres 4 et 5 de la seconde partie, nous avons illustré les formulaires et les algorithmes que nous proposons sur une version modifiée de l'objet géré système informatique. Cette définition d'objet, issue des normes, a été modifiée afin que tous les mécanismes de LOBSTERS puissent y être illustrés. Dans cet appendice, nous allons donner la spécification LOBSTERS complète de l'exemple, présenter les résultats de l'application de l'algorithme de collecte de comportement ainsi que l'extension pour le calcul par héritage.

B.1 Le système

L'exemple que nous avons utilisé dans les chapitres précédents représente un système informatique défini en deux étapes. Le premier système faisant office de super-classe est un objet qui modélise des fonctionnalités de base du système informatique qui sont la capacité de visualiser son état opérationnel, d'être activé au travers de l'interface de gestion, d'être désactivé par la ressource et d'émettre des notifications de changement d'état (voir B.1).

Pour illustrer le mécanisme d'héritage et la manipulation des modules optionnels, nous avons conçu un système informatique étendu qui hérite des fonctionnalités du système de base et qui en offre de nouvelles grâce à un module obligatoire et un module optionnel. Les fonctionnalités de ce système étendu sont:

- au niveau du module obligatoire:
  - un attribut de gestion administrative (`administrativeState`),
  - des actions de gestion administrative (blocage, déblocage, fermeture).
- au niveau du module optionnel:
  - un attribut de statut d'utilisation (`usageState`),
  - un attribut de capacité maximale d'utilisateurs (`capacity`) ainsi qu'un attribut comportant le nombre actuel d'utilisateurs du système (`users`),
  - des actions de manipulation de la capacité (augmentation, diminution),
  - des actions de connexion et de déconnexion (login, logout)

Les deux objets accèdent à trois interfaces distinctes qui sont:

- l'interface de gestion (`Management`) au travers de laquelle les actions de gestion sont invoquées,

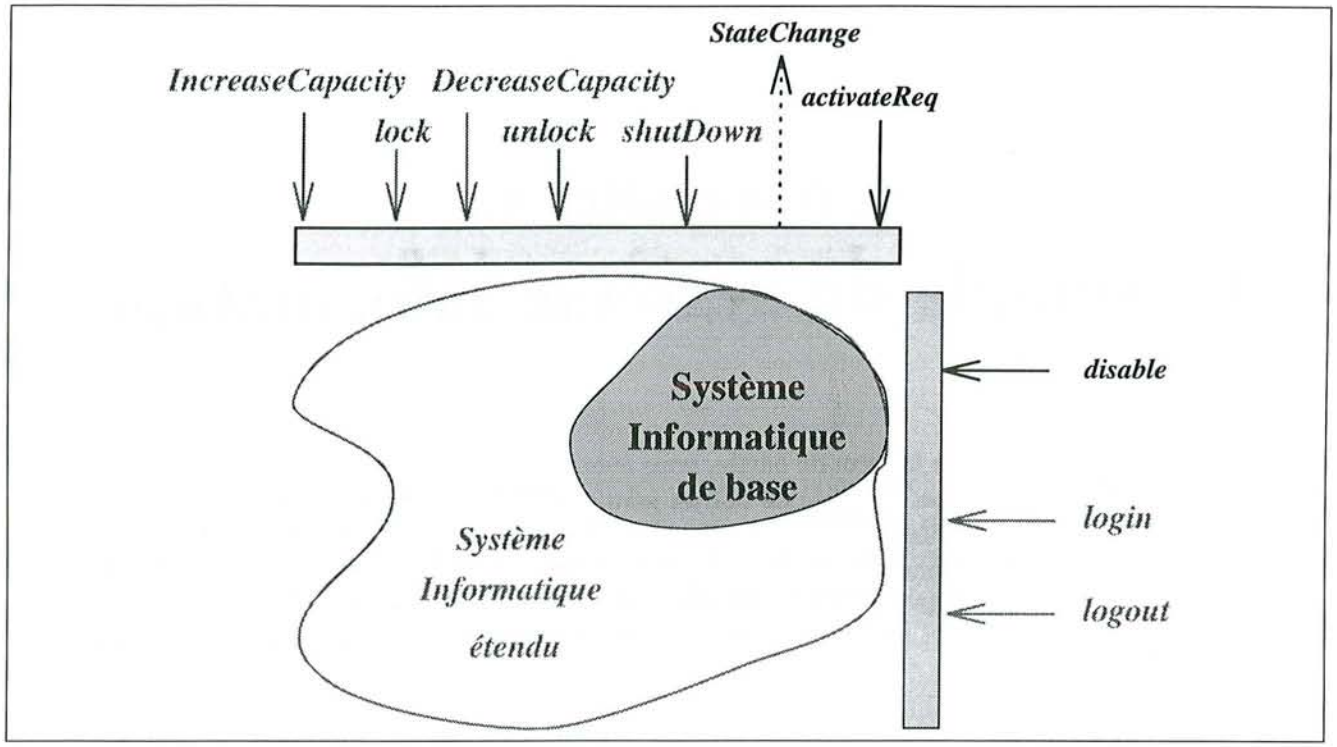

FIG. B.1 - . *L'objet géré système informatique*

- l'interface de notifications (`Notification`), interface asynchrone au travers de laquelle les notifications de changement d'état sont émises,
- un interface externe (`external`) au travers de laquelle l'objet géré est informé du statut de la ressource qu'il modélise et au travers de laquelle les utilisateurs se connectent ou se déconnectent du système.

B.2 La spécification LOBSTERS

Dans cette section, nous présentons les formulaires LOBSTERS qui spécifient les deux objets gérés ainsi que leurs interfaces.

B.2.1 Les interfaces de base

Le système de base accède à trois interfaces de base qui sont l'interface de gestion, de notifications et de l'environnement (ressource réelle ou autres objets gérés).

La spécification LOBSTERS de celles-ci est:

```
StandardManagementGate GATE
 DERIVED FROM SyncGate;
 DEFINITION " This interface allows the invocation of activate actions";
 DECLARATIONS
    response : ActivateActionReply;
```

```
 EVENTS
    activateReq();
    activateRsp(response);
;

StandardNotificationGate GATE
 DERIVED FROM AsyncGate;
 DEFINITION " This interface is a standard interface which
                allows state change notifications to be issued";
 DECLARATIONS
       stateSet : StateSet;
 EVENTS
       stateChange(stateSet);
;

StandardExternalGate GATE
 DERIVED FROM SyncGate;
 DEFINITION " Through this interface the Computer System Mo is informed as soon
                as the ressource becomes disabled";
 USAGE RESSOURCE;
 EVENTS
     disable();
;
```

B.2.2 Le système de base

Le système de base accède aux trois interfaces et possède un seul module. La spécification de l'objet géré est donnée ci-dessous.

```
BasicComputerSystem MANAGED OBJECT CLASS
 DERIVED FROM Top;
 CHARACTERIZED BY BasicComputerSystemPackage;
 INTERFACES
          Management   : StandardManagementGate,
          Notification : StandardNotificationGate,
          External     : StandardExternalGate;
 BEHAVIOUR BasicComputerSystemBehaviour;
 REGISTERED AS { lobsters-moc 1};
```

L'objet géré de base ne comporte qu'un seul module. Sa spécification est la suivante:

```
BasicComputerSystemPackage PACKAGE
   BEHAVIOUR BasicComputerSystemPackageBehaviour;
   ATTRIBUTES operationalState GET, INITIAL VALUE enabled;
   ACTIONS activate;
   NOTIFICATIONS stateChange;
;
```

Le comportement lié au module est spécifié ci-dessous:

```
BasicComputerSystemPackageBehaviour BEHAVIOUR
```

```
 DEFINED AS
"@
  DECLARATIONS

        stateSet  : StateSet;

  RULES

        activateSuccess;
          COND: operationalState = disabled;
          EFFECTS: operationalState = enabled;

        activateFailure;
          EFFECTS: operationalState = 'operationalState;

        disable;
          EFFECTS: operationalState = disabled;

  NOTIFICATIONS

        stateChange;
            TRIGGER: operationalState <> 'operationalState;
            ISSUANCE: MEMBER(stateSet,{"OperationalState ",operationalState});

 @ This is the behaviour associated with the mandatory package of the standard Computer
   system Managed Object";
;
```

L'action d'activation est définie dans le chapitre 4. Son comportement a été spécifié dans le chapitre 5 de la seconde partie. Le comportement associé à l'objet géré de base recense l'ensemble des occurences d'événements possibles aux portes de ce dernier.

```
BasicComputerSystemBehaviour BEHAVIOUR
 DEFINED AS
"@
   EVENTS
     activateSuccess;
       COND: Management.activateReq();
       EFFECTS: Management.activateRsp(response);

     activateFailure;
       COND: Management.activateReq();
       EFFECTS: Management.activateRsp(response);

     disable;
       COND: External.disable();

     stateChange;
       ISSUANCE: Notification.stateChange(stateSet);
 @ This is the basic computer system behaviour part";
;
```

Cette spécification définit entièrement l'objet géré modélisant un système informatique de base. Nous allons maintenant donner la spécification des interfaces et du système étendus.

B.2.3 Les interfaces étendues

Les deux interfaces étendues qui sont l'interface de gestion et l'interface externe sont spécifiées ci-dessous:

```
ExtendedManagementGate GATE
 DERIVED FROM StandardManagementGate;
 DEFINITION " This interface extends the standard managament interface";
 DECLARATIONS
      number : INTEGER;
 EVENTS
     lock();
     unlock();
     shutdown();
    increaseCapacity(number);
    decreaseCapacity(number);
;

ExtendedExternalGate GATE
 DERIVED FROM StandardExternalGateGate;
 DEFINITION " This interface extends the standard environment gate";
 EVENTS
     login();
     logout();
;
```

B.2.4 Le système étendu

Le système informatique étendu hérite des fonctionnalités du système informatique de base et se caractérise par deux modules supplémentaires qui sont le module de gestion administrative et le module de détection d'utilisation. Ce dernier est optionnel. La description LOBSTERS du système étendu est donnée ci-dessous.

```
ExtendedComputerSystem MANAGED OBJECT CLASS
 DERIVED FROM BasicComputerSystem
 CHARACTERIZED BY AdministrativeStatePackage;
 CONDITIONAL PACKAGES
   UsageStatePackage PRESENT IF "@SUPPORT@ This package is present if the system can detect us
 INTERFACES Management   : ExtendedManagementGate,
            External     : ExtendedExternalGate;
 BEHAVIOUR ExtendedComputerSystemBehaviour;
 REGISTERED AS { lobsters-moc 2};
```

L'objet géré étendu est composé de deux modules. La spécification de ces deux modules est:

```
AdministrativeStatePackage PACKAGE
  BEHAVIOUR AdministrativeStatePackageBehaviour;
```

```
  ATTRIBUTES administrativeState GET-REPLACE, INITIAL VALUE unlocked;
  ACTIONS lock,
          unlock,
         shutdown;
;

UsageStatePackage PACKAGE
  BEHAVIOUR UsageStatePackageBehaviour;
  ATTRIBUTES usageState GET, INITIAL VALUE idle,
                capacity   GET-REPLACE, INTIAL VALUE 20,
                users      GET, INITIAL VALUE 0;
  ACTIONS login,
             logout,
             increaseCapacity,
             decreaseCapacity;
;
```

Le comportement lié au module de gestion administrative est le suivant:

```
AdministrativeStatePackageBehaviour BEHAVIOUR
 DEFINED AS
"@
   RULES
       lock;
         COND: administrativeState <> locked;
         EFFECTS: administrativeState = locked;

       unlock;
         EFFECTS: administrativeState = unlocked;

       shutdown;
         COND: administrativeState = unlocked;
         EFFECTS: administrativeState = shuttingDown OR administrativeState = locked;
@ This is the behaviour associated with the administrativeState package";
;
```

Le comportement lié au module conditionnel de gestion de l'utilisation est:

```
UsageStatePackageBehaviour BEHAVIOUR
 DEFINED AS
"@
   DECLARATIONS
         n : INTEGER;

   RULES
       login;
         COND: usageState <>  busy;
         EFFECTS: users = users + 1
                  AND IF (users = capacity)
                      THEN usageState = busy
                      ELSE IF (users = 1)
```

```
                        THEN usageState = active
                    FI  FI;

    logout;
       EFFECTS: users = users -1
               AND IF (users = 0)
                   THEN usageState = idle
                   ELSE IF ('usageState = busy)
                        THEN usageState = active
                   FI   FI;

     increaseCapacity;
       EFFECTS: capacity = 'capacity + n
                AND IF (usageState = busy AND n > 0)
                    THEN usageState = active
                    FI;

     decreaseCapacity;
       COND: capacity -n >= users;
       EFFECTS: capacity = 'capacity -n
                AND IF (users = capacity)
                    THEN usageState = busy
                    FI;

 @ This is the behaviour associated with the usageState package";
;
```

Le comportement associé à l'objet géré étendu est donné ci-dessous.

```
ExtendedComputerSystemBehaviour BEHAVIOUR
 DEFINED AS
"@
  EVENTS
    lock;
       COND: Management.lock();

    unlock;
       COND: Management.unlock();

    shutdown;
       COND: Management.shutdown();

    login;
       COND: UsageStatePackage AND External.login();

    logout;
       COND: UsageStatePackage AND External.logout();

    increaseCapacity;
       COND: UsageStatePackage AND Management.increaseCapacity(n);

    decreaseCapacitySucess;
       COND: UsageStatePackage AND Management.decreaseCapacityReq(n);
```

RULES

```
lock;
    EFFECTS: IF ('UsageStatePackage)
             THEN usageState = idle AND users = 0
             FI;

shutdown;
   EFFECTS: IF ('UsageStatePackage)
            THEN IF usageState = idle
                 THEN administrativeState = locked
                 ELSE administrativeState = shuttingDown
                 FI
            ELSE administrativeState = locked
            FI;

login;
   COND: operationalState = enabled AND administrativeState = unlocked;

logout;
   EFFECTS: IF (usageState = idle AND 'administrativeState = shuttingDown)
            THEN administrativeState = locked
            FI;

activateSuccess;
   COND: administrativeState = locked;
   EFFECTS: administrativeState = unlocked;

activateFailure;
   EFFECTS: administrativeState = 'administrativeState;

disable;
   EFFECTS: IF ('UsageStatePackage)
            THEN usageState = idle AND users = 0
            FI;
```

NOTIFICATIONS

```
stateChange;
  TRIGGER: ( administrativeState <> 'administrativeState)
               OR ( UsageStatePackage AND (usageState <> 'usageState));
  ISSUANCE: IF ( administrativeState <> 'administrativeState)
               THEN MEMBER(stateSet,{"administrativeState ",administrativeState})
               FI
            AND IF ( UsageStatePackage AND (usageState <> 'usageState))
               THEN MEMBER(stateSet,{"usageState ",usageState})
               FI;
@ This is the behaviour associated to the extended computer system MO ";
```

Cette spécification caractérise entièrement le système étendu.

B.3 La collecte du comportement

Dans cette section, nous appliquons l'algorithme de collecte du comportement aux deux objets gérés présentés auparavant. Il s'agit de calculer les trois ensembles E, B et N pour chaque objet géré. Par la suite, ces ensembles vont nous servir à appliquer les règles de construction par héritage pour obtenir le comportement complet de la sous-classe.

B.3.1 Les ensembles sur le système de base

L'ensemble des événements du système de base est:

$E_{BasicComputerSystem}$ =

```
{
 (activateSuccess;
    COND: Management.activateReq();
    EFFECTS: Management.activateRsp(response);
 ),
 (activateFailure;
    COND: Management.activateReq();
    EFFECTS: Management.activateRsp(response);
 ),
 (disable;
    COND: External.disable();
 ),
 (stateChange;
   ISSUANCE: Notification.stateChange(stateSet);
 )
}
```

L'ensemble des règles de comportement du système de base est:

$B_{BasicComputerSystem}$ =

```
{
 (activateSuccess;
   COND: operationalState = disabled;
   EFFECTS: operationalState = enabled
            AND response = successful;
 ),
 (activateFailure;
   EFFECTS: operationalState = 'operationalState
            AND response = failureResponse;
 ),
 (disable;
   EFFECTS: operationalState = disabled;
 )
}
```

Finalement, l'ensemble des comportements de notifications du système informatique de base est:

$N_{BasicComputerSystem}$ =

```
{
 (stateChange;
     TRIGGER: operationalState <> 'operationalState;
     ISSUANCE: IF (operationalState <> 'operationalState)
                  THEN MEMBER(stateSet,{"OperationalState ",operationalState})
                  FI;
 )
}
```

B.3.2 Les ensembles sur le système étendu

Calculons maintenant ces trois ensembles sur le système étendu:

$E_{ExtendedComputerSystem}$ =

```
{
 (lock;
    COND: Management.lock();
 ),
 (unlock;
     COND: Management.unlock();
 ),
 (shutdown;
     COND: Management.shutdown();
 ),
 (login;
     COND: UsageStatePackage AND External.login();
 ),
 (logout;
     COND: UsageStatePackage AND External.logout();
 ),
 (increaseCapacity;
     COND: UsageStatePackage AND Management.increaseCapacity(n);
 ),
 (decreaseCapacitySucess;
     COND: UsageStatePackage AND Management.decreaseCapacityReq(n);
 )
}
```

L'ensemble des règles de comportement associé au système étendu est:

$B_{ExtendedComputerSystem}$ =

```
{
 (lock;
    COND: administrativeState <> locked;
    EFFECTS: administrativeState = locked
          AND IF ('UsageStatePackage)
               THEN usageState = idle AND users = 0
               FI;
 ),
 (unlock;
```

```
    EFFECTS: administrativeState = unlocked;
),
(shutdown;
    COND: administrativeState = unlocked;
    EFFECTS: ( administrativeState = shuttingDown OR administrativeState = locked )
       AND (IF ('UsageStatePackage)
             THEN IF usageState = idle
                  THEN administrativeState = locked
                  ELSE administrativeState = shuttingDown
                  FI
             ELSE administrativeState = locked
             FI );

),
(login;
    COND: (IF UsageStatePackage THEN usageState <> busy FI)
        AND ( operationalState = enabled AND administrativeState = unlocked );
    EFFECTS: IF 'UsageStatePackage
             THEN users = users + 1
                  AND IF (users = capacity)
                      THEN usageState = busy
                      ELSE IF (users = 1)
                      THEN usageState = active
          FI   FI   FI;
),
(logout;
    COND: IF UsageStatePackage THEN users > 0 FI;
    EFFECTS: IF 'UsageStatePackage
             THEN
                  ( users = users -1
               AND IF (users = 0)
                   THEN usageState = idle
                   ELSE IF ('usageState = busy)
                        THEN usageState = active
             FI      FI   FI
          AND ( IF (usageState = idle AND 'administrativeState = shuttingDown)
                   THEN administrativeState = locked
                   FI );


),
(increaseCapacity;
    COND: IF UsageStatePackage THEN TRUE FI;
    EFFECTS: IF 'UsageStatePackage
             THEN capacity = 'capacity + n
                  AND IF (usageState = busy AND n > 0)
                      THEN usageState = active
             FI       FI;

),
(decreaseCapacitySucess;
    COND: IF UsageStatePackage THEN capacity -n >= users FI;
    EFFECTS: IF 'UsageStatePackage
             THEN capacity = 'capacity -n
```

```
                    AND IF (users = capacity)
                        THEN usageState = busy
                FI      FI;
  ),
  (activateSuccess;
      COND: administrativeState = locked;
      EFFECTS: administrativeState = unlocked;
  ),
  (activateFailure;
      EFFECTS: administrativeState = 'administrativeState;
  ),
  (disable;
      EFFECTS: IF 'UsageStatePackage
               THEN usageState = idle
               FI;
  )
 }
```

L'ensemble des comportements de notifications associé au système étendu est:

$N_{ExtendedComputerSystem}$ =

```
 {
  (stateChange;
   TRIGGER: ( administrativeState <> 'administrativeState)
                OR ( UsageStatePackage AND (usageState <> 'usageState));
   ISSUANCE: IF ( administrativeState <> 'administrativeState)
                THEN MEMBER(stateSet,{"administrativeState ",administrativeState})
                FI
             AND IF ( UsageStatePackage AND (usageState <> 'usageState))
                THEN MEMBER(stateSet,{"usageState ",usageState})
                FI;
  )
 }
```

B.4 Calcul des ensembles résultants par héritage

L'ensemble E_{res} des événements de la sous-classe calculé par l'algorithme d'héritage est égal à l'union des ensembles $E_{BasicComputerSystem}$ et $E_{ExtendedComputerSystem}$.

L'ensemble B_{res} des comportements de la sous-classe est égal au couplage des ensembles $B_{BasicComputerSystem}$ et $B_{ExtendedComputerSystem}$ par l'algorithme de construction par héritage. Pour des raisons de taille des ensembles, nous ne donnerons ici que deux éléments significatifs de l'ensemble B_{res}. Ces éléments sont, d'une part la règle d'activation qui est héritée, et d'autre part la règle liée à l'action de blocage qui est une nouvelle action de la sous-classe.

On a donc:

B_{res} =

```
 {
  ...
  (activateSuccess;
    COND: operationalState = disabled
        OR administrativeState = locked;
```

```
    EFFECTS: operationalState = enabled
             AND response = successful
             AND administrativeState = unlocked;
  ),
   ....
  (lock;
     COND: administrativeState <> locked;
     EFFECTS: administrativeState = locked
          AND IF ('UsageStatePackage)
              THEN usageState = idle AND users = 0
              FI;
  ),
  ...
}
```

On retrouve bien ici le couplage **OR** des pré-conditions et le couplage **AND** des post-conidtions sur les règles spécialisées dans la sous-classe (cas de la fonction `activate`). Les nouvelles règles sont ajoutées à B_{res} sans modifications (cas de la règle `lock`).

L'ensemble N_{res} est égal à la composition par héritage des ensembles $N_{BasicComputerSystem}$ et $N_{ExtendedCon}$ par l'algorithme de construction par héritage. Le résultat de celui-ci est:

N_{res} =

```
 {
  (stateChange;
     TRIGGER: operationalState <> 'operationalState
           OR (( administrativeState <> 'administrativeState)
                 OR ( UsageStatePackage AND (usageState <> 'usageState)));
     ISSUANCE: MEMBER(stateSet,{"OperationalState ",operationalState})
           AND (  IF ( administrativeState <> 'administrativeState)
                   THEN MEMBER(stateSet,{"administrativeState ",administrativeState})
                   FI
                  AND IF ( UsageStatePackage AND (usageState <> 'usageState))
                      THEN MEMBER(stateSet,{"usageState ",usageState})
                      FI );
  )
 }
```

Ici aussi, on retrouve la construction par affaiblissement de la pré-condition et renforcement de la post-condition.

B.5 Calcul des règles de comportement

Il nous reste maintenant à construire les règles CRS associées à ces trois ensembles. Cela se fait en deux étapes. Dans un premier temps, les spécifications de l'ensemble de E_{res} sont couplées avec celles de B_{res} et N_{res}. En appliquant la première étape sur les ensembles B_{res} et N_{res} obtenus précédemment, les règles sont étendues de la manière suivante:

B_{res} =

```
 {
  ...
  (activateSuccess;
```

```
    COND: ( operationalState = disabled
            OR administrativeState = locked )
          AND Management.activateReq();

    EFFECTS: operationalState = enabled
             AND response = successful
             AND administrativeState = unlocked
          AND Management.activateRsp(response);
  ),
   ....
  (lock;
    COND: administrativeState <> locked
         AND Management.lock();
    EFFECTS: administrativeState = locked
          AND IF ('UsageStatePackage)
               THEN usageState = idle AND users = 0
               FI;
  ),
  ...
}
```

où l'on voit que les événements sont couplés par **AND** avec les règles, et

N_{res} =

```
 {
  (stateChange;
     TRIGGER: operationalState <> 'operationalState
                  ...
     ISSUANCE: ( MEMBER(stateSet,{"OperationalState ",operationalState})
                 AND (  IF ( administrativeState <> 'administrativeState)
                        THEN MEMBER(stateSet,{"administrativeState ",administrativeState})
                        FI
                     AND IF ( UsageStatePackage AND (usageState <> 'usageState))
                        THEN MEMBER(stateSet,{"usageState ",usageState})
                        FI )
               AND Notification.stateChange(stateSet);
  )
 }
```

où la clause **ISSUANCE** est étendue avec l'occurence de l'événement (ici `stateChange`).

La dernière étape consiste à intégrer, d'après la dernière partie de l'algorithme donné dans le chapitre 5 de la seconde partie du document, le comportement de notifications (ensemble N_{res} étendu dans les règles du comportement B_{res} étendu). On obtient alors l'ensemble $RULES$ qui comporte l'ensemble des règles du comportement associé à la sous-classe.

Sur l'exemple des ensembles B_{res} et N_{res}, le résultat est le suivant (en ne considérant pour des raisons de taille que l'action `activate`):

$RULES$ =

```
 {
  ...
  (activateSuccess;
    COND: ( operationalState = disabled
```

```
            OR administrativeState = locked )
        AND Management.activateReq();

    EFFECTS: operationalState = enabled
             AND response = successful
             AND administrativeState = unlocked
             AND Management.activateRsp(response)
        AND IF (operationalState <> 'operationalState
                ...)
            THEN ( MEMBER(stateSet,{"OperationalState ",operationalState})
                     AND (  IF ( administrativeState <> 'administrativeState)
                            THEN MEMBER(stateSet,{"administrativeState ",administrativeState})
                            FI
                       AND IF ( UsageStatePackage AND (usageState <> 'usageState))
                            THEN MEMBER(stateSet,{"usageState ",usageState})
                            FI )
                       AND Notification.stateChange(stateSet)
                     )
                FI;
 ),
 ...
}
```

B.6 Résumé

Dans cet appendice, nous avons développé entièrement un exemple d'application du formalisme LOBSTERS. Cet exemple nous a permis de suivre les différentes étapes de construction du système de règles associés à un objet géré formalisé avec LOBSTERS. Ceci nous a mené jusqu'à la description complète des règles CRS qui régissent le comportement de la sous-classe.

Glossaire

ACSE Association Control Service Element

ADT Abstract Data Type

ALS Application Layer Structure

API Application Programming Interface

ASE Application Service Element

ASN.1 Abstract Syntax Notation One

ATM Asynchronous Transfer Mode

CARUSSIM Communicating Automated RUle Systems SImulator

CCITT Comité Consultatif International Télégraphique et Téléphonique

CCR Commitment Concurrency and Recovery

CMIP Common Management Information Protocol

CMIS Common Management Information Service

CORBA Common Object Request Broker Architecture

CRS Communicating Rule Systems

CRUSADE Communicating RUle Systems Automated Development Environment

DMI Definition of Management Information

DML Domain Management Language

DQDB Dual Queue Dual Bus

EFSM Extended Finite State Machine

ENC European Networking Center

ESTELLE Extended State Transition Language

FDT Formal Description Technique

GDMO Guidelines for the Definition of Managed Objects

GDMR Guidelines for the Definition of Managed Relationships

GRM General Relationship Model

IAB Internet Activity Board

IBM International Business Machines Corporation

IDL Interface Description Language

IP Internet Protocol

ISDN Integrated Service Digital Network

ISO International Standardization Organization

ITU International Telecommunications Union

IVMO Initial Value Managed Object

LAN Local Area Network

LM Layer Management

LME Layer Management Entity

LOBSTERS Language for Object Behaviour Specification based on Templates and Extended Rule Systems

LOTOS Language Of Temporal Ordering Specifications

LTS Labelled Transition System

MACF Multiple Access Control Function

MAN Metropolitan Area Network

MIB Management Information Base

MO Managed Object

MOC Managed Object Class

MOCS Managed Object Conformance Statement

MODE Managed Object Development Environment

MOI Managed Object Instance

MOIF Managed Object InterFace

NMF Network Management Forum

ODP Open Distributed Processing

OMG Object Management Group

OS/2 PM Operating System/2 Presentation Manager

OSDL Object Specification and Description Language

OSF Open Software Foundation

OSI Open Systems Interconnection

PABX Private Automatic Branch eXchange

PDU Protocol Data Unit

PM Protocol Machine

PTS Pass Through Service

RDN Relative Distinguished Name

RMON Remote network MONitoring MIB

ROSE Remote Operations Service Element

SACF Simple Association Control Function

SAP Service Access Point

SDL Specification and Description Language

SM System Management

SMAE System Management Application Entity

SMAP System Management Application Process

SMI Structure of Management Information

SMP Simple Management Protocol

SNA Systems Network Architecture

SNACC Sample Neufeld ASN.1 to C/C++ Compiler

SNMP Simple Network Management Protocol

TCP Transmission Control Protocol

TDF Technique de Description Formelle

TMN Telecommunication Management Network

TP Distributed Transaction Processing

TR Token-Ring

TTCN Tree and Tabular Combined Notation

UDP User Datagram Protocol

UIL User Interface Language

VDM Vienna Development Method

WAN Wide Area Network

XMP X/Open Management Protocol

XOM X/Open Object Management

Index

Nom : FESTOR

Prénom : Olivier

DOCTORAT DE L'UNIVERSITE DE NANCY I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 14 OCT. 1994 n°289

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ NANCY 1 ★ LE PRESIDENT

J.P. FINANCE

3C-F9

UNIVERSITÉ HENRI POINCARÉ · NANCY 1
24-30, rue Lionnois B.P.3069 54013 Nancy cedex
Tél : 83 32 81 81 Fax : 83 32 95 90
Adresse électronique : _____ @presn1.u-nancy.fr

Formalisation du comportement des objets gérés dans le cadre du modèle OS

Résumé: Dans le cadre des activités de gestion de réseaux OSI, une approche orientée-o a été retenue pour la modélisation sous forme d'objets gérés, des ressources de communication. Pour permettre une spécification rigoureuse et uniforme des objets gérés, l'ISO a normalisé le langage GDMO. Ce langage permet une description statique des ressources, mais ne comporte à ce jour aucun mécanisme pour la description formelle du comportement associé. Cette thèse propose d'étendre le langage GDMO avec de tels mécanismes sur la base de contraintes de compatibilité et de simplicité d'utilisation. Basé à la fois sur les concepts de systèmes de règles communiquants et sur la notation GDMO, nous proposons le langage LOBSTERS. Il permet de spécifier formellement l'ensemble des caractéristiques des objets gérés de manière compatible à GDMO.
Pour faciliter et encourager l'utilisation du formalisme, un ensemble d'outils de développement basés sur LOBSTERS est également proposé. Cet environnement s'appelle MODE et exploite à différents niveaux du développement, les informations contenues dans la description formelle du comportement des objets gérés.
Mots clefs : Comportement, GDMO, Gestion de réseaux, Modèle de référence OSI, Objet géré, Spécification, Technique de Description Formelle.

Formal description of managed object behaviour in the OSI management framework

Abstract: Within the OSI management framework, an object-oriented approach was choosen for the modeling of network resources as managed objects. To allow a rigorous and unique specification of these managed objects, the GDMO notation has been standardized within ISO. This notation allows a static description of managed objects but does not provide any mechanism for the formal specification of their behaviour. The work presented in this thesis proposes an extension of the GDMO notation called LOBSTERS. This formalism provides mechanisms for the formal description of managed object behaviour which are compatible with the standardized GDMO notation.
To facilitate and encourage the use of the formalism, several development tools based on LOBSTERS are proposed. The resulting environment is called MODE and exploits, at different levels of the development process, information provided by the formally described behaviour of managed objects.
Keywords: Behaviour, GDMO, Network management, OSI reference model, Managed object, Specification, Formal Description Technique.