Contribution à la spécification et à l'exploitation des relations entre objets de gestion de réseaux

Emmanuel Nataf

Université Henri Poincaré NANCY 1

Département de formation doctorale en informatique
École doctorale IAE + M
UFR STMIA

Thèse

présentée pour l'obtention du titre de

Docteur de l'Université Henri Poincaré, Nancy-I

en Informatique

par Emmanuel NATAF

Contribution à la spécification et à l'exploitation des relations entre objets de gestion de réseaux

Soutenance publique prévue le 28 octobre 1998 à 15h au bâtiment LORIA, salle A008 devant la commission d'examen :

Membres du jury :

Rapporteurs : M. J.J. PANSIOT Professeur, Université Louis Pasteur, Strasbourg
M. J.P. THOMESSE Professeur, ENSEM, Institut National Polytechnique de Lorraine
M. S. ZNATY Professeur, ENST Bretagne, Cesson-Sevigne

Examinateurs : P. DINI Chercheur, CRIM, Canada
erche INRIA, Nancy
versité Henri Poincaré, Nancy-I

Laboratoire Lorrain de Recherche en Informatique et ses Applications
INRIA–Lorraine

Remerciements

Je remercie les membres de mon jury :
- Monsieur Jean Jacques Pansiot, Professeur à l'Université Louis Pasteur de Strasbourg,
- Monsieur Jean Pierre Thomesse, Professeur à l'Ecole Nationale Supérieure d'Electricité et de Mécanique de Nancy,
- Monsieur Simon Znaty, Professeur à l'Ecole Nationale Supérieure de Télécommunication de Brest,

d'avoir accepté la charge de rapporteur.

Je remercie Monsieur André Schaff, Professeur à l'Ecole Supérieure d'Informatique et Applications de Lorraine à Nancy de m'avoir proposé cette thèse et acceuilli au sein de son équipe.

Je remercie Monsieur Olivier Festor, Chargé de Recherche à l'Institut National de Recherche en Informatique et en Automatique pour avoir suivi et participé à mon travail ainsi qu'à la rédaction d'articles. Son amitié, son enthousiasme et son interêt pour ce travail m'ont encouragé tout au long de ces trois années. Nos nombreuses discussions et ses multiples relectures m'ont permis d'achever ce mémoire.

Je remercie Monsieur Petre Dini, Chargé de Recherche au Centre de Recherche Informatique de Montréal d'avoir accepté de relire ce mémoire, ainsi que pour les idées qu'il m'a apporté et qui y ont contribué.

Je remercie Messieurs Laurent Andrey et Samir Tata pour le travail en commun qui a donné lieu à des publications.

Table des matières

Table des figures

Chapitre 1

Introduction générale

1.1 L'importance des réseaux

Les domaines des réseaux d'ordinateurs et des télécommunications ont une complexité croissante. Les technologies de transmission sont de plus en plus évoluées et performantes; l'étendue géographique des réseaux est planétaire, les applications multi-média requièrent de plus en plus de ressources, et le nombre d'utilisateurs est en contante progression. Les demandes de ces derniers tendent vers une interconnexion de tous ces éléments; par exemple, une entreprise ayant des filiales réparties sur l'ensemble d'un pays peut désirer un service de télécommunication lui permettant de voir ses réseaux locaux éloignés comme une unique entité, où une application distribuée peut se déployer de manière transparente. La fourniture de services de télécommunication, dans un environnement de concurrence commerciale internationnale, devient pour les opérateurs l'enjeu d'un marché dans lequel les qualités telles que la rapidité de disponibilité du service, les moyens de contrôler qui les accompagent, la qualité de la facturation, etc, deviennent primordiales.

1.2 La gestion des réseaux

On peut considérer que la notion de gestion dans les réseaux est apparue quasiment en même temps que ceux-ci. Les protocoles de communication intègrent des moyens pour contrôler, lors des émissions et réceptions de données, que la fonctionnalité désirée est réalisée. Avec l'ampleur des réseaux, de nouveaux problèmes apparaîssent fréquemment, comme localiser une machine dans un immeuble, tracer une connexion entre des réseaux hétérogènes ou commander des équipements distants. Les fournisseurs de réseaux locaux proposent des applications adaptées à la gestion de leurs produits. Ce ne sont pas généralement des outils génériques et extensibles à volonté par l'utilisateur, mais au contraire ce sont souvent des logiciels spécialisés dans leur domaine et figés dans leurs fonctionnalités. Une conséquence de cette approche propriétaire est l'impossibilité d'utiliser ces outils pour une interconnexion de réseaux hétérogènes, où chaque système de gestion reste cloisonné dans son environnement.

L'émergence du réseau Internet et les évolutions dans le domaine des télécommunications ont poussé respectivement l'IAB [1] et l'ISO [2] à travailler sur la standardisation de la gestion des réseaux. Le but de ces travaux est de proposer des moyens de gestion indépendants des fournisseurs de réseaux, afin de pouvoir gérér des interconnexions hétérogènes et d'y développer des applications de gestion qui soient portables. Alors que l'IAB se focalise plus sur une approche technique de la gestion, fondée sur les protocoles, l'ISO définit un ensemble de concepts pouvant s'adapter à tous types de réseaux. L'association de ce

1. Internet Activity Board
2. International Standardization Organization

dernier organisme avec l'UIT-T[3] pour de nombreuses normes de gestion de réseaux montre l'importance et la portée de la gestion ISO. Celle-ci ne doit pas cependant assombrir le succès de la gestion de l'IAB qui, par son pragmatisme et sa mise en oeuvre peu onéreuse s'est rapidement imposée sur le marché de la gestion. La situtation actuelle est celle d'une cohabitation des différentes **approches de gestion** et elle évolue vers leurs interactions (les approches de gestion peuvent échanger des informations) et leur intégration (une même vision indépendante des approches).

La plupart des approches de gestion définissent un concept d'**objet géré**. Un tel objet est une entité d'information localisée dans un noeud du réseau et dont les valeurs sont significatives de l'état d'une **ressource** de réseau. La ressource est le terme usuel pour désigner un équipement physique, un logiciel ou un processus, etc qui sont présents dans un réseau. L'état d'une ressource lui est spécifique; un équipement peut être allumé ou éteint, un processus peut être (entre autres) actif ou suspendu. Un objet géré qui maintient de telles informations sur un ressource est une modélisation de cette dernière. Chaque objet géré nécessite des mécanismes dépendants de la ressource pour lui extraire ses informations d'état. Cette instrumentation de la ressource doit aussi permettre de modifier ses information ou de lui envoyer une commande à partir de l'objet géré.

Si l'instrumentation est à priori différente pour chaque ressource, les informations significatives de l'état d'un même type de ressource (par ex. les routeurs ou les connexions) sont semblables. C'est là l'intérêt des objets gérés : offrir une **interface de gestion** homogène qui fasse abstraction de la diversité et de la complexité interne des ressources. Les objets gérés qui modélisent un même type de ressource présentent donc les mêmes types (au sens types de données) d'informations. Ces objets ont donc tous la même définition de leur interface et l'ensemble des définitions des différentes interfaces constitue un **modèle de gestion**. Ces modèles regroupent généralement des définitions d'interfaces (ou spécifications d'objets gérés) propres à un type de réseaux (TCP/IP, ATM, SDH, X25, ...) ou un type de service (réseaux privés virtuels, téléconférence, ...). La complexité de tels modèles entraîne souvent l'utilisation d'autres moyens de spécification que les interfaces de gestion, comme des schémas, des exemples, des automates, des formules mathématiques, etc.

1.3 Les relations entre objets gérés

Les ressources modélisées par les objets gérés ne sont généralement pas isolées. Une ressource peut être composée de ressources elles-même décomposables, jusqu'à des ressources de base. Des ressources séparées sont dépendantes lorsqu'elles réalisent chacune une fonctionnalité d'un but commun, comme l'établissement d'une connexion entre deux entités de protocoles ou la fourniture de courant d'un onduleur à des équipements.

Dans de nombreuses approches de gestion ou, plus généralement, d'environnements distribués faisant interagir des objets, le concept de **relation** entre les objets est séparé de celui des objets. Dans un environnement non distribué, les relations entre les objets sont généralement représentées par leurs adresses en mémoire. Les langages de programmation disposent de nombreuses fonctionnalités qui tiennent compte de telles relations, par exemple dans la copie d'un objet qui peut se faire soit uniquement en "surface", où il n'est recopié que l'objet, soit en "profondeur", où tous les objets référencés par le premier sont également recopiés. Dans les environnements distribués, outre le problème de l'identification globale des objets, subsiste le besoin de maintenir des relations entre les objets et d'attribuer à celles-ci des traitements qui leurs soient propres.

Le domaine de la gestion de réseaux est particulièrement propice à la présence de relation entre les objets gérés. Les ressources que ces derniers modélisent sont des composantes de réseaux généralement complexes avec une activité intensive, maintenant de grandes quantités d'informations. Les objets sont

3. Union Internationale des Télécommunications - secteur Télécommunications

donc souvent eux-même très dynamiques et nombreux, liant et rompant des relations, rapidement et avec plusieurs autres objets. La prise en compte des relations est également importante dans la conception des modèles de gestion. Il y est très souvent fait une distinction entre les entités qui vont devenir des objets et les associations entre ces entités. Un modèle de gestion sera d'autant plus facilement déduit de la phase de conception si les concepts de cette dernière (entité et association) sont directement traduisibles en termes du modèle (objet et relation).

La gestion de réseaux de l'ISO prévoit, dans le contexte de son modèle orienté-objet, un modèle dédié à la spécification des relations entre les objets. Ce modèle est plus récent que celui des objets et son utilisation progresse dans plusieurs directions. Le but de cette thèse est de contribuer à deux de ces directions; d'une part la **formalisation du comportement** des relations et d'autre part **l'exploitation** par un outil de gestion des spécifications de relations.

1.3.1 Le comportement

Les objets du modèle de l'ISO sont formellement décrits au niveau de l'interface qu'ils proposent aux utilisateurs (i.e. les outils de gestion). Ces interfaces sont constituées globalement d'attributs et de méthodes avec leurs types de paramètres d'appel et de retour. Les **comportements**, dans les approches orientés-objet traditionnelles, établissent la sémantique de ces éléments, en termes d'invariants, de pré- et post-conditions. Dans les langages de programmation orientés-objet, le comportement est généralement représenté par le code des méthodes. Les comportements des objets de l'ISO sont spécifiés informellement, en langage naturel, laissant le choix au spécifieur d'y mettre ce qu'il désire, de la description à titre informel de l'élément de réseau modélisé, à l'énoncé de règles sur ses valeurs.

1.3.2 Validations du comportement

Le comportement des objets de la gestion ISO a été le sujet de plusieurs travaux visant à en formaliser la spécification. Les buts de ces spécifications sont, par exemple, la validation de propriétés assurant que la spécification est correcte ou la génération de tests pour vérifier le bon fonctionnement d'une implantation. L'intérêt de ces travaux est grandissant en regard de la complexité des modèles de gestion. Pour la même raison, l'existance d'outils logiciels permettant de traiter des spécifications formelles de comportements est une nécessité. Nous nous sommes donc intéressés à l'utilisation de méthodes de spécification formelles, dotées d'un outil de validation, pour modéliser d'une part le comportement des objets de la gestion de l'ISO, d'autre part celui des relations qu'ils ont entre eux. La séparation entre ces deux types de comportements, tout en restant dans un même formalisme, permet de faciliter le passage d'un modèle de gestion orienté-objet et relation à une spécification exploitable avant implantation. Comme il n'existe pas de méthode formelle adaptée à tous les types de besoins et ayant un support logiciel conséquant, la spécification d'un même modèle de gestion gagne à être faite avec plusieurs méthodes qui permettent d'en avoir des vues complémentaires. Les fonctionnalités des outils qui supportent ces méthodes sont généralement différentes, ce qui permet d'obtenir différents résultats. Les validations que nous avons effectuées sont exhaustives ou effectuées via des simulations interactives. Dans les premières, des propriétés sont prouvées par une recherche ou un parcours sur la totalité des états possibles du modèle de gestion. Les simulations interactives ont permis de vérifier qu'il existait des successions d'états qui étaient désirées dans le modèle de gestion. En plus de la présence requise d'outils, il nous a semblé important d'en choisir des connus et déjà utilisés dans le domaine des télécommunications et de leur gestion.

1.3.3 Un formalisme normalisé

Les normes de la gestion de l'ISO ont récemment évoluées pour étendre leur modèle objet d'un formalisme de description du comportement. La puissance d'expression du modèle objet initial de l'ISO rend nécessaire un formalisme qui soit conçu spécifiquement pour ce modèle. L'expérience acquise précédement dans la validation de comportement, où nous avons exprimé ceux des objets et des relations avec une même méthode, nous a conduit à rechercher comment réutiliser au maximum ce nouveau formalisme. De plus, le langage de spécification pour modéliser des relations étant similaire sur plusieurs points avec celui des objets, il serait perturbant de ne pas retrouver des similitudes entre les fomalismes de spécification des comportements. Un modèle de gestion entièrement et uniquement spécifié avec les formalismes de données (interfaces des objets et des relations) et de comportement peut alors être la base d'une traduction vers des méthodes formelles pour par exemple, en valider des propriétés.

En adaptant le formalisme comportemental des objets à celui des relations, nous avons saisi l'occasion pour ajouter des fonctionnalités absentes dans le modèle de relation de l'ISO. Ces fonctionnalités visent à permettre l'exploitation des relations au sein d'un système de gestion déployé sur un réseaux réel.

1.3.4 L'exploitation

L'exploitation des objets dans un système de gestion, ou **environnement de gestion**, normalisé consiste à manipuler (contrôler et observer) ou à maintenir ces derniers selon les contraintes spécifiées avec les formalismes précédents. L'environnement de gestion peut accueillir des **applications de gestion** qui automatisent ces tâches. La prise en compte de l'exploitation dans les normes concerne uniquement les objets. Ainsi, un service et un protocole de communication existent pour manipuler les objets à distance. Dans le but de réutiliser le travail de spécification des relations, nous avons cherché à concevoir ce que pourrait être l'exploitation de ces relations par des applications de gestion. En premier lieu, certaines fonctionnalités similaires à celles rencontrées dans l'exploitation des objets ont été envisagées. Ainsi, un service dédié à la manipulation des relations et une structure de base de données se sont avérés être un minimum à pourvoir aux utilisateurs. Dans un deuxième temps, d'autres fonctionnalités sont apparues dignes d'intérêt, comme l'inférence de nouvelles relations dans un environnement de gestion, ou la manipulation des relations à un niveau d'abstraction plus élevé.

L'ensemble des concepts pour l'exploitation des relations constitue une architecture logicielle pour une plate-forme de gestion. Une hypothèse dans notre conception est la prise en compte dynamique par une telle plate-forme de spécifications de relations (et de leur comportement). De nombreuses relations peuvent exister entre les objets et il est essentiel de pouvoir contrôler la vue d'un environnement de gestion au travers de ces relations. De nouvelles relations peuvent donc apparaître pour les besoins d'une application de gestion, puis disparaître lorsque celle-ci a terminé ses traitements.

1.4 Plan

Le document est divisé en trois parties et comprend une annexe. La première partie présente le domaine de recherche. La partie suivante expose nos contributions et la dernière partie décrit un prototype qui implante quelques unes de ces propositions.

Le deuxième chapitre est une introduction à la gestion de réseaux, les concepts généraux y sont donnés et leur application dans les approches de la gestion par l'IAB et l'OSI y est résumée. Le chapitre 3 dresse un état de l'art concernant l'utilisation du concept de relation dans la gestion de réseaux. Le modèle de relation de l'ISO est détaillé dans le quatrième chapitre, avec un exemple de son utilisation. Le dernier chapitre de cette partie compare ce modèle avec d'autres modèles de relation et établit des équivalences entre eux.

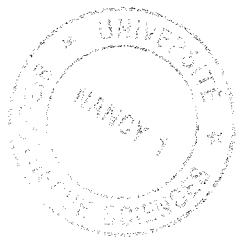
Le chapitre 6 expose notre contribution dans le domaine de la validation de propriétés d'un ensemble de spécifications d'informations de gestion. Ce chapitre vise à motiver le besoin de formaliser le comportement des relations. Il propose des solutions pour spécifier avec des outils génériques des concepts propres à la gestion de réseaux. Le chapitre 7 contient notre proposition de formalisme pour la spécification du comportement des relations. L'intégration de celui-ci est faite avec le modèle de relation et avec un formalisme existant pour le comportement des objets. Un exemple de spécification y est présenté partiellement, le complément se trouvant en annexe. Le chapitre qui clôture la deuxième partie présente notre contribution à l'exploitation des relations. Un ensemble de concepts permettant la création d'une architecture logicielle y est défini. Cette architecture a comme principale fonctionnalité la gestion des relations à partir de leur spécification et doit s'effectuer au travers des réseaux gérés.

Le chapitre 9 commence par présenter les outils que nous avons développés dans un usage général et qui ont été utilisés pour le prototype de cette thèse. Le chapitre 10 détaille ce prototype et en montre une application.

Une conclusion générale termine le document. L'état de nos travaux y est résumé et il des perspectives pour l'intégration des contributions de cette thèse dans des applications de gestion et dans des environnements distribués sont établies.

Première partie

Administration de réseaux et modélisation de relations



Cette partie présente l'administration de réseaux par ses principaux concepts. Quelques unes des applications de ces concepts par différents organismes regroupant des fabricants et des opérateurs de télécommunications seront survolées. Nous insisterons sur l'approche de la gestion qui est faite par l'ISO.

Le concept de relation étant dans la problématique centrale de cette thèse, nous nous interesserons particulièrement à l'application de ce concept dans les différentes approches de gestion. Le modèle de relation de l'ISO sera vu en détail. Il sera comparé avec d'autres modèles existants.

Chapitre 2

Concepts de la gestion de réseaux et leurs applications

2.1 Introduction

L'objectif de ce chapitre est de présenter les concepts de l'administration[4] de réseaux qui forment le cadre général de cette thèse. L'administration de réseaux, telle que nous allons la présenter, est définie par différents organismes de standardisation. Une première section est consacrée aux points communs entre la majorité de ces définitions d'origines diverses. Les deux sections suivantes montrent comment sont appliqués ces concepts communs par les deux organismes les plus influents dans le domaine de la gestion de réseaux: l'IAB et l'ISO. C'est principalement sur la gestion vue par l'ISO que porte le reste du document. Une dernière section décrit brièvement quelques autres approches de gestion de réseaux et d'environnements distribués. Avant de décrire les concepts de base, un aperçu des aires fonctionnelles de la gestion de réseaux termine cette section.

Les aires fonctionnelles sont définies dans une norme de l'ISO [ISO89] mais elles sont valables pour toute approche de gestion. Chacune de ces aires peut être vue comme un objectif général de l'exploitation d'un environnement de gestion. Les cinq aires fonctionnelles sont :

- l'aire de **gestion des fautes** qui concerne les défaillances de fonctionnement des ressources. Une défaillance doit pouvoir être détectée; les dépendances entre les ressources entraînant souvent des défaillances en cascades, elles ne facilitent pas la découverte de l'origine. Les traitements visant à remédier à une faute dépendent de sa gravité et des moyens de contrôle sur les ressources fautives;
- l'aire de **gestion de la configuration** qui vise à maintenir les ressources dans des états cohérents. La complexité des ressources permet de nombreuses configurations de leur état et les interconnexions de ces ressources rendent ces configurations dépendantes. La gestion de la configuration concerne aussi l'identification des ressources et la topologie des réseaux;
- l'aire de **gestion de la performance** qui doit assurer la qualité de service des réseaux (QoS[5]), lorsque des débits de communication doivent être respectés, tout un ensemble de ressources doivent fournir leur part de service pour que le tout satisfasse l'utilisateur. Les informations à manipuler pour estimer la QoS sont nombreuses et variées, comme les temps de disponibilités d'un service ou d'un équipement, les délais de réponses, les taux d'erreurs, etc ...;
- l'aire de **gestion de la sécurité** qui consiste à prévenir les réseaux de dégradations volontaires ou non et de les préserver d'effraction ou d'espionnage. La cryptographie, la gestion des utilisateurs et de leurs droits font partie de cette aire;

4. on utilisera indifféremment : administration ou gestion
5. Quality of Service

- l'aire de **gestion de la comptabilité** qui regroupe tout ce qui permet d'établir un coût d'utilisation des ressources. Dans les réseaux publics cette gestion est de première importance et la diversité des services offerts nécessite des processus complexes de facturation.

Les aires fonctionnelles peuvent utiliser des fonctionnalités communes. Par exemple la journalisation des accès extérieurs ou celle des défaillances d'une ressource. Les aires fonctionnelles peuvent également utiliser des fonctionnalités entre elles, par exemple suite à la détection d'une faute, le réseau peut nécessiter une reconfiguration visant à isoler la ressource à l'origine de la faute. Dans [Sta93], les aires fonctionnelles sont sub-divisées en surveillance (*monitoring*) et commande (*control*) des ressources.

La division en aires fonctionnelles n'est pas la seule manière connue de voir les buts de la gestion. Dans [ZS97], un concept de qualité de service est défini pour les couches de communication d'un réseau, les équipements, les services et les applications. Ce concept est évalué dans chacune de ses application par un même ensemble de critères qui permettent d'estimer l'état de fonctionnement du réseau. Par exemple, le critère de délai mesure la notion de temps de réponse. Appliqué à un réseau au niveau physique le délai s'exprime en temps de traitement du signal dans un nœud, c.a.d. le temps mis pour que les bits envoyés à l'équipement de transmission soient effectivement sur le support et en temps de propagation sur ce support physique. Ce même critère est évalué au niveau réseau par les temps de traitement et d'attente dans un nœud, lié à la commutation ou au routage. Entre les nœuds, ce critère est mesuré par le délai de l'établissement d'une connexion et le temps de transit.

2.2 Concepts de base

Les approches de gestions sont des ensembles de concepts définis pour gérer les réseaux. Ces approches peuvent être applicables à tous types de réseaux, spécifiques à une certaine catégorie (télécommunications, réseaux locaux) ou encore totalement propriétaires (réseaux Novell, SNA d'IBM, Apple talk). Les concepts de base qui sont décrits dans cette section se retrouvent généralement dans les différentes approches de gestion.

2.2.1 Organisation de la gestion

La gestion de réseaux est une activité distribuée entre deux types d'acteurs, le gestionnaire et l'agent, qui communiquent via un réseau pour exploiter un environnement de gestion. Le gestionnaire a généralement l'initiative de la communication, l'agent reçoit les données émises et agit en conséquence dans son contexte (qui sera vu dans la section suivante). Un gestionnaire peut communiquer avec plusieurs agents et un agent peut être accédé par plusieurs gestionnaires. Des extensions de ce schéma de communication existent, fondées sur l'échange d'informations entre gestionnaires ou sur une ambivalence entre gestionnaire et agent. La répartition des acteurs de la gestion peut se faire suivant les aires fonctionnelles, en disposant d'autant de gestionnaires agissant sur un même ensemble d'agents. Les gestionnaires peuvent aussi avoir différents niveaux de responsabilité, les agents pouvant, de leur coté, n'accepter certaines communications que si elles sont originaires de gestionnaires d'un niveau donné. Dans les approches de gestion qui seront vues plus loin, la fonctionnalité de base du gestionnaire est de fournir aux applications de gestion le côté client d'un service de communication dédié à la gestion. Il existe d'autres approches de la gestion qui ne font pas la distinction entre gestionnaire et agent, tout n'étant qu'objets modélisant aussi bien des applications que les ressources de réseaux sur lesquelles ces applications reposent.

2.2.2 Modèle de l'information

Dans le chapitre précédent, les objets gérés ont été présentés comme étant des informations qui modélisent les ressources et qui pouvaient être décrits par une interface de gestion. Le modèle de l'information

consiste à définir le concept d'objet géré et à le formaliser.

Le concept d'objet géré peut être comparé à celui d'un méta-type de données. Un modèle de gestion qui est défini dans une approche est composé de plusieurs applications du même concept d'objet géré. Avec un modèle de l'information assez puissant, comme celui de l'ISO, l'analyse préalable à la conception d'un modèle de gestion peut être directement menée avec le modèle de l'information. Il y a là un avantage par rapport à l'utilisation d'un modèle de données différent et dont les spécifications devront être transposées dans le modèle de l'information de gestion, avec les risques de perte de sémantique que cela engendre. Généralement, les modèles de l'information se limitent à la description des interfaces des informations. Il est rarement décrit comment exprimer le comportement de ces interfaces, qui imposeraient des contraintes sur les valeurs de ces informations ou sur l'accès même aux interfaces. On trouve dans quelques approches des conseils d'utilisations de méthodes formelles reconnues (comme Z ou LDS). Le concept de **relation gérée** fait partie du modèle de l'information.

Dans un environnement de gestion, les objets gérés sont regroupés dans des bases d'informations de gestion (MIB [6]). Les MIBs sont sous la responsabilité des agents (un agent maintient une MIB et une MIB est maintenue par un agent). Une partie d'un modèle d'information de gestion définit la structure générale des MIBs grâce à laquelle les objets gérés sont accessibles depuis un gestionnaire. L'intérêt visé d'une telle structure est de rendre efficace la recherche des objets dans une MIB qui peut à priori en contenir un grand nombre.

Le concept d'objet géré du modèle de l'information a une grande influence sur la communication entre gestionnaires et agents car celle-ci vise à manipuler les objets gérés.

2.2.3 Communications de gestion

Les communications entre gestionnaires et agents reposent sur un service de gestion, supporté lui même par un protocole de communication. Dans la plupart des cas, un modèle client / serveur est suivi. Ainsi, le service de gestion est invoqué par le gestionnaire (le client) pour manipuler les objets gérés. L'agent (le serveur) perçoit l'indication correspondant à cette requête, applique le service aux objets gérés ciblés et constitue des éventuelles réponses. La transmission de ces réponses utilise à nouveau le service de gestion depuis l'agent, à la suite de quoi le gestionnaire reçoit la confirmation de sa requête. Cependant, à l'inverse du modèle client/serveur, l'agent peut faire des requêtes au service de gestion pour notifier un gestionnaire des événements survenus dans ses objets. Le service de gestion est proposé aux acteurs sous la forme de **primitives de gestion** dont les paramètres et leur sémantique sont définis dans l'approche. Une primitive de gestion est généralement composée d'une primitive d'appel et d'une primitive de retour qui sont utilisées respectivement pour la requête / indication et la réponse / confirmation du service. Un échange de ces primitives entre un gestionnaire et un agent est appelé une **interaction de gestion**.

Le protocole de gestion spécifie les types de données (ou PDU [7] qui véhiculent les primitives de service. Plusieurs primitives peuvent utiliser la même PDU, ce qui diminue le nombre de PDU mais pas forcément leur complexité. La communication de gestion est conçue avec la contrainte de ne pas surcharger les réseaux gérés avec des communications de gestion. L'utilisation d'un réseau séparé physiquement ou d'un partage de la bande passante pour la gestion est possible.

Dans une interconnexion de réseaux, différentes approches de gestion peuvent être appliquées sur les sous-réseaux. La communication d'information de gestion entre les approches est nécessaire pour donner une vision homogène de l'interconnexion. Un agent de proximité (agent *Proxy*) doit alors être interposé entre les environnements de gestion, faisant la traduction des interactions de gestion. Les requêtes de service d'un gestionnaire d'une approche sont dirigées vers le *Proxy*, celui-ci les transmet vers les agents

6. Management Information Base
7. Protocol Data Unit

de l'autre approche. Ce passage d'une approche vers une autre doit se faire au niveau du service de gestion et du modèle de l'information. Chaque primitive de service de gestion et le concept d'objet géré d'une approche doivent avoir une correpondance dans l'autre approche. Cette traduction doit également s'effectuer dans l'autre sens, les agents émettant des réponses et des notifications vers le gestionnaire via le *Proxy*.

La figure 2.1 illustre les différents modèles intervenant dans la gestion qui viennent d'être cités. Sur cet exemple, un service de communication est diponible pour un ensemble de ressources (une seule ressource `R` est schématisée). Un service de gestion `G1` est offert à un `gestionnaire` qui peut l'utiliser pour les communications de gestion avec les `agents` `A` et `AB`. Le premier de ces agents contient dans sa MIB, un objet géré modélisant la ressource `R`. Cette ressource n'a pas accès au service de gestion, aussi, la valeur de l'objet géré est évaluée par l'agent qui utilise le service de communication pour récupérer les informations sur la ressource. Dans l'exemple, deux services de gestion sont présents, traduisant une cohabitation possible entre deux approches différentes. L'agent `AB` est plus qu'un simple agent car il maintient une MIB pour chaque approche. La MIB de l'approche de gestion correspondant au service `G1` peut être la vision, adaptée à son modèle de l'information, de la MIB de l'approche liée au service `G2`. Cette MIB peut être constituée par l'agent `AB` en utilisant le service de gestion `G2` pour rechercher des valeurs dans les MIBs des agents `B1` et `B2`. Ce mécanisme confère alors à cet agent le rôle de gestionnaire pour l'approche de gestion `G2`. L'agent `AB` peut aussi être considéré comme un agent *Proxy* si l'on considère que les MIBs qu'il contient sont virtuelles.

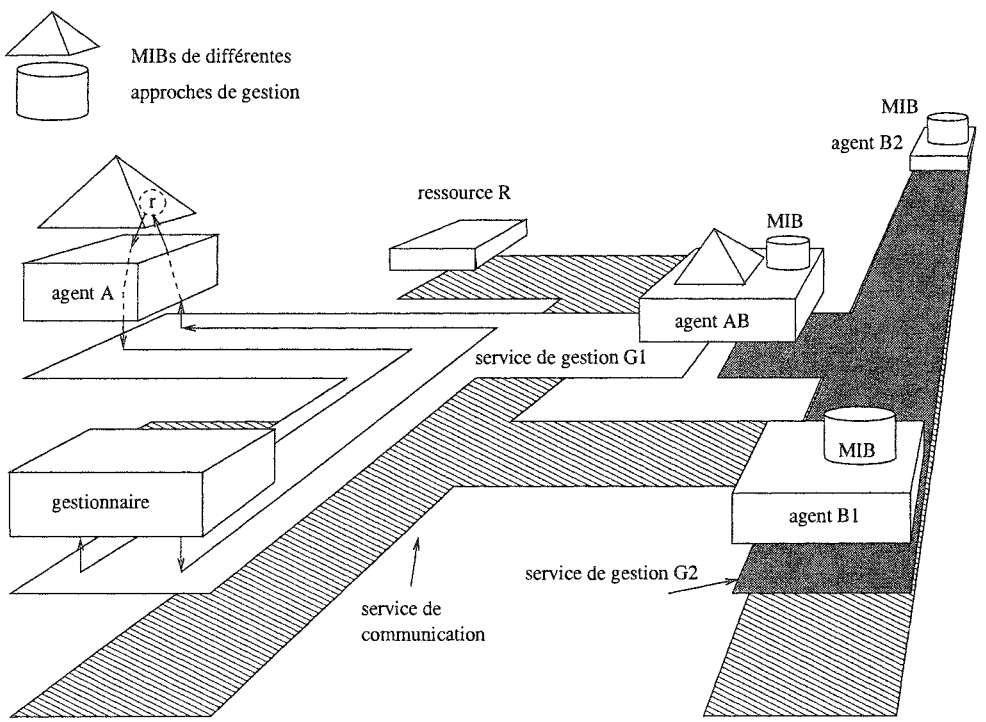


FIG. 2.1 – *Les modèles de gestion*

2.2.4 Fonctions de gestion

Les fonctions de gestion ne se retrouvent pas dans toutes les approches. Une fonction de gestion a un but et elle est définie par un ensemble d'objets gérés qui vont servir à réaliser ce but. Une même fonction

de gestion peut participer à plusieurs aires fonctionnelles. Quelques exemples (issus de la gestion de l'ISO) de telles fonctions sont la journalisation [ISO92c] et le filtrage d'événements [ISO92d], le contrôle de l'accès aux objets gérés [ISO91b] ou le test sur des objets gérés [ISO91a].

2.3 La gestion SNMP

La gestion SNMP apparaît dans la fin des années 80 [Ros94] avec l'objectif d'instrumenter les équipements de réseaux utilisant la pile TCP/IP. Comme pour toutes les spécifications de ces protocoles, l'IETF[8] produit des RFC[9] concernant la gestion. Les RFCs servent de référence pour les fabricants qui peuvent assurer la conformité de leur produits avec les spécifications de l'IETF.

Ne désirant faire qu'une présentation rapide des concepts de l'approche SNMP, nous n'abordons que la première version, qui est la base des extentions proposées dans la deuxième version (SNMPv2 [CMRW93a] [CMRW93b] [CMRW93c]) et dans la dernière version en date : SNMPv3 [HW98a] [HW98b] [LS98].

2.3.1 Le modèle de l'information

Les objets gérés de l'approche SNMP modélisent, dans leur première définition, des entités de protocoles contenues dans un nœud de réseau (une station de travail, un routeur, etc). Il existe aussi de nombreuses applications du concept d'objet géré pour modéliser tout ce qui participe à son fonctionnement (onduleurs, systèmes d'exploitation, etc).

Les objets gérés contiennent des valeurs qui sont généralement d'un type simple, mono-valué. Les objets gérés peuvent être placés dans des tables à deux dimensions, qui sont également considérées comme des objets gérés (mais ils n'ont pas de valeurs autres que celles des objets gérés contenus). Un objet géré contient, en plus de sa valeur modélisant la ressource, des paramètres qui en précise l'utilisation; ces paramètres sont : les **droits d'accès**, le caractère obligatoire ou optionnel de la **présence** de l'objet géré dans une MIB et une description en langage naturel de l'objet. Lorsque l'objet est une table (plus précisemment la définition des entrées de la table) un paramètre supplémentaire indique quels sont les colonnes de la table qui permettent de distinguer chaque ligne (i.e. l'**index** de la table).

Le formalisme utilisé pour spécifier les objets gérés est la syntaxe ASN.1[10] [ISO92i]; une syntaxe abstraite (différente d'une syntaxe concrète qui décrit un langage [Mey92]) permettant la définition de divers types de données. La syntaxe ASN.1 s'accompagne de règles de codage en binaire pour le transfert de valeurs des types définis en ASN.1 [ISO87]. La spécification du type de la valeur contenue dans un objet géré et des paramètres vus ci-dessus constituent le type de l'objet géré. Il est lui même spécifié avec un (méta) type spécialement défini en ASN.1[11] pour les objets de l'approche SNMP [RM90] : le type `OBJECT-TYPE`.

La syntaxe des types simples contenus dans les objets gérés utilise un sous-ensemble de la notation ASN.1; des types simples comme `INTEGER`, `OctetString` et le type complexe `SEQUENCE`, qui décrit une liste ordonnée de types simples. D'autre types simples ont été rajoutés, comme `IpAdress` pour la valeur d'une adresse IP. Une table est du type `SEQUENCE OF` et décrit une liste ordonnée de valeurs d'un même type, ce type devant être du type `SEQUENCE`.

Les types des paramètres d'accès et de présence sont des types énumérés; les valeurs pour l'accès déterminent si la lecture et/ou l'écriture sont possibles, ou encore si la valeur est accessible. La présence

8. Internet Engineering Task Force
9. Request For Comments
10. Abstract Syntax Notation Number One
11. à l'aide d'un mécanisme de macro-définition de type

de l'objet géré dans la MIB est obligatoire (`mandatory`), optionnelle (`optional`) ou tolérée (`deprecated`). Dans ce cas, l'objet ne devrait plus être utilisé car un agent peut ne pas l'implanter dans sa MIB tout en se déclarant conforme au RFC.

Un dernier paramètre dans la définition d'un tel type est l'**identificateur** de l'objet géré, dont la valeur distingue le type de l'objet géré de tous les autres types d'objets gérés existants. En plus de cette distinction, l'identificateur positionne le type de l'objet géré parmis les autre types. Ce placement explicite des types d'objet géré permet leur regroupement en fonction des ressources qu'ils modélisent.

Pour illustrer la spécification d'objets gérés, la définition d'une table est détaillée dans la suite. La figure 2.2 schématise les objets gérés définis pour spécifier la table des connexions utilisant le protocole TCP. L'objet `tcpConnTable` est en fait composé d'un objet `tcpConnEntry` qui décrit le type des entrées (lignes) de la table. Ce type est une séquence de cinq types simples qui permettent de caractériser une connexion TCP. Une telle connexion est établie entre deux entités de protocoles, l'une locale et l'autre éloignée, désignée chacune par son adresse IP et le numéro de port utilisé. Ces valeurs constituent les quatres dernières colonnes de la table. Le protocole TCP est défini par un automate et la première colonne caractérise l'état dans lequel se trouve l'automate de l'entité locale. C'est à l'agent de remplir ou vider cette table, en fonction des connexions TCP établies avec la ressource gérée.

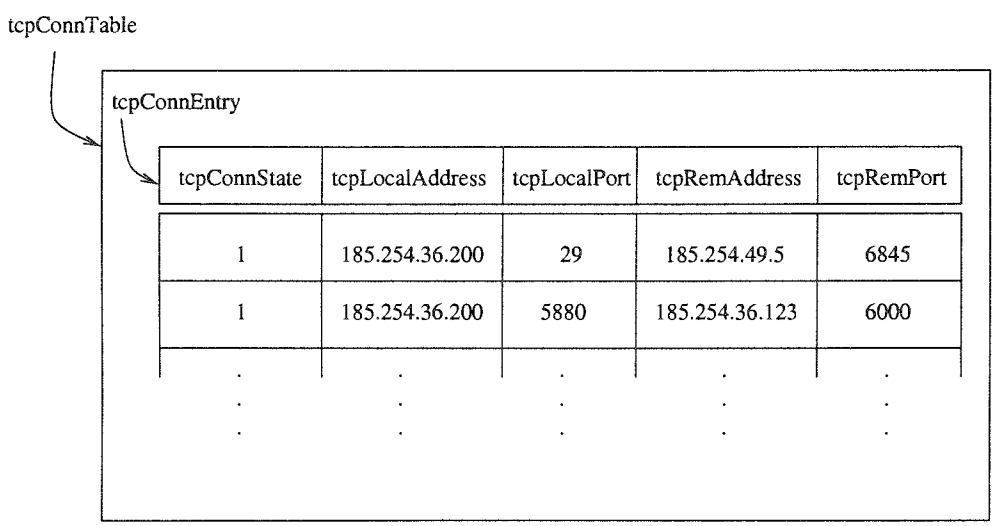


FIG. 2.2 – *Exemple d'instanciation de la table* `tcpConnTable`

La figure 2.3 montre les spécifications ASN.1 qui définissent cette table. Le type d'objet `tcpConnTable` (ligne 1) a une valeur de type `SEQUENCE OF` (ligne 2), dont les éléments ont le type `TcpConnEntry` qui s'avère être une séquence de types simples (lignes 10 à 16). Comme nous l'avons fait remarquer plus haut, les tables n'ont pas de valeur en soit et leur accès est interdit (ligne 3). Cette table est obligatoire pour une MIB qui veut gérér une entité de protocole TCP suivant le RFC prévu à cet effet (la MIB-II [MR91]). L'identificateur (ligne 9) spécifie une succession de nombres (`tcp` étant lui même une suite de nombre) dont nous verrons la composition et la sémantique plus loin. Le type `tcpConnEntry` (ligne 17) définit le type des entrées de la table `tcpConnTable`, ce type (ligne 18) est celui référencé dans la table (ligne 2). L'accès et le statut de l'entrée `tcpConnEntry` (lignes 19 et 20) sont identiques à celui de la table, tout comme cette dernière, un type d'entrée n'est pas à accéder. L'entrée contient l'énumération des champs de sa séquence qui servent d'index à la table (lignes 22 à 26) pour distinguer chacune de ses lignes. Les tables contiennent des objets gérés et la définition de ceux-ci doit être faite pour chaque colonne de la table (i.e. pour chaque champ de la séquence de l'entrée). Par

```
(1)  tcpConnTable OBJECT-TYPE
(2)    SYNTAX SEQUENCE OF TcpConnEntry
(3)    ACCESS not-accessible
(4)    STATUS mandatory
(5)    DESCRIPTION
(6)      ''Une table contenant des
(7)        informations spécifiques
(8)        des connexions TCP.''
(9)    ::= {tcp 13}

(10) TcpConnEntry ::=
(11)   SEQUENCE
(12) {tcpConnState  INTEGER(1..12),
(13)  tcpConnLocalAddress ipAdress,
(14)  tcpConnLocalPort INTEGER,
(15)  tcpConnRemAddress ipAdress,
(16)  tcpConnRemPort INTEGER}

(17) tcpConnEntry OBJECT-TYPE
(18)   SYNTAX TcpConnEntry
(19)   ACCESS not-accessible
(20)   STATUS mandatory
(21)   DESCRIPTION ''...''
(22)   INDEX
(23) {tcpConnLocalAddress,
(24)  tcpConnLocalPort,
(25)  tcpConnRemAddress,
(26)  tcpConnRemPort}
(27) ::= { tcpConnTable 1}

(28) TcpConnState OBJECT-TYPE
(29)     SYNTAX INTEGER
(30)    {closed (1),
(31)      listen (2),
(32)      ...
(33)      deleteTCB(12)}
(34)     ACCESS read-write
(35)     STATUS mandatory
(36)     DESCRIPTION ''...''
(37)   ::= { tcpConnEntry 1}
```

FIG. 2.3 – *Définitions de types d'objets gérés*

exemple l'objet modélisant les état de l'automate du protocole TCP est spécifié dans les lignes 28 à 37.

L'identificateur de l'entrée (ligne 27) positionne ce type d'objet géré sous le type de la table. En effet, la valeur `tcpConnTable` de l'identificateur est aussi égale à : `tcp 13`, c'est à dire l'identificateur de la table (ligne 9). L'identificateur de l'objet modélisant l'état de l'automate positionne ce type en dessous de celui de l'objet `tcpConnEntry`, la valeur de `tcpConnEntry` (ligne 37) ayant la même valeur que `tcpConnTable 1` (ligne 27).

L'ensemble des identificateurs de tous les objets gérés peut être vu comme un arbre; (cas particulier d'un graphe) où les nœuds sont ces identificateurs et une arête entre deux nœuds signifie qu'un nœud est le préfixe de l'autre. Les identificateurs sont des suites d'entiers et le préfixe que l'on considère est cette suite sans le dernier chiffre. L'arbre dans lequel est contenue la définition des objets gérés de l'approche SNMP est géré par l'ISO et l'IAB. Ils attribuent des identificateurs à toutes sortes d'objets. La figure 2.4 montre une partie de cet arbre. L'IAB a en fait la responsabilité du sous-arbre à partir du nœud `internet`, identifié par la séquence : 1.3.6.1. Le sous-arbre `mib-2` (1.3.6.1.2.1) contient les identificateurs des objets gérés définis par l'IETF pour gérer les nœuds du réseau Internet [MR91]. On y retrouve donc la table des connexions TCP et son entrée (1.3.6.1.2.1.6.13 et 1.3.6.1.2.1.6.13.1). Le nœud tcp (ou groupe tcp) contient les objets gérés modélisant une entité de protocole TCP. Il existe ainsi d'autres groupes contenant des identificateurs d'objets spécifiques à un protocole ou une couche (groupes IP, interface, system ...). Les fabricants de ressources ont la possibilité de définir des objets gérés propres à leurs produits, dans la branche `enterprises` (1.3.6.1.4.1). Dans la figure, les objets gérés entourés en traits discontinus contiennent des valeurs accessibles (ce sont les feuilles de l'arbre).

Un agent contient une MIB d'objets gérés. Cette MIB est caractérisée par les groupes d'identificateurs précédents. Par exemple, un agent qui contient les objets modélisant les nœuds de l'Internet utilisera le groupe `mib-2`. Un agent ne peut contenir dans sa MIB qu'une occurence par type d'objet géré. Les objets gérés modélisant les entrées des tables sont des exceptions où les lignes de la table sont des occurences du même type d'entrée. De par cette unicité, les objets peuvent être identifiés sans ambiguïté dans la MIB grâce à leur identificateur de type. Par exemple, l'objet géré qui contient le temps écoulé depuis le démarrage d'un agent est l'objet `sysUpTime`, dont l'identificateur de type (1.3.6.1.2.1.1.3) sert à l'identification de l'occurence de l'objet géré. L'identification des objets gérés contenus dans les tables se fait par la concaténation de l'identificateur de l'objet géré de la colonne où se situe l'objet recherché avec la concaténation des valeurs contenues dans les objets des colonnes faisant office d'index.

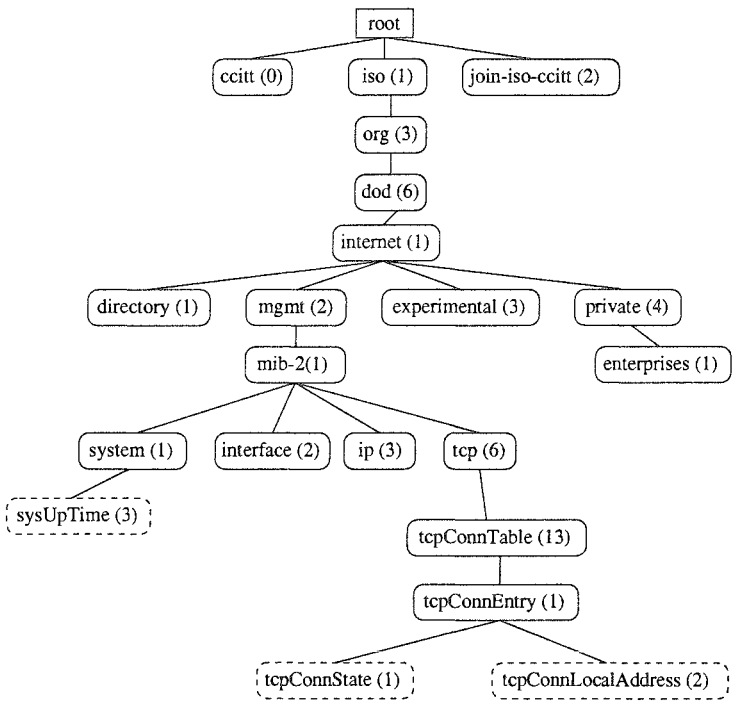


FIG. 2.4 – *L'arbre de référencement des objets gérés*

2.3.2 Le protocole SNMP

La communication entre gestionnaire et agent a été définie sur la base du protocole SNMP [CFSD90]. Le service que rend ce protocole est constitué de quatre primitives:

- la primitive GET permet de lire une valeur d'objet géré, identifié par l'identificateur de son type, concaténé avec la valeur 0 dans le cas des objets gérés qui ne sont pas contenus dans une table (par ex. l'occurence du type d'objet `sysUpTime` est accessible avec la séquence 1.3.6.1.2.1.1.3.**0**). Il est possible de spécifier plusieurs identificateurs dans une primitive d'appel et de recevoir les réponses correspondantes en retour sous la forme d'une liste de couples (identificateur, valeur);
- la primitive GET-NEXT prend les mêmes paramètres que le service GET mais retourne, pour chaque objet géré accédé, le couple (identificateur, valeur) de l'objet suivant dans la MIB. L'ordre entre les objets gérés correpond à un parcours en profondeur d'abord de l'arbre de ses identificateurs;
- la primitive SET permet la mise à jour des valeurs contenues dans les objets gérés, comme précédemment, plusieurs objets gérés peuvent être désignés dans une requête de primitive. La réponse est constituée des couples (identificateur, nouvelle valeur) des objets modifiés.

Si une erreur se produit dans la liste de couples (objet inexistant ou valeur non adéquate) des requêtes de services précédentes, la réponse contient un code d'erreur et l'indice dans la liste de l'objet géré ayant provoqué l'erreur.

- le service TRAP est émis de l'agent vers le gestionnaire. L'information qu'il contient réfère à un type d'événement prédéfini, comme une réinitialisation de l'agent ou un problème de communication sur une interface. Les fabriquants peuvent également spécifier leurs propres événements; un des types prédéfinis d'évément étant prévu pour cela. Ce type doit être accompagné d'un type d'événement spécifique au produit.

Le protocole SNMP définit trois PDUs (figure 2.5) : la PDU pour l'émission d'une requête de service d'un gestionnaire vers un agent, la PDU pour toutes les réponses d'un agent vers un gestionnaire et la PDU pour le service de notification. Le protocole d'échange des PDUs est simple; le gestionnaire peut émettre une PDU à tout instant et attend une PDU de réponse. De son coté, l'agent attend les PDUs du gestionnaire et compose les PDUs de réponses. L'agent peut également émettre une notification, mais il n'attend rien en retour. Ce comportement correspond bien à celui de l'entité de protocole UDP[12] sur laquelle SNMP repose.

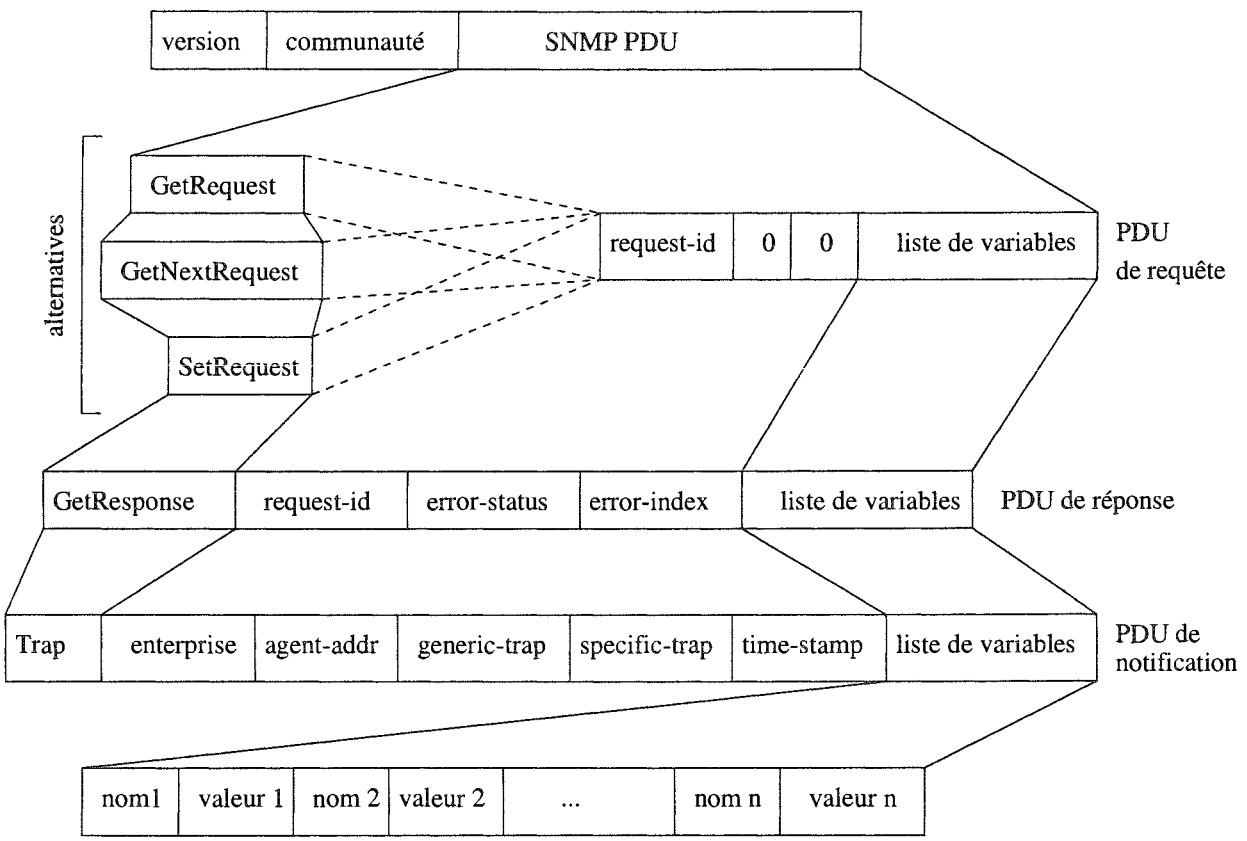


FIG. 2.5 – *Composition des PDUs SNMP*

2.4 La gestion OSI

La gestion de réseaux définie par l'ISO est faite dans le cadre de l'interconnexion des systèmes ouverts (OSI[13]) [ISO84]. On utilisera le terme de gestion OSI pour la désigner. Il existe trois types de gestion dans l'architecture en couches du modèle de référence [ISO89] (figure 2.6) :

- les **opérations de couche N** sont échangées entre deux instances d'une entité de la couche N, situées chacune dans un système ouvert différent. Ces opérations font partie de la définition de chaque protocole utilisé dans la couche N. Elles ont pour but de maintenir un bon fonctionnement d'une communication entre les entités de la couche N;

12. User Datagram Protocol
13. Open System Interconnection

- la **gestion de couche N** concerne toutes les entités de protocoles présentes dans la couche. Elle vise à maintenir un bon fonctionnement de l'ensemble de la couche. La gestion de couche requiert des protocoles de gestion de couche, spécifiques à une couche et utilisés entre des entité de gestion de couche (N-LME : *N-Layer Management Entity*);
- **la gestion système** se fait sur l'ensemble des couches d'un réseau (à l'inverse des deux précédentes) et vise à assurer un fonctionnement global du réseau par le contrôle de toutes ses ressources. Dans chaque système ouvert, une instance d'un processus d'application (SMAP : *System Management Application Process*) exécute les opérations relatives à la gestion au sein du système ouvert en coordination avec des SMAPs d'autres systèmes. Chaque SMAP utilise une entité de la couche application dédiée à la gestion (SMAE : *System Management Application Entity*) qui est responsable des communications de gestion entre SMAEs. Une SMAE contient elle-même plusieurs éléments de service de la couche application (ASE : *Application Service Element*), spécifiques à la gestion ou d'utilisation générale. Les fonctions de gestion (cf. §2.2.4) comportent des traitements qui sont regroupés dans des éléments de service de gestion (SMASE : *System Management Application Service Element*). Lorsque ces SMASEs nécessitent une communication avec d'autres SMASEs distants, ils utilisent l'élément de service d'information de gestion (CMISE : *Common Management Information Service Element*). Les éléments SMASE et CMISE sont propres à la gestion OSI. Ce dernier utilise deux éléments plus généraux; l'élément de contrôle d'une association (ACSE *Application Control Service Element*) qui permet le contrôle d'une association entre deux CMISEs. Une fois une telle association établie, les éléments CMISEs utilisent un élément permettant l'appel de procédure à distance (ROSE : *Remote Operation Service Element*).

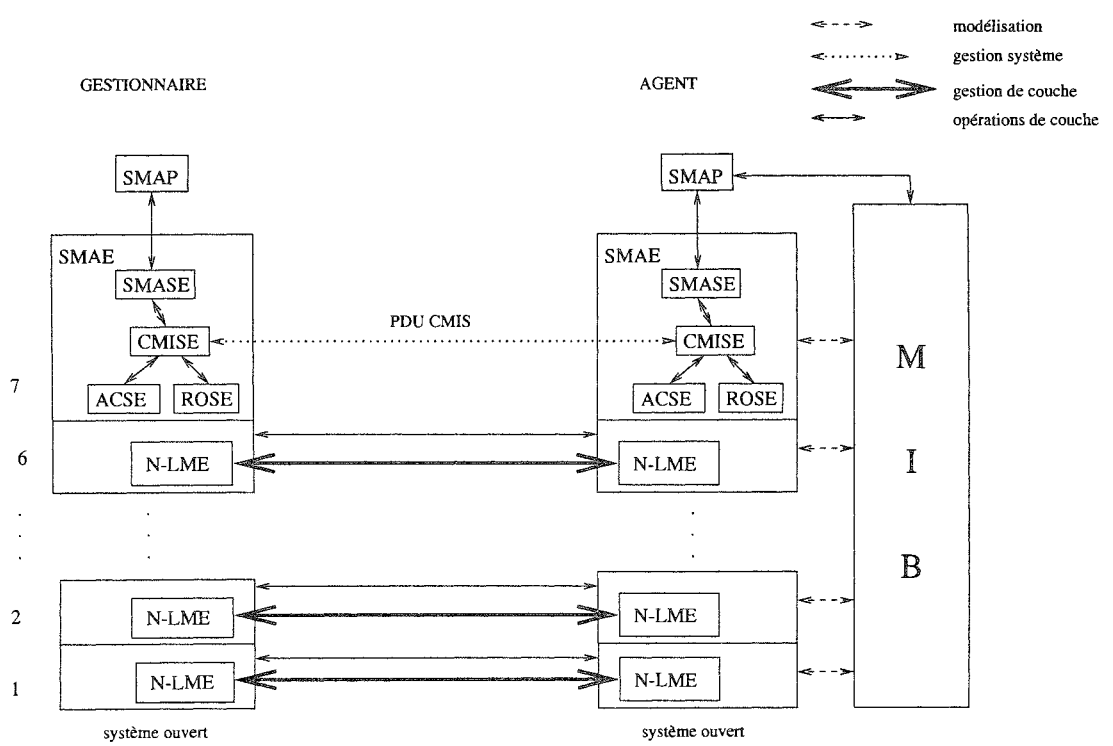


FIG. 2.6 – *La gestion dans l'architecture OSI*

Rapporté à l'organisation de gestion (cf. §2.2.1), le gestionnaire et l'agent sont deux SMAPs [ISO92a]. La MIB de l'agent maintient des objets gérés qui modélisent les ressources de toutes les couches. Une requête de gestion reçue dans un SMASE est traitée par le SMAP qui l'exécute en relation directe avec la MIB.

2.4.1 Le modèle de l'information

Tout comme dans l'approche SNMP, les objets gérés contiennent des informations dont la valeur modélise l'état des ressources. A la différence de SNMP, les objets gérés peuvent définir des opérations qui modélisent des actions applicables aux ressources et des notifications signifiant qu'un événement a été détecté dans la ressource.

Le concept d'objet géré défini par l'ISO [ISO92e] est similaire à celui du paradigme objet, comme dans [Mey88] [Boo91] et il y apporte ses particularités. Les définitions des types d'informations (ou attributs) contenus dans les objets, des actions que l'on peut leur appliquer et des notifications qu'ils peuvent émettre constituent sa définition de classe. Une classe modélise un type de ressource et les ressources effectivement présentes sont représentées par des instances d'objets gérés. Une instance peut être créée dans une MIB sur ordre d'un gestionnaire, ou s'y trouver dès la mise en route de l'agent. Des objets gérés peuvent se créer ou être détruits par l'agent en réaction à des changements dans la ressource gérée. Une instance peut également être détruite par un gestionnaire (ce qui n'était pas le cas des objets gérés de l'approche SNMP). Les caractéristiques qui constituent les classes du concept d'objet géré sont de plusieurs types, la figure 2.7 schématise cette composition :

- l'identification de la classe qui est d'une part un nom, constitué d'une chaîne de caractère, d'autre part, un numéro d'enregistrement (dans une autre partie de l'arbre de référencement de l'ISO vu dans l'approche SNMP). Cette notion de nom et de numéro d'enregistrement est utilisée pour toute caractéristique;
- un ensemble de modules obligatoires, c'est à dire qu'ils devront être présents dans toutes les intances de la classe. Un module est lui même défini par des attributs, des actions et des notifications auxquels peuvent être associés des paramètres qui interviennent dans les manipulations de ces composants (lecture d'un attribut, invocation d'action ...). Le comportement d'un module est la description de la sémantique de ses composants, elle doit exprimer des conditions qui doivent toujours être vérifiées entre l'instanciation d'un objet géré et sa destruction (conditions invariantes). La sémantique des actions et des notifications est exprimée avec d'une part des conditions devant être vérifiées avant l'exécution d'une action ou l'émission d'une notification : des pré-conditions; d'autre part, avec des conditions devant être vérifiées après l'exécution de l'action ou l'émission de la notification : des post-conditions. Dans ces modules obligatoires, le comportement du module est assimilé au comportement de la classe d'objet géré. Il est à noter que toutes les autres caractéristiques d'un module peuvent également définir un comportement qui leur est propre;
- un ensemble de modules conditionnels, qui sont des modules identiques aux précédents mais dont la présence dans une instance est liée à une condition qui est évaluée à l'instanciation de l'objet géré. Le comportement de ces modules fait partie du comportement de la classe, en conjonction avec leur condition de présence;
- un ensemble d'autres classes d'objets gérés dont les définitions font partie de celle de la classe courante.

Pour exprimer ces concepts de modélisation orientés-objet, l'ISO a défini le langage GDMO [14] [ISO92g]. La norme établit un certain nombre de formulaires (*template*) qui sont chacun dédié à la spécification des éléments de la figure 2.7. La figure 2.8 donne un exemple de définition d'une classe d'objet géré extraite

14. Guidelines for the Definition of Managed Object

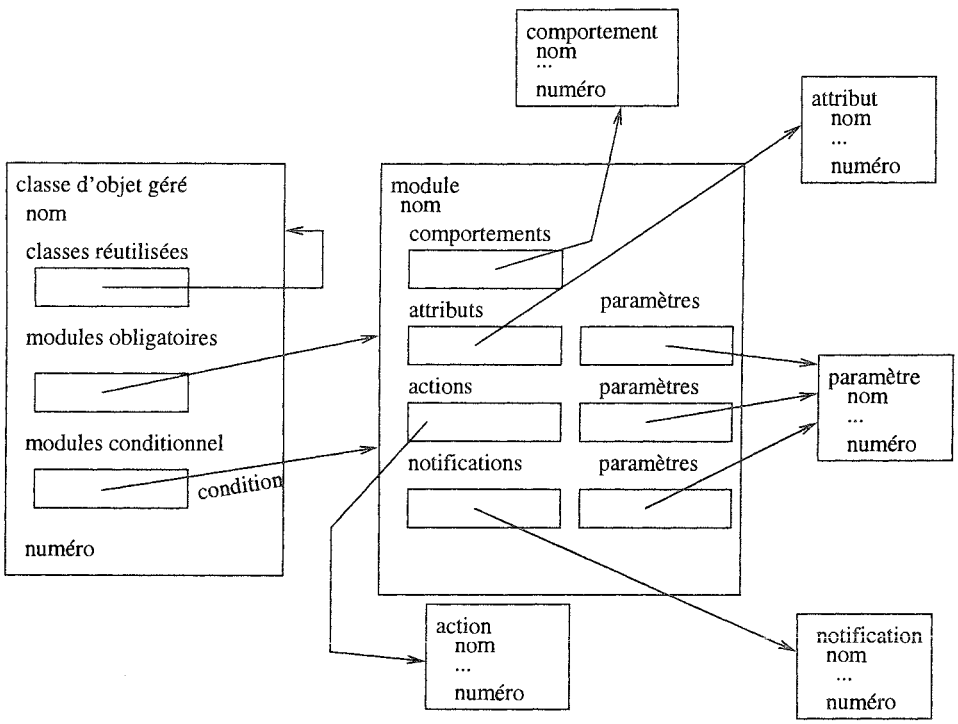


FIG. 2.7 – *La composition d'un objet géré*

de [M3195]. La classe `pipe` (ligne 1) hérite (ou est dérivée) de la classe `Top` (ligne 2) qui est la super-classe de toutes les classes GDMO [ISO92f]. La classe `pipe` contient un module obligatoire (ligne 3) nommé `pipePackage` (ligne 4). Nous avons reproduit en partie le comportement de ce module (lignes 5 à 12). Le module `pipePackage` contient quelques attributs (lignes 13 à 19). Les droits d'accès aux valeurs de ces attributs sont spécifiés dans le module. Par exemple, la lecture et l'écriture sont autorisées pour l'attribut `administrativeState` (ligne 15). La définition de la classe contient un module conditionnel (ligne 21) et sa condition de présence dans une instance de la classe `pipe` (ligne 22). Comme pour le comportement, cette condition est exprimée en langage naturel.

La réutilisation de spécifications d'autres classes correspond à la notion de spécialisation (ou d'héritage) entre classes. Ormis l'avantage de la réutilisation au niveau des spécifications de classes, l'héritage permet une vision du réseau géré à plusieurs niveaux d'abstractions. En effet, lorsqu'une classe, dite **sous-classe**, hérite d'une autre, dite **super-classe**, toutes les instances de la sous-classe peuvent être manipulées comme étant des instances de la super-classe. L'héritage peut être multiple (une sous-classe hérite de plusieurs super-classes) et il est strict dans tous les cas. l'héritage strict signifie qu'il n'est pas possible pour une sous-classe d'ôter des caractéristiques de ses super-classes. Une redéfinition de caractéristiques appartenant aux super-classes est néanmoins autorisée sous certaines conditions:

- les attributs peuvent avoir un ensemble de valeurs, dites **permises**, qui explicite l'étendue des valeurs possibles. Un autre ensemble de valeurs, dites **requises**, explicite les valeurs que doit pouvoir prendre l'attribut pour assurer son rôle dans la modélisation d'une ressource. Les valeurs requises sont un sous-ensemble des valeurs permises. Lors de la redéfinition d'un attribut, l'ensemble de ses valeurs permises est l'intersection des ensembles de valeurs permises dans les super-classes, cette intersection peut de plus être restreinte. On assure ainsi que le nouvel attribut de la sous-classe ne pourra pas avoir des valeurs non permises dans une de ses super-classes. L'ensemble des valeurs

```
(1) pipe MANAGED  OBJECT CLASS
(2)  DERIVED FROM "ISO 101652 | ITU-T X.721: 1992" : Top;
(3) CHARACTERIZED BY
(4)  pipePackage PACKAGE
(5) BEHAVIOUR
(6) pipeBehaviour BEHAVIOUR
(7) DEFINED AS
(8)      "La classe d'objets canal de communication est une classe d'objet
(9)      géré qui assure  le transfert de  l'information entre deux points
(10)     de terminaison. [...] Le sens de la connexité est déterminé par la
(11)     directionnalité des  points   de terminaison a  et  z.[...] L'état
(12)     opérationnel indique l'aptitude à transporter un signal.";;

(13)      ATTRIBUTES
(14)        directionality GET,
(15)        "ITU-T X.721: 1992":administrativeState GET-REPLACE,
(16)        "ITU-T X.721: 1992":operationalState GET,
(17)        a-TPInstance GET SET-BY-CREATE,
(18)        z-TPInstance GET SET-BY-CREATE
(19)      ;;;
(20)    CONDITIONAL PACKAGES
...
(21)    characteristicInformationPackage PRESENT IF
(22)    "une instance le prend en charge.",
...
(23) REGISTERED AS {m3100ObjectClass 24};
```

FIG. 2.8 – *La classe d'objet modélisant un canal de communication*

requises est l'union des valeurs requises dans les super-classes, cette union peut être étendue[15]. On assure ainsi que toutes les valeurs requises dans les super-classes le seront également dans la sous-classe.

- les actions et les notifications ont des arguments d'appel et de retour qui peuvent être redéfinis en utilisant la notion de sous-type : un type A est un sous-type de B si et seulement si toute valeur de type A est également une valeur du type B. Les types des arguments d'appel dans la sous-classe doivent être des super-types des arguments d'appel dans les super-classes (i.e. ces derniers doivent en être des sous-types). Les types des arguments de retour dans la sous-classe doivent être des sous-types de ceux des super-classes. Ainsi, un appel adressé à une instance d'une sous-classe, mais vue comme une instance d'une super-classe, aura toujours des arguments d'appel et de retour du bon type;
- le comportement des super-classes est fusionné dans la sous-classe de la manière suivante : l'invariant de la sous-classe est la conjonction des invariants des super-classes et de son propre invariant. Les pré-conditions des actions et notifications de la sous-classe sont la disjonction de celles des super-classes et des pré-conditions propres à la sous-classe. Les post-conditions de la sous-classe sont la conjonction de celles des super-classes et des post-conditions propres à la sous-classe.

2.4.2 La base de données d'information de gestion

L'organisation des objets gérés dans la MIB est faite pour donner une structure d'arbre (MIT : *Management Information Tree*) similaire à celle des objets gérés de l'approche SNMP. A la différence de cette dernière, il peut exister plusieurs instances d'une même classe d'objet géré. Tous les objets gérés sont potentiellement accessibles (et non uniquement les feuilles de l'arbre) et surtout, le positionnement des instances d'une classe relativement à d'autres instances n'est pas forcément unique. Ce positionnement est spécifié par une relation dite de **corrélation de noms**; elle s'établit entre deux classes, l'une dite de rôle **supérieur** et l'autre dite de rôle **subordonné**. Quand une telle relation existe, cela signifie que les

15. les valeurs requises doivent toujours être un sous-ensemble des valeurs permises

instances de la classe subordonnée sont positionnées en dessous d'une instance de la classe supérieure. Mais il est possible d'avoir plusieurs corrélations de noms pour une même classe d'objet géré dans le rôle subordonné. Par conséquent, les instances d'une telle classe peuvent se retrouver positionnées en dessous d'instances de différentes classes. La figure 2.9 montre un exemple de cette situation; les trois corrélations de noms sont nécessaires pour que la MIB d'objets gérés soit instanciée. Un objet de la classe `B` peut être placé soit sous une instance de la classe `A` (cas de `b1`), soit sous une instance de la classe `C` (cas de `b2`). Une fois créées, les instances ne sont pas déplaçables, les corrélations de noms définissent donc la structure potentielle des MIBs. Elles sont définies par un formulaire spécial (vu ci-dessous), comme la plupart des concepts du modèle informationnel de la gestion OSI, comportant un nom et un identificateur chiffré.

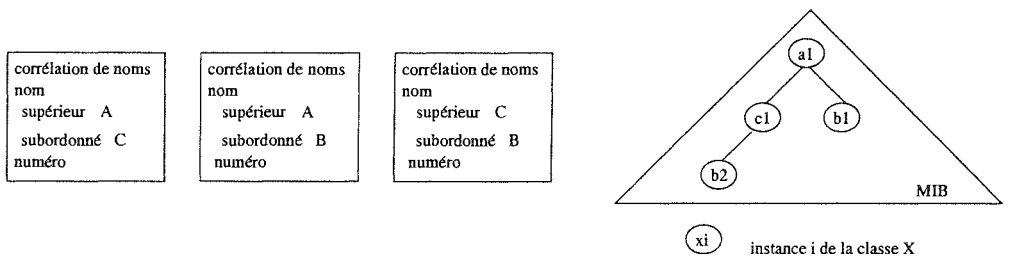


FIG. 2.9 – *Les corrélations de noms et la MIB*

Tout comme dans l'approche SNMP, la structure arborescente de la MIB permet l'identification des objets gérés. Pour comprendre comment cela est réalisé, nous présentons un exemple du formulaire dédié aux corrélations de noms : le formulaire de `NAME BINDING`. La figure 2.10 contient trois spécifications GDMO; deux classes d'objets gérés (lignes 1 à 9) et une corrélation de noms (lignes 10 à 18). La classe `connection` est dérivée (ligne 2) de celle présentée à la figure 2.8. Elle contient en plus un attribut `connectionId`. La classe `network` contient un attribut `networkId` (ligne 8) qui sert d'identificateur pour les instances de cette classe. Le formulaire de corrélation de nom positionne les objets gérés de la classe `connection` comme subordonnés (ligne 13) aux objets de la classe supérieure `network` (ligne 14). Pour les objets gérés de la classe `connection` qui sont subordonnés à un même objet de la classe `network`, la valeur de l'attribut `connectionId` (ligne 15) permet de les distinguer. La spécification de la création et la suppression (lignes 16 et 17) concernent les objets gérés subordonnés.

```
(1) connection MANAGED OBJECT CLASS
(2)  DERIVED FROM pipe;
 ...
(3)  ATTRIBUTES
(4)    connectionId;
...
(5) REGISTERED AS {M310bjectClass 23}
```

```
(6) network MANAGED OBJECT CLASS
...
(7)   ATTRIBUTES
(8)     networkId;
...
(9) REGISTERED AS {M310bjectClass 1}
```

```
(10) network-connection NAME BINDING
(13)   SUBORDINATE OBJECT CLASS connection
(14)   NAMED-BY SUPERIOR OBJECT CLASS network
(15)   WITH ATTRIBUTE connectionId;
(16)   CREATE;
(17)   DELETE;
(18) REGISTERED AS {m3100NameBinding 2}
```

FIG. 2.10 – *Un exemple de* `NAME BINDING`

L'instanciation d'un objet géré dans une MIB est faite de deux manière; soit avec le service de gestion, soit manuellement, directement sur l'agent ou la ressource. Dans le premier cas, l'instanciation d'un objet

est toujours subordonné à un autre (sauf pour l'objet au sommet du MIT) et la corrélation de noms utilisée doit spécifier que la création est possible, de même pour la suppression. Dans le second cas, la création n'étant pas possible par voie de gestion, la création ne doit pas être spécifiée (cela n'implique pas que la suppression par gestion soit impossible).

La spécification des corrélations de noms permet d'identifier chaque objet géré dans la MIB. La valeur de l'attribut référencé dans la corrélation de noms donne le nom relatif de l'objet géré (RDN : *Relatif Distinguish Name*) par rapport à l'objet supérieur. Si ce dernier est lui aussi un subordonné par une autre corrélation de noms, il possède alors lui aussi un RDN. La concaténation des RDNs des objets supérieurs à un objet forme l'identificateur global d'un objet (DN : *Distinguish Name*). La figure 2.11 montre deux exemples de DN avec les spécifications de la figure 2.10.

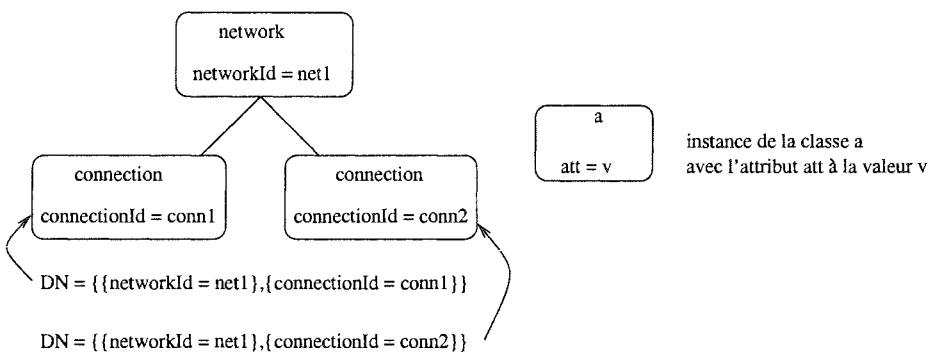


FIG. 2.11 – *Les DNs des objets*

2.4.3 Le service CMIS et le protocole CMIP

Le service de la gestion OSI offre les primitives suivantes:
- `M-CREATE` pour la création d'un objet géré avec des valeurs initiales d'attributs;
- `M-DELETE` pour la suppression d'un objet géré;
- `M-GET` pour la lecture de valeurs d'attributs dans les objets gérés;
- `M-CANCEL-GET` pour annuler une précédente requête du service M-GET;
- `M-SET` pour la mise à jour de valeurs d'attributs dans les objets gérés;
- `M-ACTION` pour demander l'exécution d'une action dans les objets gérés;
- `M-EVENT-REPORT` pour signaler à un gestionnaire l'émission d'une notification par un objet géré.

Les quatre premiers sont forcément confirmés (utilisation d'une primitive d'appel et de retour) et les trois derniers peuvent être sans réponse. Tous les services ont comme paramètre un identificateur de requête (réutilisé dans les éventuelles réponses).

Les paramètres à donner pour invoquer le service `M-CREATE` sont la classe et le DN d'un objet géré. Celui-ci doit être d'une classe qui soit dans le rôle supérieur d'une corrélation de noms où la classe de l'objet à créer (également en paramètre du service) est dans le rôle subordonné.

Les services `M-DELETE`, `M-GET`, `M-SET` et `M-ACTION` peuvent être adressés à plusieurs objets gérés en une seule primitive d'appel. La technique utilisée pour cela est la portée et le filtrage (*Scope* et *Filter*). Le principe en est la section d'objets dans un sous-arbre du MIT de la MIB pour ensuite leur appliquer le service demandé. La figure 2.12 montre les différents types de portée:
- objet de base (ou racine) pour uniquement un objet géré;
- jusqu'au niveau *n* à partir d'une racine. Un niveau est un ensemble d'objets gérés qui ont tous le même nombre d'objets gérés participants à la construction de leur DN entre eux et leur racine

commune. Tous les objets gérés de tous les niveaux depuis l'objet racine jusqu'au niveau *n* sont sélectionnés par cette portée;
- les objets du niveau *n* à partir d'une racine;
- tous les objets du sous-arbre à partir d'une racine.

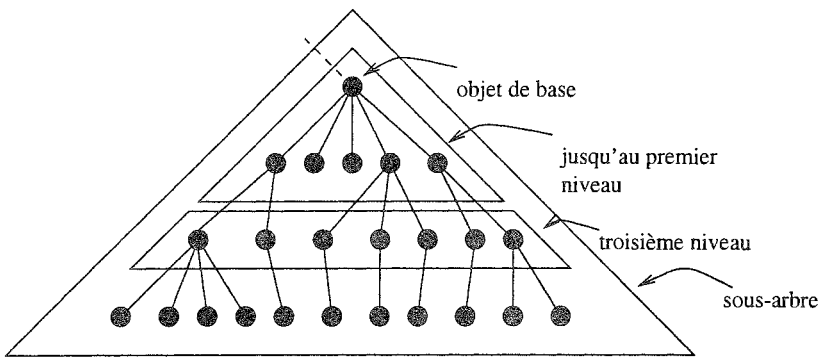


FIG. 2.12 – *Les différentes portées*

Suite à cette sélection, les objets gérés sont à nouveau sélectionnés par un filtre qui teste les valeurs des attributs d'objets. Les paramètres de ce mécanisme sont un DN d'objet qui sera la racine du sous-arbre, la portée et le filtre. Pour chacun des objets sélectionnés, lorsque le service est confirmé, une primitive de retour est émise vers le gestionnaire à l'origine de la requête. Afin que ce dernier puisse identifier toutes les réponses, un deuxième identificateur de requête les numérote.

Les autres paramètres sont spécifiques aux services, on notera que la mise à jour d'un attribut par le service `M-SET` peut se faire soit par un remplacement de valeur (`REPLACE`), soit par un ajoût ou un retrait de valeur (`ADD`, `REMOVE`) dans le cas d'un attribut dont le type est composé de plusieurs valeurs (attribut ensembliste). Le remplacement par une valeur par défaut (`SET-TO-DEFAULT`) définie dans la classe de l'objet (plus précisément dans le module) est également possible.

Le service `M-CANCEL-GET` est particulier car il ne s'adresse pas, comme tous les autres, à des objets gérés. Il vise à interrompre le travail de l'agent concernant une requête du service `M-GET` émise au préalable. Un tel service est utile lorsque de nombreux objets sont traités par une requête `M-GET`, ainsi, si une réponse particulière est reçue, ou si elle met trop de temps à arriver au gestionnaire, celui-ci peu interrompre le flot de réponses à la base en prévenant l'agent par le service `M-CANCEL-GET`.

Le protocole CMIP est composé d'autant de PDUs qu'il y a de primitives de services pour CMIS. Comme en SNMP, le gestionnaire peut faire une requête de service à tout moment. Il lui appartient d'attendre les éventuelles réponses, simples ou multiples. L'agent, de son côté, attend des indications de service, les traite sur les objets contenus dans sa MIB puis émet les réponses vers le gestionnaire qui les reçoit comme des confirmations de service.

Le protocole CMIP utilise le service de l'entité de service d'application ROSE; son utilité générale est l'appel de procédures à distance. Il est composé de quatres primitives:
- `RO-INVOKE` pour appeler une procédure à s'exécuter sur un système ouvert distant (le code de la procédure est sur le système distant);
- `RO-RESULT` pour retourner le résultat d'une procédure vers le système qui l'a appelée;
- `RO-ERROR` pour signaler à un système ayant appelé une procédure qu'elle s'est mal déroulée;
- `RO-REJECT` pour signaler qu'un appel de procédure n'a pas été accepté par le destinataire (opération inconnue, mauvais arguments, etc).

La figure 2.13 montre, par un exemple d'appel de service CMIS `M-GET`, comment sont utilisés les services de ROSE. On remarque que pour une primitive CMIS, deux primitives ROSE sont nécessaires.

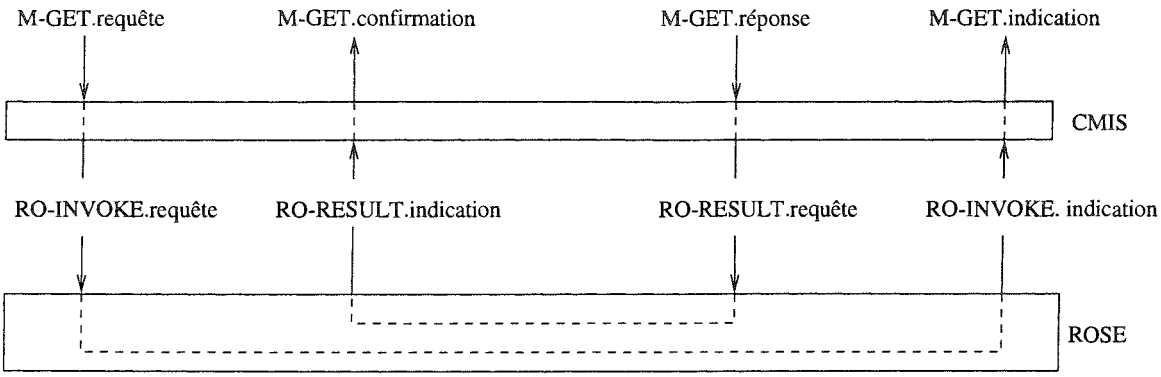


FIG. 2.13 – *Utilisation de l'élément ROSE*

2.5 Autres approches

Cette section donne un aperçu d'autres approches de la gestion de réseaux. Soit ces approches utilisent les approches précédentes, soit elles sont indépendantes. Quelques unes seront comparées dans le chapitre 5 sur le plan de la modélisation des relations qu'elles proposent.

2.5.1 Le Réseau de Gestion des Télécommunications

Dans la continuité du travail de l'ISO, auquel il a largement participé, l'UIT définit un environnement de gestion pour les réseaux de télécommunications. Le Réseau de Gestion des Télécommunications (RGT ou TMN[16]) [CCI92a] définit l'architecture nécessaire à la gestion. Le RGT étend les concepts de la gestion OSI dans plusieurs directions :

- le RGT voit les gestionnaire, agent et *Proxy* comme des blocs fonctionnels (OSF *Operating System Function*, NEF *Network Element Function* et QAF *Q Adaptor Function*) parmi d'autres, comme par exemple la fonction de médiation (MF *Mediation Function*) qui maintient des objets gérés intermédiaires, ou la fonction de station de travail qui modélise un utilisateur du RGT (WSF *Working Station Function*);
- des interfaces (ou points de références) sont prévues entre différents blocs fonctionnels, comme l'interface x, entre plusieurs gestionnaires, ou q3 entre gestionnaires et agents (*Proxy* et médiation compris);
- des niveaux de gestion et leur objets gérés sont identifiés, comme les niveaux élément de réseau, réseau et service de réseau. Les niveaux de gestion du RGT sont dépendants entre eux, des services vers les éléments;
- une architecture physique qui permet de placer les blocs fonctionnels dans le réseau de gestion;
- une architecture informationnelle [M3195], exprimée en GDMO qui décrit un réseau de télécommunication générique, fondé sur la transmission en hiérarchie digitale synchrone (SDH[17]) [G.893] (la SDH a été modélisée par un modèle de gestion dérivé de l'architecture informationnelle [G.792]).

16. Telecommunication Management Network
17. Synchronous Digital Hierarchy

2.5.2 ODP et ODMA

L'ISO normalise des concepts pour la spécification d'applications distribuée, l'ODP[18] [N1089] propose cinq points de vues devant permettre la description de ces applications. Chaque point de vue décrit une même application avec un niveau d'abstraction différent:

- le point de vue de l'entreprise, où les acteurs d'une application sont définis ainsi que les services ou contrats qui sont passés entre ces acteurs;
- le point de vue de l'information qui décrit les données de l'application et les traitements;
- le point de vue du traitement qui exprime une architecture fonctionnelle de l'application en terme d'objets de calcul qui échangent des messages par des interfaces de traitement;
- le point de vue de l'ingénierie détaille la distribution réelle des objets de calcul et les communications entre eux;
- le point de vue de la technologie traite de l'implantation physique, comme le choix d'un langage de programmation.

Ces concepts peuvent être utilisés pour la conception des applications d'administration de réseaux, qui sont par nature distribuées [IR94][CS95] [SFL96] [PK96]. L'ODMA[19] de l'ITUT-T normalise la description des concepts de gestion de réseaux en utilisant le cadre de spécification de l'ODP [G.896]. Par exemple, un formalisme est défini pour les objets et les relations du point de vue de l'information; le comportement de ces entités pouvant être spécifié en Z. Des fonctions de gestion, comme la sélection et la répartition des notification ou la gestion des connexions de sous-réseaux sont spécifiées avec ces concepts.

2.5.3 TINA-C

Le consortium TINA (TINAC[20]) définit un ensemble de concepts visant à décrire des systèmes de télécommunication. L'architecture TINA est sub-divisée en quatre [CM95]:

- l'architecture de service pour concevoir, spécifier, implanter et gérer des service de télécommunication;
- l'architecture de réseaux pour concevoir, spécifier, implanter des réseaux de transport d'information;
- l'architecture de calcul pour concevoir et composer des logiciels distribués et leur environnement;
- l'architecture de gestion pour concevoir, spécifier et implanter des logiciels utilisés pour gérer les services et les ressources sous-jaçentes.

L'architecture de gestion s'applique à toutes les activités de gestion définies dans les autres architectures (et à sa propre gestion). Elle est divisée en deux types de gestion :

- la gestion du traitement qui concerne la gestion des équipements de calcul, des plate-formes et des équipements de transmission, mais aussi des logiciels présents (déploiement, gestion de version, etc);
- la gestion des télécommunications englobe la gestion des services et du réseaux de transport;

La spécification des données utilisée dans les architectures suit une approche orientée-objet avec la prise en compte des relations [CC95]. Concernant la gestion, le concept Gestionnaire / Agent n'est pas autant marqué que pour l'OSI; les objets peuvent interagir en utilisant les services d'un environnement de traitement distribué (DPE *Distributed Processing Environment*) qui masque la distribution et qui communique les messages entre les objets [GM95].

18. Open Distributed Processing
19. Open Distributed Management Architecture
20. Telecommunications Information Networking Architecture Consortium

2.5.4 WBEM

L'approche de gestion de réseaux WBEM[21] [Tod97b] [Tod97a], à l'initiative d'entreprises telles que Microsoft Corp. ou Intel Corp., définit ses propres modèles dans l'objectif d'unifier toutes les autres approches. Concernant la modélisation des informations de gestion, une approche orientée-objet est utilisée, formalisée par un langage de spécification propre à WBEM, nommée MOF[22] [DMT97]. L'organisation de cette approche est divisée en deux couples : client/serveur et producteur/consommateur. Un acteur de la gestion peut prendre un ou plusieurs de ces rôles. Le concept d'agent est représenté par un acteur gestionnaire d'objet (HMOM : *Hyper-Media Object Manager*). Les objets qu'il maintient sont accessibles via un protocole spécifique (HMMP : *Hyper-Media Management Protocol*), pour les requêtes de service en tant que serveur et pour les notifications des objets en tant que producteur. Le HMOM peut aussi intégrer d'autres protocoles de gestion qui sont utilisés pour récupérer des informations des autre approches.

2.5.5 JMAPI

La *Java Management Application Programming Interface* (JMAPI) est une interface de programmation écrite avec le langage Java dans le but de développer des applications de gestion. Quoiqu'étant une implantation plus que des concepts du niveau des sections précédentes, la JMAPI définit un nouveau paradigme de gestion. Trois types d'acteurs sont présents dans un réseau géré avec cette API:

- les utilisateurs qui gèrent leur réseau au travers d'interfaces et d'objets Java de la JMAPI spécialisés dans la visualisation des objets gérés ou dans leur documentation (*Browser User Interface*);
- un serveur d'objets gérés (*Admin Runtime Module*) qui délivre aux utilisateur des interfaces d'accès distant à des objets gérés (en Java) maintenus localement. Un serveur peut avoir accès à une base de données pour certains attributs des objets;
- des équipements gérés (*Appliance*) qui maintiennent des informations localement aux ressources gérées. Ces informations sont des objets Java (*AgentObject*) accessibles à distance par les objets gérés contenus dans les serveurs.

Par rapport aux approches SNMP ou OSI, la JMAPI correspond à une extraction des objets gérés. Ceux-ci sont localisés dans les ARMs qui disposent de mécanismes pour l'émission de notifications. Les *Appliance* sont des agents qui délivrent des informations brutes (les *AgentObject*) aux objets gérés.

Les mécanismes de communication utilisés sont le protocole HTTP[23] et le système des invocations de méthodes à distance (RMI : *Remote Method Invocation*). Les objets gérés utilisent les RMI pour manipuler les informations locales aux ressources. Le serveur contient des programmes spécifiques aux objets gérés (représentations graphiques, organisation en table ou hiérarchique, ...) qui seront téléchargés[24] par les utilisateurs avec HTTP. Ce principe de chargement à la demande facilite l'évolution de l'environnement de gestion, les dernières versions de programmes étant immédiatement disponibles pour les utilisateurs.

La JMAPI définit également des interfaces et des objets pour utiliser le protocole SNMP. L'intégration des deux approches se fait au niveau des *Appliance* qui deviennent alors des *Proxy* entre les serveurs et les agents SNMP. Une requête RMI issue d'un serveur vers un *AgentObject* est traduite dans celui-ci par des requêtes SNMP dont les réponses mettent à jour l'*AgentObject* ou sont transmises, après une traduction inverse de la précédente, à l'objet géré demandeur.

21. Web-Based Enterprise Management
22. Managed Object Format
23. Hyper Text Transport Protocol
24. de tels programmes téléchargeables sont appelés des *applets*

2.5.6 CORBA et IDL

l'OMG[25] a été constitué pour proposer des solutions visant à faciliter l'intégration de différents logiciels dans un environnement distribué et hétérogène. Une architecture (OMA : *Object Management Architecture*) pour la mise en œuvre de ces intégrations en est le résultat. L'OMA suit une approche orientée-objet pour la modélisation des informations qui composent l'intégration. La notion de "courtier" de requête d'objet (ORB : *Object Request Broker*) en est une caractéristique importante; un courtier a la tâche de retrouver dans un environnement distribué un objet qui puisse rendre un service demandé par un objet client (ou un service équivalent). Les interfaces de programmation de cette architecture sont regroupés sous la dénomination de CORBA (*Common Object Request Broker Architecture*) [Gro98]), qui définit entre autre le langage IDL (*Interface Definition Langage*, chap. 3 de [Gro98]) pour la spécification des objets manipulés par l'ensemble des services CORBA. Les courtiers peuvent communiquer entre eux par le protocole GIOP[26], qui est implanté par exemple au dessus de la pile TCP/IP dans le protocole IIOP[27].

2.6 Conclusions

Ce chapitre a présenté les principes généraux de l'administration de réseaux. Ces principes sont tout d'abord des motivations pour mettre en place un environnement de gestion en connaissance de ce qui doit être géré. Ensuite, un ensemble de concepts qui sont valables pour tous les environnements permet de voir la gestion de réseaux indépendamment des différentes solutions existantes. L'approche SNMP est l'une des plus déployée dans les réseaux actuels et maintient sa place grâce à son acceptation et sa simplicité de mise en œuvre. Si l'approche de l'ISO est plus difficile à implanter, elle offre toutefois un pouvoir de modélisation des ressources beaucoup plus étendu que celui de SNMP. De nombreux travaux portent sur la conception de passerelles et autres *Proxy* entre ces deux approches. Sur ces approches de base, de nombreuses architectures de spécifications pour les traitements répartis ont été définies. Des applications complexes où la simple vision du couple gestionnaire / agent n'est pas assez abstraite, sont visées par ces architectures.

Le déploiement et la reconnaissance des technologies Java et CORBA ouvrent des voies pour de nouveaux paradigmes comme les agents mobiles [LKP98] [SBP98], la gestion par délégation [COSS98] ou les agents programmables [VPK97].

25. Object Management Group
26. General Inter-ORB Protocol
27. Internet Inter-ORB Protocol

Chapitre 3

Les relations dans l'administration de réseaux

Après la présentation des principes généraux de l'administration de réseaux du chapitre précédent, nous présentons dans ce chapitre différentes propositions relatives à la prise en compte des relations entre objets de gestion. Ces travaux sont principalement réalisés dans le contexte de la gestion de réseaux, avec quelques cas plus généraux liés à des environnements de traitements distribués.

3.1 Les bases de données dans la gestion de réseaux

Les systèmes de gestion de bases de données (SGBD) dont nous parlons dans cette section ne sont pas les MIBs maintenues par les agents, mais des SGBDs génériques utilisant les fonctionnalités d'un gestionnaire. Les interêts de l'utilisation de ces SGBDs sont nombreux; par exemple:

- ils peuvent offrir une transparence à la distribution des objets de gestion et donner l'illusion à des applications de ne disposer que d'une MIB globale et centralisée;
- ils peuvent gérer les accès concurrents aux objets de gestion lorsque les agents ne disposent pas ou de peu de mécanismes à cet effet;
- ils permettent l'adjonction d'objets de gestion à caractère uniquement administratif (i.e. qui ne modélisent pas une ressource, comme des entreprises, des journaux d'enregistrements) et qui sont intégrés dans la MIB globale;
- ils permettent d'organiser les objets de gestion afin que les relations qui existent entre eux soient formalisées en suivant les modèles des SGBD, libérant ainsi les applications des aspects techniques liés à la représentation de ces relations dans l'approche de gestion (des corrélations de noms, des groupes SNMP, etc). Les relations peuvent également être uniquement un choix des utilisateurs qui relient logiquement des objets de gestion sans que ces relations soient représentées dans les objets de gestion. Les exemples d'applications dont les traitements sont très liés aux relations sont nombreux [MT89] [ML89] [YYF91] [VG93] [JP93] [Tsa93] [DLW93], de même que les modélisations de réseaux utilisant des relations [Syl89] [Dit91] [BL93] [Bap93].

La grande quantité de produits disponibles facilite le choix d'un SGBD adapté à la complexité du réseau à gérer [YT94] et les plate-formes ou les environnements de gestion sont souvent associés avec des SGBDs (par ex. la JM API[28] est ouvert à l'utilisation de l'interface JDBC[29]). Ils représentent alors des fondations solides permettant de supporter des extensions des fonctionnalités et la distribution de ces plate-formes [Feh89] [Fel89] [SS89].

28. Java Management Application Programming Interface [Sun97]
29. Java Data Base Connection [WH98]

Un intérêt grandissant des SGBDs est la possiblité d'unifier des informations de gestion pour en offrir une vue indépendante des approches de gestion qui peuvent cohabiter dans une interconnexion de réseaux. Pour ce faire, les SGBDs doivent avoir un pouvoir d'expression des données suffisament fort pour englober des modèles de l'information aussi différents qu'entre les approches SNMP et OSI. La présence de plusieurs interfaces de communication (par ex. SNMP et CMIP) est également une nécessité [WS89]. Les applications y gagnent en portabilité [Agu89], elles sont à la fois indépendantes des implantations, des données dans les MIBs et des protocoles de communications.

L'utilisation des SGBDs a l'inconvénient de rendre dépendantes les applications de gestion du modèle de données implanté par ces systèmes et des interfaces de programmation qu'ils proposent. La suite de ce chapitre montre des propositions indépendantes d'un SGBD particulier, visant à permettre une spécification des relations dans les mêmes objectifs qu'un SGBD, mais dont les modèles de données utilisés sont ceux qui ont été définis dans des approches de gestion ou des environnements distribués.

3.2 La gestion SNMP

Dans l'approche SNMP, les relations qui existent entre les objets gérés sont fixées dans la définition des MIBs. Les MIBs standards ne disposent pas de moyen pour représenter une relation qui n'est pas prévue dans leur schéma. On peut citer toutefois l'exemple de la MIB RMON [Wal95] dont une table permet d'associer n'importe quel événement répertorié dans une autre table, avec n'importe quelle alarme répertoriée dans une troisième table. La deuxième version de SNMP autorise l'ajoût de lignes dans une table, ce qui pourrait donner lieu à des tables spécialisées pour représenter des relations inexistantes dans les MIBs standards.

3.2.1 Le paradigme du tableur appliqué à SNMP

Le paradigme du tableur appliqué à SNMP [SP94] est une proposition originale de structure de données adaptée au modèle de l'information de l'approche SNMP. Le paradigme du tableur décrit un tableur comme un tableau à deux dimensions (ou feuille) contenant des cellules dans chacune desquelles se trouve une information et une spécification de contrôle. Cette dernière donne une valeur à l'information, éventuellement en fonction d'autres informations contenues dans d'autres cellules de la feuille. Les autres sources d'informations des cellules sont les données contenues dans les objets gérés des MIBs SNMP. Les relations entre les objets gérés sont modélisées par les relations qui existent entre des cellules lorsque la partie contrôle de certaines d'entre elles font appel aux parties informations des autres. Plus précisément, certaines cellules contiennent dans leur partie information la valeur d'un objet géré, d'autre cellules contiennent dans leur partie contrôle les références aux cellules précédentes. Ces cellules représentent donc une relation entre des objes gérés et la partie contrôle en définit le comportement, par exemple par des notifications d'événements (des *trap*) ou plus simplement par le calcul d'une valeur pour la partie information de la cellule de relation.

L'intégration de ce paradigme dans l'approche SNMP se fait sur la base de la conformité de la structure à deux dimensions du tableur avec les objets gérés utilisés pour représenter des tables SNMP. Chaque cellule du tableur est donc accessible par un Gestionnaire, dès lors qu'il connait la valeur de l'identificateur d'objet désignant la colonne et la valeur de l'index désignant la ligne. Le tableur SNMP est donc un objet géré, sa position dans le modèle organisationnel de la gestion SNMP en définit sa portée; dans un agent, elle est limitée aux objets gérés de la MIB associée (et dans laquelle le tableur est implanté), dans un *proxy* (figure 3.1) ou un Gestionnaire, elle permet la spécification de relations entre des objets gérés de différentes MIBs. Le tableur SNMP peut être programmé dynamiquement car la partie

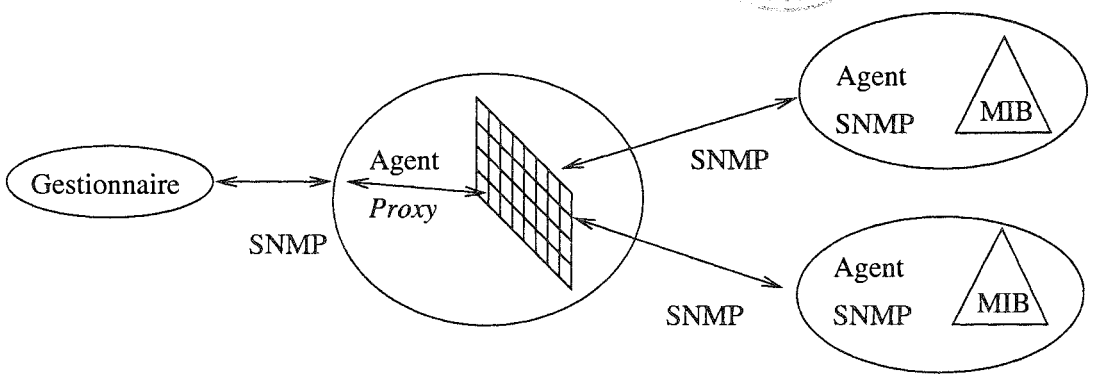


FIG. 3.1 – *Les places du tableur SNMP*

de contrôle est modifiable [KSS96], offrant ainsi la possiblité de créer de nouvelles relations entre des objets gérés et de tirer parti de ces relations grâce au langage de script des parties de contrôle [KSSZ97].

3.2.2 Les tables dynamiques

Une autre expérience, qui vise à réaliser l'équivalent des vues d'une base de données relationnelle sur les tables de variables de l'approche SNMP, a été proposée dans [AY95]. Une telle vue est une table qui est calculée à la demande, à partir des tables existantes. Dans la solution adoptée, le calcul des vues est effectué par l'agent, sur les tables qu'il contient dans sa MIB. Deux nouvelles MACROs ASN.1 sont définies; celle permettant de définir la structure des tables correspondantes aux vues; la macro `VIEW--TYPE` et celle pour définir les calculs qui construisent les vues; la macro `VIEW-FUNCTION`. La figure 3.2 montre un exemple d'utilisation de ces MACROs. La table `viewAtmIfTable` est définie (ligne 1) par le type de son entrée (ligne 2) et le champ dans ce type qui servira d'index à la table (ligne 3). Comme tout objet de la gestion SNMP, les types d'objets sont identifiés (ligne 4). La définition de l'entrée est également faite avec la macro `VIEW-TYPE` (ligne 5), elle consiste en une séquence ASN.1 (lignes 6 à 8). Chaque champ est également défini avec cette MACRO; par exemple, le champ `vIfIndex` (ligne 9), de type entier (ligne 10) et dont la valeur sera définie au moment où on désire y accéder, par la fonction de calcul des vues `func_vIfIndex` (ligne 11). Cette fonction est définie avec la MACRO `VIEW--FUNCTION` (ligne 13). La spécification du calcul est similaire à du SQL[30]. L'expression de l'exemple fait correspondre au champ `vIfIndex` le champ *ifIndex* d'une table réelle[31] (ligne 14). La totalité du champ de cette table n'est pas intégralement prise en compte; seules les valeurs qui sont dans une ligne où un autre champ de la table réelle vérifient une condition (ligne 15). Ici, le champ `ifType` doit être à la valeur 37 pour que la valeur de `ifIndex` sur la même ligne soit dans le champ `vIfIndex` de la vue.

3.3 Relations dans des modèles généraux

3.3.1 Service de relations de CORBA

Dans l'architecture CORBA, il existe plusieurs spécifications de classes liées aux services [Gro97] que peut rendre un système CORBA comme par exemple la persistance des objets ou l'émission d'événe-

30. Simple Query Langage [ISO92j]
31. la table `ifTable` des interfaces réseaux dans la MIB-II [MR91]

```
(1) viewAtmIfTable VIEW-TYPE                (9)  vIfIndex VIEW-TYPE
(2)   SYNTAX SEQUENCE OF vAtmIfTableEntry (10)    SYNTAX INTEGER
(3)   INDEX {vIfIndex}                     (11)   COMPUTED-BY func_vIfIndex
(4)   ::= {view 1}                         (12) ::= {vAtmIfTableEntry 1}

(5) vAtmIfTableEntry VIEW-TYPE             (13) func_vIfIndex VIEW-FUNCTION
(6)   SYNTAX SEQUENCE {                    (14)   SELECT ifIndex[SELF-INDEX]
(7)      vIfIndex INTEGER                  (15)   WHERE
(8)      vIfSpeed Gauge }                  (16)     ifType[SELF-INDEX] = 37
```

FIG. 3.2 – *Spécification d'une vue de table SNMP*

ments. Les instances d'objets CORBA ont chacunes une référence qui permet d'y invoquer des requêtes de services (i.e. des appels de méthodes). Les relations qui existent entre ces instances sont représentées par des attributs dont le type est celui des références d'instances d'objet CORBA. Un service de relations (chapitre 9 de [Gro97]) a été défini pour apporter de nouveaux services qui n'étaient pas directement accessibles avec la seule notion de référence. Les objets qui offrent le service de relations sont définis dans différents modules (unité de spécification en IDL comprenant des définitions d'interfaces de classes d'objets, de types et de variables) suivant le niveau de service.

Un premier niveau (le niveau des relations de base) définit la notion de relation comme étant un groupe d'au moins deux objets, chacun étant dans un rôle de la relation. Une relation est définie comme la composition d'au moins deux rôle, un même rôle ne peut être réutilisé pour définir une autre relation. Chaque rôle définit lui même, en plus de la classe de l'objet, des nombres entiers positifs ou nuls (ou cardinalités) qui précisent les nombres de relations auxquelles peut participer un objet dans ce rôle. La figure 3.3.a montre deux exemples de relations. La relation `R` possède trois rôles; `Ra`, `Rb` et `Rc`, dont les classes d'objets CORBA sont respectivement `A`, `B` et `C`. Les cardinalités indiquent qu'un objet dans le rôle `Ra` peut participer à une ou deux relations `R`, alors que les objets dans les rôles `Rb` et `Rc` ne doivent participer qu'à une seule relation `R`. Sur la base de ces concepts, le module de premier niveau spécifie des interfaces de classes pour la notion de relation et celle de rôle. Une relation sera représentée par des instances de la classe `Relationship` qui référencent des instances de la classe `Role`, elles même référençant chacune une instance d'objet; ce sont ces objets référencés qui sont en relation. Les méthodes définies dans la classe `Role` permettent de parcourir les relations, les autres rôles et les autres objets reliés à une instance de la classe `Role`. Une instance de relation est créée avec en paramètre les instances de chaque rôle. Une instance d'un rôle peut participer à plusieurs instanciations de relations de la même classe. La figure 3.3.b montre des instanciations des relations précédentes. On remarque le rôle `ra1` qui a été utilisé pour instancier les deux relations `r1` et `r2` et l'objet `a1` qui sert aux trois relations. Une fois créée, la relation ne peut perdre ou gagner une instance de rôle, sauf au cours des opérations permettant la destruction des relations ou des rôles. Deux classes d'objets permettent de générer (d'instancier) des instances des classes de relations et de rôle; les classes `RelationshipFactory` et `RoleFactory`. Les instances de ces classes ont un attribut qui détermine la classe de relation ou de rôle dont elles produisent les instances. Il est à noter que ce sont les intances de la classe `RelationshipFactory` qui doivent, lors de la génération d'une instance de relation, veiller à ce que la cardinalité maximale des rôles ne soit pas violée. Ceci s'explique car une relation ne peut pas recevoir de nouveaux rôles (et donc de nouveaux objets) au cours de son existence. En revanche, le respect de la cardinalité minimale est contrôlé par les rôles eux même. Une violation de la cardinalité minimale d'une instance de rôle peut être considérée comme une situation transitoire, où de nouvelles relations sont encore à créer avec l'instance de rôle.

Dans ce niveau de service, des classes abstraites `Relationship` et `Role` sont définies et toutes les classes de relations et de rôle du service CORBA devront être dérivées de ces classes. Cela signifie également que toutes les relations et les rôles d'un système CORBA peuvent être vues comme des ins-

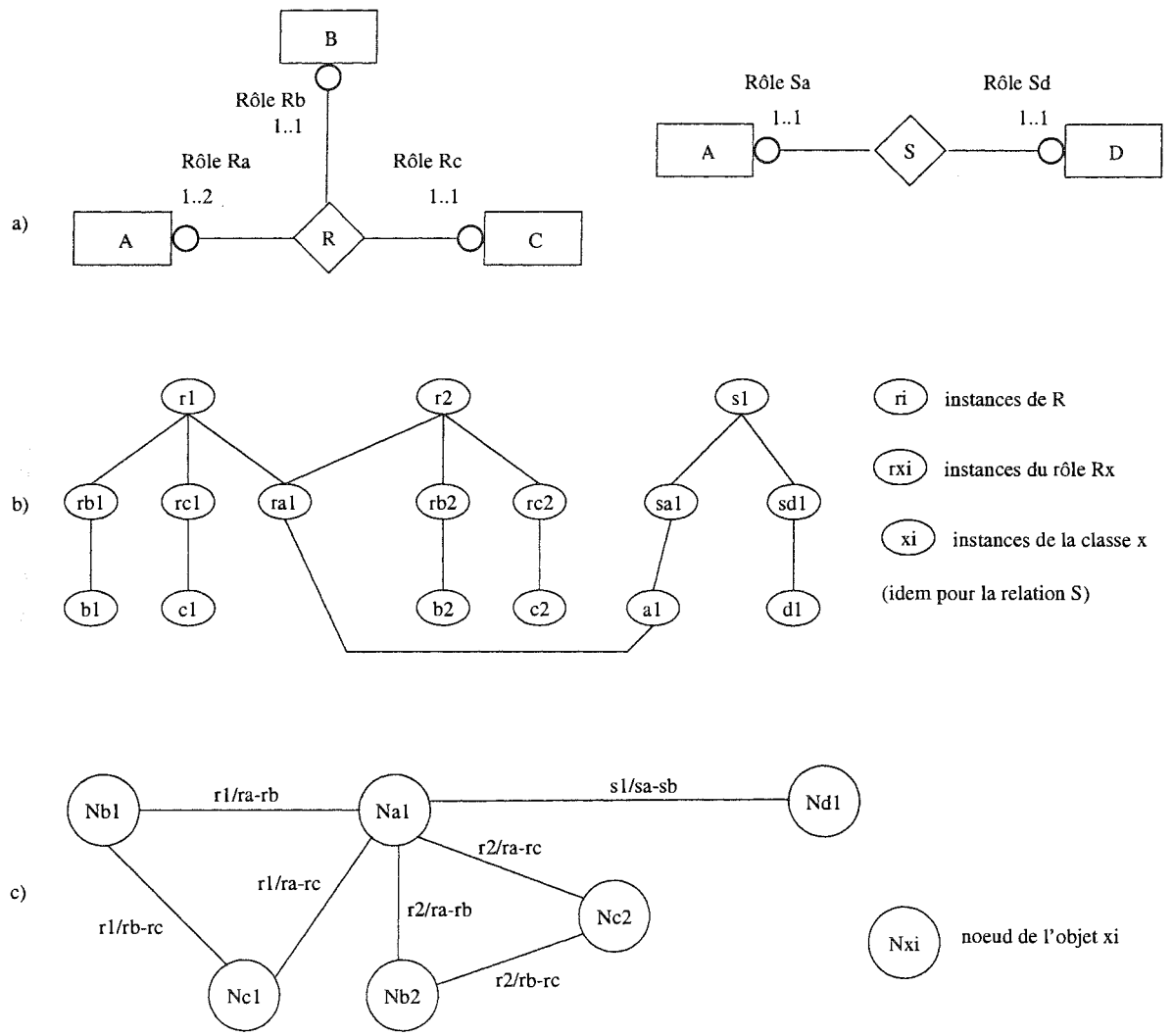


FIG. 3.3 – *Service de relation CORBA*

tances de ces deux classes abstraites. C'est sur cette vision homogène qu'est conçue le niveau suivant de service de relation de CORBA.

Le deuxième niveau de service est le niveau du graphe des objets reliés. Il permet de parcourir les objets en suivant les relations qu'ils ont entre eux. Pour cela, ce niveau utilise les notions du niveau précédent pour modéliser un graphe d'objet correpondant au graphe où les noeuds sont les instances d'objet CORBA et les arêtes sont les instances de relation CORBA. La classe `Node` modélise un noeud, elle est constituée d'une classe d'objet et d'un ensemble de classes de rôles. Une instance de noeud contient une instance d'objet et une instance de chaque rôle auxquel il participe. Les instances de rôle peuvent être dynamiquement ajoutées et enlevées, traduisant la participation de l'objet à des relations. Les arêtes entre les noeuds correspondent aux relations entre les rôles. Le graphe d'objets issus des relations précédentes est montré dans la figure 3.3.c.

Une classe d'objet (la classe `Traversal`) est définie spécialement pour parcourir un graphe de noeuds; les instances en sont générées (par une instance d'une classe `TraversalFactory`) avec une instance de noeud en paramètre. Le but de ces objets est de parcourir toutes les arêtes liées au noeud de départ, d'en récupérer les noeuds associés et de recommencer l'opération sur ceux-ci. L'arrêt du processus se fait lorsqu'il n'y a plus de nouveaux noeuds (durant le parcours les éventuels cycles sont surveillés). Une instance de la classe `Traversal` peut effectuer une sélection (ou critère de parcours) sur les noeuds ou les arêtes afin de contrôler le parcours.

Ce type de service est par exemple utile pour demander une copie d'un objet en copiant également les objets qui en dépendent. Le critère de parcours pourrait alors être la représentation des arêtes par une relation particulière : la `CosReference`, qui est reprise à la figure 3.4. La définition montre que l'interface hérite de l'interface générale des relations (lignes 3 et 4) et que chaque rôle (lignes 5 et 6) hérite des rôles définis dans le niveau du graphe des objets. La sémantique des rôles est exprimée dans

```
(1) #include <Graphs.idl>
(2) module CosReference {
(3)         interface Relationship:
(4)               ::CosRelationships::Relationship {};
(5)         interface ReferencesRole : CosGraphs::Role{};
(6)         interface ReferencedByRole : ::CosGraphs::Role{};
(7)};
```

FIG. 3.4 – *Spécification de l'interface de la relations de référencement*

des valeurs des attributs de l'interface `RelationshipFactory` et `RoleFactory`, comme le degré de la relation et les cardinalités des rôles (ici, le degré est de deux et les rôles peuvent avoir n'importe quelle cardinalité positive ou nulle).

La cohabitation de l'architecture CORBA et de l'environnement de gestion OSI sera étudiée plus en détail dans la section 5.2.4 du chapitre 5.

3.3.2 TINA-C et le quasi GDMO GRM

Dans l'approche TINA-C (cf. §2.5.3 chap. 2), le modèle de l'information est composé d'un modèle pour les objets gérés et d'un pour les relations. Les formalismes de description de ces modèles, respectivement le Quasi-GDMO et le Quasi-GRM, ont de fortes similitudes syntaxiques avec GDMO et GRM. Pour ce dernier, qui sera vu dans le chapitre 4, les différences avec le Quasi-GRM sont nombreuses. En fait, les relations exprimées avec le Quasi-GRM sont l'équivalent des relations du service de relation de CORBA (CORBA est d'ailleurs une architecture reconnue par TINAC pour son environnement de traitement). En effet, un type de relation Quasi-GRM spécifie une relation entre uniquement deux classes d'objets Quasi-GDMO. Une instance de relation ne relie que deux instances d'objets. Contrairement à

CORBA, mais dans les principes du GRM, un type de relation n'explicite pas le nom des classes d'objets reliées et ceci est fait dans une spécification de ROLE BINDING, séparée de celle du type de relation. Il peut exister plusieurs ROLE BINDING pour une classe de relation. D'autres caractéristiques comme des attributs qui représentent l'état de la relation, un comportement et des opérations de relation, sont repris du modèle générique de relation de l'ISO vu au chapitre suivant. En résumé, on peut dire que le Quasi-GRM est une restriction du GRM visant à rendre possible la traduction de tous types de relations du premier vers les relations du service CORBA. La figure 3.5 donne un exemple de type de relation en Quasi-GRM; le type connexion (ligne 1) est dérivé (ligne 2) d'un autre type de relation (le concept de dérivation de relations sera vu avec le GRM). La définition des types de relation est modulaire (ligne 3) et un module (*package*) doit contenir la spécification du comportement (ligne 4) de la relation et la signature des opérations de la relation; ici (ligne 5) seule la pré-condition de la création est informellement exprimée (ligne 6). TINA impose une organisation en invariants, pré- et post-conditions des spécification du comportement, mais n'impose pas de langage formel particulier (des études sont faites en Z ou SDL). Le type de relations doit alors définir ses deux rôles (lignes 7,8 et 9,10) qui vont acceuillir chacun une instance d'objet TINA. Les cardinalités (lignes 8 et 10) fixent à un le nombre de variables de ce type de relation auquel les objets peuvent participer. La spécification de ROLE BINDING ne fait qu'associer une classe d'objet TINA pour chaque rôle, nous ne l'avons pas reprise ici.

```
(1) connexion RELATIONSHIP TYPE
(2)    DERIVED FROM trafic;
(3)    CHARACTERIZED BY connexionPackage
(4)    BEHAVIOUR connexionBehaviour BEHAVIOUR DEFINED AS
(5)     create-connexion(x1: source, x2: destination) : y : connexion
(6)      PRECONDS: "x1 et x2 doivent être opérationnel"
(7)     ROLE source
(8)        ROLE-CARDINALITY-CONSTRAINT 1..1
(9)      ROLE destination
(10)       ROLE-CARDINALITY-CONSTRAINT 1..1
```

FIG. 3.5 – *Spécification d'un type de relation en Quasi-GRM*

3.3.3 WBEM

La prise en compte des relations entre les objets gérés de cette approche est formalisée explicitement par des classes particulières: les classes d'**Associations**. Dans la spécification d'un modèle de gestion, les classes d'Associations sont vues au même niveau que celui des classes d'objets, une classe d'association devant toujours se trouver entre au moins deux classes d'objets. La notation graphique UML peut être utilisée pour générer des spécifications MOF. La définition des classes d'associations doit contenir au moins deux attributs dont le type est une ou plusieurs références d'instances d'objets gérés. La spécification d'une classe de relation permet de définir pour chaque attribut deux types de contraintes:

- un nombre minimum et un nombre maximum d'instances d'objet qui pourront être référencées dans une instance de l'association;
- une **multiplicité** qui est soit single-valued, soit multi-valued suivant que les instances d'objet peuvent être référencées par une ou plusieurs instances de la même classe d'association (par le même attribut).

Etant des classes d'objets, les classes d'associations peuvent hériter d'autres classes d'associations (héritage simple uniquement), mais aussi d'autres classes d'objets. Une classe d'association peut donc contenir des attributs (autres que ceux de références) et des méthodes. La figure 3.6 donne un exemple d'une association nommée connexion (ligne 1) qui relie exclusivement un objet de la classe terminationPoint comme source (ligne 3 et 4) et destination (lignes 5 et 6) d'un trafic de communication.

Ces objets ne peuvent par ailleurs ne participer qu'à une seule association de cette classe. Le mot clé ASSOCIATION précédant le nom de la classe (ligne 1) marque bien le fait qu'une classe d'association est également une classe d'objet.

```
(1) [ASSOCIATION] CLASS connexion
(2)
(3) [Min(1), Max(1), Multiplicity("single-valued")
(4)                              TerminationPoint REF source;
(5) [Min(1), Max(1), Multiplicity("single-valued")
(6)                              TerminationPoint REF destination;
(7)
(8);
```

FIG. 3.6 – *Spécification d'association MOF*

3.4 Les relations dans la gestion OSI

Plusieurs propositions existent pour prendre en compte les relations entre les instances d'objets gérés définis en GDMO. Nous les présentons dans la suite et terminerons par le modèle général de relation.

3.4.1 Les attributs de relations

Les attributs de relation sont des attributs définis en GDMO dont le type est celui des identificateurs d'instances d'objets gérés (i.e. leur *Distinguish Name*). Une fonction de gestion [ISO92b] spécifie un ensemble d'attributs de relation qui pourront être utilisés dans les classes d'objets gérés. Les relations que peuvent représenter ces attributs sont des relations entre deux classes. Aussi, les attributs sont regroupés par couples et pour chacun, les attributs doivent être dans des objets gérés distincts. Une sémantique est associée à ces couples dont la liste exhaustive est donnée dans la table 3.1, par exemple une relation de *Backup* (i.e. sauvegarde ou duplication), indiquant qu'un objet géré est une copie d'un autre objet géré, est représentée d'une part avec l'attribut `backedUpObject` qui doit être contenu dans l'objet géré de sauvegarde et qui doit référencer l'objet géré à sauvegarder. D'autre part, l'attribut `backUpObject` doit être contenu dans ce dernier objet géré et référencer celui qui en est une sauvegarde. Cependant, l'utilisation des deux types d'attributs n'est pas une nécessité pour leur utilisation et le modèle des attributs de relation accepte les relations symétriques (un type attribut de relation dans chaque classe d'objet) asymétrique (un type d'attribut de relation dans une des classe). Les attributs de relation sont une bonne

Relation	attribut dans un objet A	attribut dans un objet B
Peer	`peer`	`peer`
Service	`userObject*`[a]	`providerObject*`
FallBack	`primary*`	`secondary*`
Backup	`backUpObject`	`backedUpObject`
Group	`member*`	`owner*`

TAB. 3.1 – *Les attributs de relation normalisés*

[a] le symbole * repère les attributs multivalués

solution pour modéliser des relations entre deux classes mais elle a des limites. Outre la limitation à des couples de classes, l'utilisation des attributs de relation normalisés ne permet pas d'ajouter de nouvelles

relations dynamiquement entre des objets gérés existants (cela nécessiterait l'ajoût de nouveaux attributs dans les instances).

3.4.2 Autres modélisations de relations

Dans [KK91], une classe d'objet géré GDMO est définie pour représenter des relations entre objets gérés. Les instances de cette classe contiennent une structure de donnée représentant un tableau à deux dimensions : une matrice carrée, dont chaque ligne et chaque colonne représente un objet géré; les objets des lignes sont les même que ceux des colonnes. Chaque case du tableau est de type booléen; valant faux s'il n'y a pas de relation entre l'objet représentant la ligne et celui représentant la colonne et valant vrai sinon. Différents parcours des objets suivant leurs relations peuvent alors être effectués par l'objet détenteur du tableau; par exemple, trouver tous les objets reliés à un objet particulier, ou tester si une relation directe ou indirecte existe entre deux objets. Plusieurs classes d'objets détenteurs peuvent être spécifiées selon la sémantique que l'on désire associer aux relations marquées dans le tableau (par ex. une classe pour les relations de contenance logiques et une classe pour les contenances physiques). L'inconvénient de cette solution est la création systématique d'un objet géré pour représenter des relations, de plus la taille de la matrice croît de manière exponentielle avec le nombre d'objets reliés. Enfin le modèle sous-jacent à la définition des relations est celui d'un graphe d'instances d'objets, correspondant à plusieurs relations individuelles, ce qui est différent des spécifications habituelles de relations entre des classes d'objets.

Dans le même ordre d'idée, la tableur SNMP vu précédement pourrait être implanté par une classe d'objet géré.

Des modèles de gestion divers spécifient les relations d'une manière *ad hoc*. L'ATM-forum par exemple schématise pour chacune de ses classes d'objets [Tec94], l'ensemble des classes avec lesquelles une relation existe. Cela a l'avantage d'éviter la manipulation d'un schéma général, où toutes les classes sont représentées et facilite la lecture de ces spécifications.

Avant le chapitre suivant qui détaille la proposition de l'ISO pour la spécification des relations, nous citons ici un travail qui a contribué à cette proposition. Dans [Cle93] un modèle et un formalisme sont proposés pour spécifier les relations entre objets gérés. Si le modèle s'apparente plus à un modèle E/A, le formalisme est très proche de celui de l'ISO. Les relations sont toutes représentées de la même manière; pour chacune, un objet géré contient des attributs de relation qui référencent les objets gérés participant à la relation (c'est également dans le principe de CORBA). La notion de service de relation est définie dans [Cle94], où des opérations de relation sont traduites par des opérations d'objet, sur les objets utilisés pour représenter les relations.

3.5 Conclusions

Ce chapitre a présenté différents moyens pour modéliser des relations entre des objets de gestion et plus généralement, des objets distribués. On peut constater que les approches de gestion de réseaux ou d'environnements distribués sont nombreuses à désirer exprimer et utiliser la notion de relation dans leurs spécifications et dans leur exploitation. Le concept de relation séparé de celui d'objet peut se traduire, dans certaines de ces approches, par l'utilisation du modèle objet ou, dans d'autres approches, par la conception d'un nouveau modèle dédié aux relations et adapté aux objets du modèle initial. Les avantages d'un moyen permettant de spécifier des relations de manière homogène sont d'une part l'abstraction de la notion de relation qui facilite les spécifications formelles des modèles d'information, d'autre part la mise à disposition pour des applications de cette abstraction afin que leurs traitements puissent être exprimés avec le même niveau d'abstraction.

Chapitre 4

Le Modèle de relation générique

Le modèle de relation générique (GRM[32]) [ISO95] est la porposition de l'ISO pour modéliser toutes les relations entre les objets gérés. Le document fournit d'une part un modèle pour concevoir des relations entre ces objets, d'autre part une notation semi-formelle qui permet de les spécifier. Le GRM distingue le concept abstrait de relation des moyens de représentation utilisés pour implanter ces concepts dans les objets. Dans cette section, la présentation du GRM est faite en regroupant modèle et formalisme pour ces deux visions des relations.

4.1 Les classes de relations gérées

Une **relation gérée** existe entre des objets gérés. Généralement, cette relation entraîne dans les objets, un comportement qui est différent (pas forcément un sous-ensemble) de celui qu'ils auraient s'ils n'étaient pas en relation. Cette variation de comportement d'un objet dans une relation est due à la présence des autres objets dans la même relation et l'on doit pouvoir isoler ces objets en fonction de leurs influences sur les autres objets. Cette distinction constitue les **rôles** d'une relation gérée. Les objets appartenant au même rôle (ou les **participants** à un rôle) ne peuvent pas être distingués les uns des autres. Le comportement doit donc s'exprimer aux niveaux des rôles. Par exemple une relation peut relier des objets dans un rôle `contenant` et dans un rôle `contenu` avec la conséquence que les participants au rôle `contenu` doivent être supprimés si le participant au rôle `contenant` est lui même supprimé. Toutes les relations gérées qui relient des objets en ayant les mêmes rôles et les mêmes dépendances de comportement entre les objets de ces rôles forment une **classe de relation gérée**. Une telle classe définit les caractéristiques suivantes qui sont valables pour toutes les instances de cette classe :

- des opérations que l'on peut appliquer à une relation gérée et des notifications qui peuvent être émises par une relation gérée;
- le comportement des relations gérées (i.e. invariants de relation gérée, pré- et post-conditions de ses opérations);
- des attributs **qualifiants** d'une relation gérée. Ces attributs permettent d'ajouter des informations propres à une relation, sans spécifier dans quels objets gérés ces informations doivent être contenues;
- les rôles dans lesquels les objets gérés participent. Les rôles peuvent avoir des contraintes qui leur sont propres:
  - une classe GDMO avec laquelle celles des participants doivent être compatibles;
  - le nombre de participants tolérés (ou cardinalité permise d'un rôle);
  - le nombre de participants nécessaires (ou cardinalité requise d'un rôle);

32. General Relationship Model

- la possibilité pour les participants, de quitter le rôle ou, pour des objets gérés, d'entrer dans le rôle, durant l'existance de la relation gérée;
- le nombre de relations gérées de la même classe auxquelles un objet géré peut participer dans ce rôle (ou cardinalité permise d'un rôle dans la relation).

La cardinalité requise d'un rôle doit spécifier un sous-ensemble non strict de sa cardinalité permise. Cette première impose une caractéristique minimale que le rôle doit être en mesure de respecter, même si le nombre courant de participants dans le rôle peut être en dehors de ses bornes (mais toujours dans la cardinalité permise).

Si l'on compare les classes d'objets gérés et les classes de relations, seule les opérations définies dans ces dernières constituent l'interface de gestion. Si une relation gérée était toujours instanciée comme un objet géré (comme dans WBEM ou dans [Cle93]), ces opérations de relation seraient traduites en autant d'actions d'objet. Nous verrons plus loin qu'il existe plusieurs manières d'instancier une relation gérée et que chaque opération de relation n'est pas forcément traduite par une action d'objet. Les classes de relations gérées sont des spécifications abstraites qui se veulent indépendantes de la manière dont elles seront **représentées** dans l'environnement de gestion OSI composé uniquement d'objets gérés.

Une classe de relation gérée peut être une extension d'une ou plusieurs classes, signifiant que les relations gérées de la première ont les caractéristiques des dernières et en rajoutent de nouvelles, ou en redéfinissent des existantes, en veillant à rester compatibles. Cette compatiblité d'une sous-classe de relation gérée envers ses super-classes est respectée lorsque l'on suit les règles ci dessous:

- des rôles peuvent être ajoutés, mais on ne peut en enlever un des super-classes;
- les attributs et les opérations suivent la règle précédente, sauf qu'il est interdit de définir de nouvelles notifications de relation;
- il n'y a pas de syntaxe pour spécifier d'éventuels arguments des opérations de relation, donc pas de règles de redéfinition pour ces arguments;
- le type des attributs qualifiants d'une relation ne peut être modifié dans les sous-classes;
- le comportement peut être redéfini:
  - les préconditions des opérations sont les disjonctions des préconditions des opérations des super-classes;
  - les post-conditions des opérations sont les conjonctions des post-conditions des opérations des super-classes;
  - les invariants de la classes sont les conjonctions des invariants des super-classes;
- un rôle peut être redéfini par:
  - une spécialisation de la classe GDMO compatible;
  - l'intersection des cardinalités permises des rôles des super-classes et une restriction de cette cardinalité (sans restreindre la cardinalité requise);
  - l'union des cardinalités requises des rôles des super-classes et une extension de cette cardinalité (sans excéder la cardinalité permise);
  - l'instersection des cardinalités de rôle de relation des super-classes et une restriction de cette cardinalité.

La figure 4.1 montre le formulaire de spécification des classes de relations gérées. Une classe de relation gérée possède un nom (ligne 1). Sa spécification peut réutiliser celles d'autres classes de relations gérées (lignes 2 et 3). Au moins un formulaire de comportement doit être référencé (ligne 4). Il s'agit de formulaires de type `BEHAVIOUR` définis dans GDMO (le comportement des classes de relations gérées est donc informel). Les opérations de la classe sont énumérées (ligne 5) par une série de constructions `supported` (fig. 4.2). Les attributs qualifiants d'une classe de relation (ligne 6) sont des attributs GDMO. Chaque rôle de la classe de relation est ensuite spécifié (ligne 7), la figure 4.3 en donne le détail. Enfin, toute classe de relation gérée doit avoir un numéro d'enregistrement unique (ligne 8).

(1) <*relationship-class-label*> **RELATIONSHIP CLASS**
(2) [**DERIVED FROM** <*relationship-class-label*>
(3) [, <*relationship-class-label*>]* ;]
(4) **BEHAVIOUR** <*behaviour-label*> [, <*behaviour-label*>]* ;
(5) [**SUPPORTS** supported [, supported];]
(6) [**QUALIFIED BY** <*attribute-label*> [, <*attribute-label*>] * ;]
(7) [<*role-specifier*] * ;
(8) **REGISTERED AS** {<*object-identifier*>} ;

FIG. 4.1 – *Le formulaire de classe de relation*

Le GRM définit des opérations génériques pour les relations. Leur syntaxe est donnée dans la figure 4.2. La table ci-dessous explicite la sémantique qui est associée à chacune de ces opérations.

Nom de l'opération	sémantique associée
ESTABLISH	créé une nouvelle instance de la classe de relation
TERMINATE	supprime une relation gérée
QUERY	interroge une relation gérée (généralement un rôle ou un attribut donné)
NOTIFY	émission d'information à l'initiative d'une relation gérée
USER-DEFINED	toute autre opération de relation gérée dont le sens est défini par l'utilisateur

Une classe peut définir plusieurs opérations d'un même type, un identificateur sera alors nécessaire pour les distinguer.

supported –>
(1) **ESTABLISH** [operation-name]
(2) |**TERMINATE** [operation-name]
(3) |**QUERY** [operation-name]
(4) |**NOTIFY** [notification-name]
(5) |**USER-DEFINED** [operation-name]

operation-name –> <*identifier*>
notification-name –> <*identifier*>

FIG. 4.2 – *Les opérations de relations*

La définition des rôles suit le formulaire de la figure 4.3. Un rôle a un nom (ligne 1). La classe GDMO à laquelle doivent être compatibles tous les participants au rôle est spécifiée (ligne 2), sinon la classe `top` est la valeur par défaut. Puis ce sont les cardinalités permises et requises de rôles (lignes 3 et 4). Deux nouveaux types d'opérations de relation gérée peuvent être spécifiés pour un rôle; l'opération qui ajoute un nouvel objet géré comme participant au rôle (ligne 5) et l'opération qui enlève un participant du rôle (ligne 6). Tout comme les opérations de relation précédentes, il est possible de définir plusieurs opérations du même type pour un même rôle, rendant nécessaire l'utilisation d'identificateurs. Enfin (ligne 8) un rôle peut avoir un numéro d'enregistrement qui permet d'identifier ce dernier. Toutefois les rôles ne sont pas réutilisables, ormis par héritage, dans d'autres classes de relations gérées.

role-specifier –>
(1) **ROLE** role-name
(2) [**COMPATIBLE-WITH** <*class-label*>]
(3) [**PERMITTED-ROLE-CARDINALITY-CONSTRAINT** *type-reference*]
(4) [**REQUIRED-ROLE-CARDINALITY-CONSTRAINT** *type-reference*]
(5) [**BIND-SUPPORT** [operation-name]]
(6) [**UNBIND-SUPPORT** [operation-name]]
(7) [**PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT** *type-reference*]
(8) [**REGISTERED-AS** object-identifier]

role-name –> <*identifier*>
operation-name –> <*identifier*>

FIG. 4.3 – *Le formulaire de rôle*

4.2 Les représentations des relations

Les classes de relations gérées permettent de spécifier des relations entre classes d'objets gérés par une abstraction de la notion de relation en termes de rôles, d'opérations, d'attributs et de comportements. La deuxième étape [Sa93] consiste à spécifier entre quelles classes d'objets gérés les classes de relations gérées sont utilisables. Ceci s'accompagnant de la spécification des moyens mis en oeuvre pour représenter l'existence de la relation dans les objets gérés et la traduction des opérations de relation en opérations d'objet (ou opérations système).

Une même classe de relation gérée peut avoir plusieurs **représentations** suivant les classes d'objets gérés qui sont effectivement reliées par la relation gérée. Une représentation doit spécifier:

- la classe de relation gérée représentée;
- un comportement;
- pour chaque rôle :
  - une liste de classes d'objets gérés dont les instances peuvent participer aux relations gérées de la classe représentée (ces classes doivent être compatibles avec celle définie dans la classe de relation);
  - éventuellement un moyen de représentation;
  - éventuellement les attributs qualifiants de la relation gérée représentée qui sont en fait des attributs des classes d'objets gérés pour le rôle;
- la traduction de chaque opération de relation en une ou plusieurs opérations système, chacune de ces dernières étant dirigée sur les objets gérés d'un rôle donné.

Une classe de relation gérée peut être représentée par quatre moyens différents, la figure 4.4 illustre par un exemple chacun de ces moyens appliqués à la même relation gérée issue de la classe `R` (fig. 4.4.a). La même instance, composée des objets `a1` et `a2` dans le rôle `Ra` et des objets `b1`, `b2` et `b3` dans le rôle `Rb`, est représentée suivant les quatre modes de représentation qui sont:

- des attributs de relation (du même type que ceux vus dans le §3.4.1) dans les participants d'un rôle et qui réfèrent aux participants d'un autre rôle (fig. 4.4.b);
- des corrélation de noms (formulaire `NAME BINDING`, fig. 4.4.c);
- un objet géré, dédié à la représentation des relations gérées de la classe représentée, possédant un attribut de relation pour chaque rôle (l'objet de la classe `OR` dans la figure 4.4.d);
- des actions de participants d'un rôle qui retournent les participants d'un autre rôle (les actions `actionB` qui retournent les objets `b1` à `b3` dans la figure 4.4.e).

La traduction des opérations de relation en des opérations système est dépendante de la représentation.

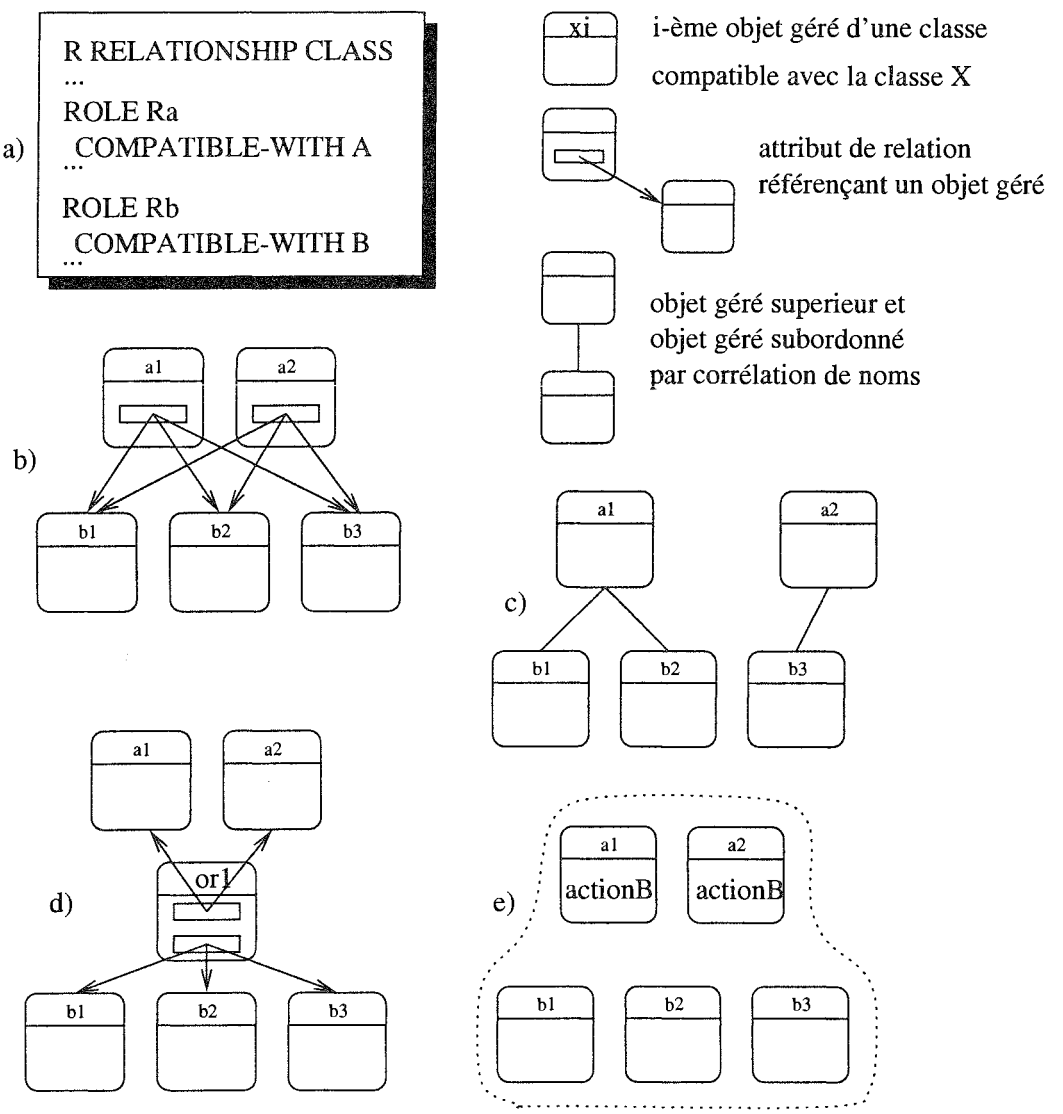


FIG. 4.4 – *Représentations des classes de relations gérées*

Suivant cette dernière, seules certaines opérations système seront acceptables pour traduire des opérations de relation. Par exemple, dans la représentation de la figure 4.4.b, l'ajoût de nouveaux participants à ce dernier par une opération BIND doit être traduite par une opération ADD ou REPLACE[33] sur les attributs des participants au rôle Ra. La spécification d'une opération de relation n'implique cependant pas la possibilité de gérer cette relation depuis un Gestionnaire OSI par l'intermédiaire de ses opérations. En reprenant l'exemple précédent, les attributs de relation peuvent être en lecture seule et donc seules des modifications à l'origine de la ressource modélisée peuvent entraîner une modification des attributs de relation et l'opération BIND sera réalisée automatiquement. Averti d'un tel phénomène, un Gestionnaire ne peut plus que vérifier les contraintes de comportement de la relation (par ex. le non dépassement de la cardinalité du rôle Rb).

Le formulaire de la figure 4.5 est utilisé pour spécifier les représentations des classes de relations gérées. Une représentation possède un nom (ligne 1) et se réfère toujours à exactement une classe de relation gérée (ligne 2) qu'elle représente. Au moins un formulaire de comportement (du type BEHAVIOUR de GDMO) est obligatoire (ligne 3). Si la représentation utilise un objet géré dédié pour cette tâche, sa classe doit être indiquée (ligne 4) hors de tout rôle car elle sert à la représentation de tous les rôles. Un tel **objet de relation** peut également contenir des attributs qualifiants spécifiés dans la classe de relation gérée représentée (cf. fig. 4.1 ligne 6). La représentation détaillée des rôles (ligne 6) est montrée dans la figure 4.6 et la traduction des opérations de relation (lignes 7 et 8) se trouve à la figure 4.7.

(1) *<relationship-mapping-label>* **RELATIONSHIP MAPPING**
(2) **RELATIONSHIP CLASS** *<relationship-class-label>*;
(3) **BEHAVIOUR** *<behaviour-label>* [, *<behaviour-label>*]* ;
(4) [**RELATIONSHIP OBJECT** *<class-label>* [**QUALIFIES** *<attribute-label>*
(5) [, *<attribute-label>*]*];]
(6) role-mapping-specification [, role-mapping-specification]* ;
(7) [**OPERATIONS MAPPING** relationship-operation maps-to
(8) [, relationship-operation maps-to]*;]
(9) **REGISTERED AS** object-identifier ;

FIG. 4.5 – *Le formulaire de représentation des classes de relations gérées*

Pour chaque rôle, on doit spécifier (ligne 1) une liste de classes d'objets gérés GDMO, qui doivent toutes être compatibles avec celle spécifiée dans la classe de relation gérée représentée (cf. fig. 4.3 ligne 2) et qui sont les classes que peuvent avoir les participants au rôle (classes d'objets gérés **rattachées** au rôle). Ensuite, un moyen de représentation peut être spécifié (ligne 2) ainsi qu'une liste des attributs de la classe de relation gérée représentée (ligne 3) qui font partie des attributs des classes d'objets gérés rattachées au rôle.

Les représentations possibles sont (lignes 4 à 7):

- la corrélation de noms (ligne 4) et un formulaire de NAME BINDING doit être spécifié, de même (ligne 8) que la classe d'objet géré dans ce formulaire qui correspond à la classe des participants au rôle;
- un attribut de relation (ligne 5) qui doit faire partie de la définition de la classe de participants à un autre rôle et dont les valeurs désignent les participants au rôle courant;
- un attribut dans un objet de relation (ligne 6);
- une opération (ligne 7) qui doit faire partie de la définition de la classe de participants à un autre rôle et qui doit retourner les identificateurs des participants au rôle courant.

33. ces deux opérations étant transportées par le service CMIS: M-SET

role-mapping-specification –>
(1) **ROLE** role-name **RELATED-CLASSES** <*class-label*> [<*class-label*>]*
(2) [**REPRESENTED-BY** representation]
(3) [**QUALIFIES** <*attribute-label*> [<*attribute-label*>]*]

representation –>
(4) **NAMING** <*name-binding-label*> **USING** superiorOrSubordinate
(5) [**ATTRIBUTE** <*attribute-label*>
(6) [**RELATIONSHIP-OBJECT-USING-POINTER** <*attribute-label*>
(7) [**OPERATION**]

superiorOrSubordinate –>
(8) **SUPERIOR | SUBORDINATE**

FIG. 4.6 – *La représentation d'un rôle*

La traduction des opérations de relation en opérations système est spécifiée suivant la partie de formulaire de la figure 4.7. Pour chaque opération supportée par la classe de relation gérée représentée (cf. fig. 4.1 ligne 5), il doit y avoir une unique traduction. Une traduction est composée de la spécification de l'opération (lignes 1 à 7) et d'une suite d'opérations système (lignes 8 à 10) qui lui correspondent. Les opérations système correspondent généralement à un appel de primitive de service CMIS; le GET (ligne 11), par exemple sera émis avec la primitive M-GET. Les trois opérations système ADD, REMOVE et REPLACE (lignes 12 à 14) sont véhiculées par le service M-SET. On a vu précédemment que les opérations système sont dépendantes de la représentation, toutefois, dans le cas d'une notification de relation (ligne 6), seules des notifications d'objets gérés (ligne 18) peuvent la traduire .

4.3 Conception avec le GRM

Dans cette section, nous développons un modèle de gestion composé de classes et de représentations de relations gérées. Cet exemple sera réutilisé tout au long de ce document. Nous qualifions l'approche de modélisation suivie comme étant ascendante car nous nous fondons sur un modèle de gestion existant, composé de classes d'objets gérés. À partir de l'observation de ce modèle, nous dégageons des relations et les exprimons avec le GRM.

4.3.1 Le modèle générique d'information de réseau

La recommandation M.3100 de l'ITU-T [M3195], décrit un réseau de télécommunication avec des classes d'objets gérés. Plusieurs aspects sont pris en compte dans cette modélisation :

- la topologie du réseau avec son partitionnement en sous-réseaux et sa composition en noeuds (éléments gérés), équipements et logiciels, comme l'illustre l'exemple des figures 4.8.a et b,
- le trafic de communication, avec une hierarchie de couches réseaux (inspirée de la SDH[34] [Bar95]) entre éléments gérés (connexions et voies) et internes à ceux ci (brassages point à point et point à multipoint) qui sont illustrés dans les figures 4.8.b et c,
- la logistique de gestion du réseau avec des objets purement administratifs (unités d'organisations, alarmes spécifiques, etc).

Dans la suite, ce sont à partir des objets définis pour la gestion de la communication que nous allons concevoir des relations gérées. La figure 4.9 modélise les associations possibles entre les classes concer-

34. Synchronous Digital Hierarchy

relationship-operation –>
(1) **ESTABLISH** [operation-name]
(2) |**TERMINATE** [operation-name]
(3) |**BIND** [operation-name] [role-name]
(4) |**UNBIND** [operation-name] [role-name]
(5) |**QUERY** [operation-name] [role-name]
(6) |**NOTIFY** [operation-name]
(7) |**USER-DEFINED** [operation-name]

maps-to –>
(8) MAPS-TO-OPERATION systems-management-operation
(9) OF role-or-relObject [systems-management-operation
(10) OF role-or-relObject]*

systems-management-operation –>
(11) **GET** <*attribute-label*> [<*parameter-label*>]*
(12) |**ADD** <*attribute-label*> [<*parameter-label*>]*
(13) |**REMOVE** <*attribute-label*> [<*parameter-label*>]*
(14) |**REPLACE** <*attribute-label*> [<*parameter-label*>]*
(15) |**CREATE** <*class-label*> [<*parameter-label*>]*
(16) |**DELETE** [<*parameter-label*>]*
(17) |**ACTION** <*action-label*> [<*parameter-label*>]*
(18) |**NOTIFICATION** <*notification-label*> [<*parameter-label*>]*
(19) role-or-relObject –>role-name | **RELATIONSHIP OBJECT**

FIG. 4.7 – *La représentation des opérations de relation*

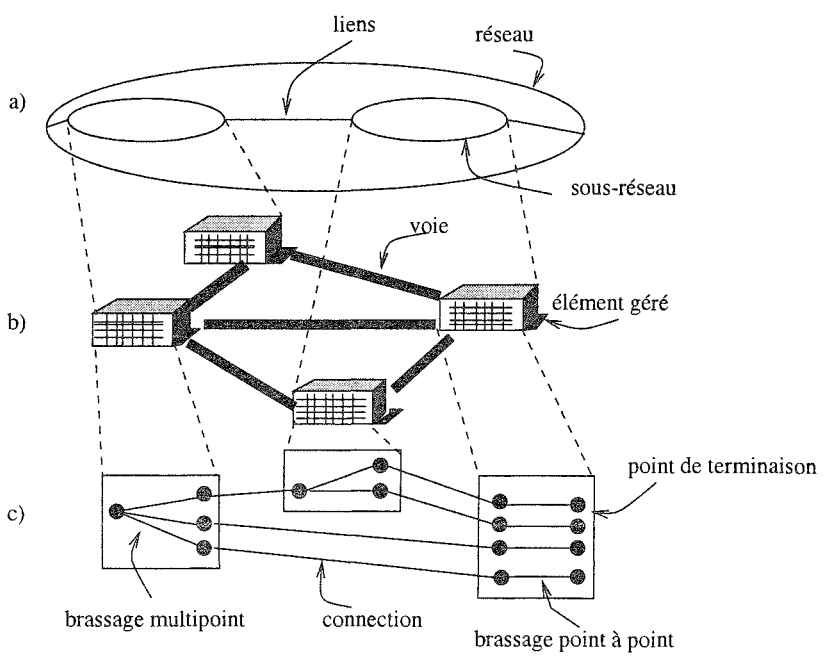


FIG. 4.8 – *Partitions d'un réseau de télécommunication*

nées par la gestion des canaux (classe `pipe`) véhiculant le trafic entre des points de terminaison (classe `terminationPoint`) dans des éléments gérés (classe `managedElement`) d'un réseau (classe `network`) et par la gestion des brassages (classes `fabric`, `mpCrossConnection` et `crossConnection`), internes à un élément géré entre des points de terminaison. Les associations sont étiquetées par

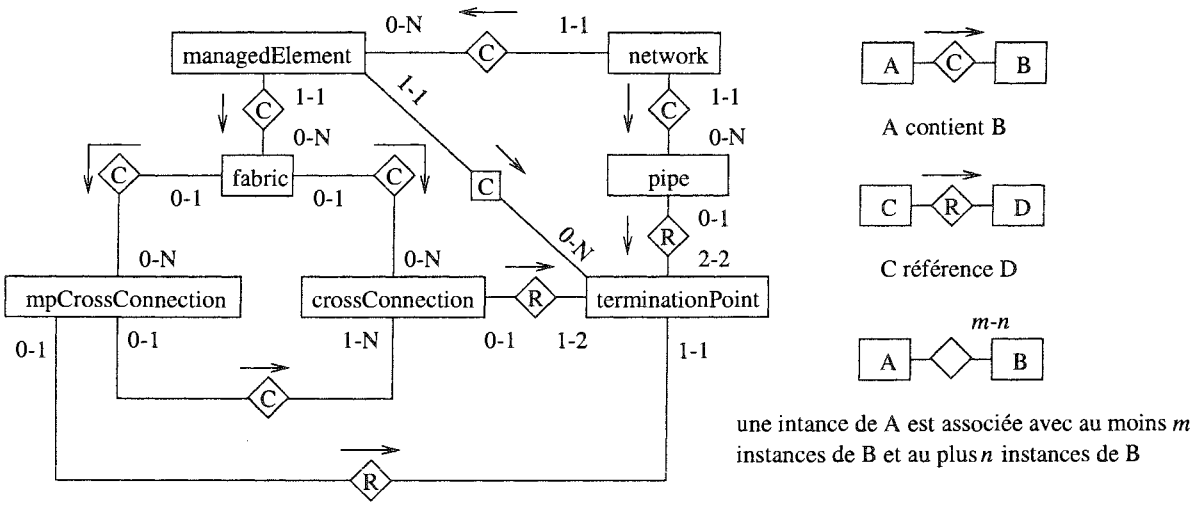


FIG. 4.9 – *Schéma E/A partiel de la recommandation ITU-T M.3100*

un `C` lorsqu'elles expriment la notion de contenance. Ces associations sont formalisées en GDMO par des corrélations de noms entre les classes. Le `R` exprime une référence entre les classes (i.e. un attribut de relation qui référence une ou plusieurs instances). Un tel schéma exprime les associations possibles entre deux classes. Il n'est pas suffisant pour formaliser des contraintes entre ces associations qui doivent alors être exprimées dans les classes elles-mêmes. L'exemple qui suit est une instanciation de ce schéma E/A où quelques unes de ces contraintes seront explicitées.

4.3.2 Exemple d'une configuration

La figure 4.10 schématise une configuration d'objets gérés (rectangles arrondis) reliés d'une part avec des corrélations de noms, d'autre part avec des attributs de relation. Les noms des attributs sont près des flèches (le nom est celui de l'attribut d'origine). Les attributs désignent des objets gérés ou sont à la valeur nulle (attribut `fromTermination` de la classe `crossConnection`).

Cette configuration est une instanciation du schéma E/A précédent et modélise deux types de trafics entre des points de terminaison : une connexion et un brassage point à multipoint. La connexion se fait entre les deux objets `ctp1` et `ctp2`, le trafic passant du premier vers le second. Le brassage fait transiter le trafic de `ctp2` vers `ctp3` et `ctp4`. Il apparaît que tous les attributs de relation impliqués dans la configuration n'ont pas été pris en compte dans le schéma E/A. Dans cet exemple particulier, des associations étiquetées `R` devraient être spécifiées en parallèle avec celles qui existent, mais dans le sens inverse.

Les contraintes relatives à la gestion de ces configurations ne sont pas visibles depuis le schéma E/A. La communication du trafic entre points de terminaison doit se faire à partir de points existants; elle est réalisée effectivement lors de la création des objets –dans notre exemple– `connect1` et `mpCross1` et au moins un objet de la classe `crossConnection` (`cross1` et `cross2`) avec des valeurs d'attributs de relation référençant les points de terminaison. La fin de la communication doit se faire en supprimant ces objets. Le brassage a la particularité d'être flexible, c'est à dire qu'il est possible d'ajouter, d'enlever

ou de substituer des points recevant le trafic. Ceci se faisant au travers de création et destruction des objets de la classe `crossConnection`. Les objets gérés des classes `connectionTermination-`

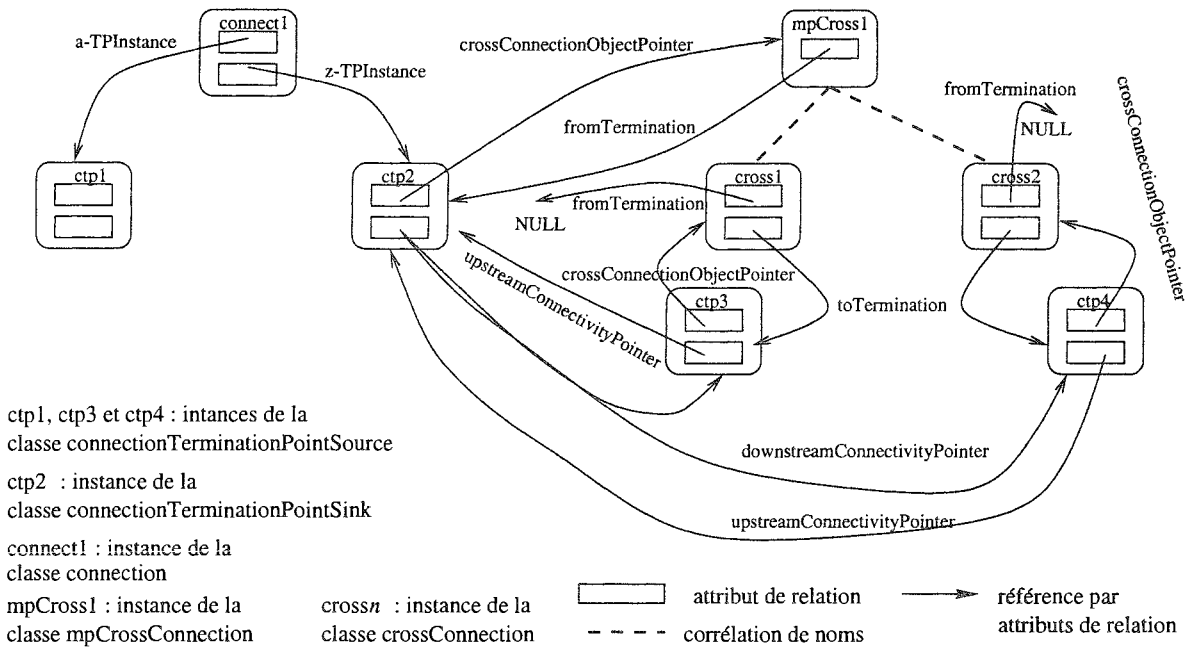


FIG. 4.10 – *Configuration de connexions*

`PointSink`, `connectionTerminationPointSource` et `connection` peuvent être remplacés par d'autres, dont les classes sont dérivées comme elles, des classes génériques `terminationPoint` et `pipe` (cf. fig. 4.11). Cependant, toutes les combinaisons ne sont pas admises et nous les détaillerons plus loin. Les objets issus des classes `mpCrossConnection` et `crossConnection` n'ont pas de classe dérivée et ce sont donc toujours des instances de ces classes qui sont utilisées dans les configurations de brassage multipoint. Il faut noter que la définition de la corrélation de noms qui lie ces deux dernières classes ne permet pas à un Gestionnaire OSI de créer avec le service `M-CREATE` des instances de la classe subordonnée. La création du brassage et l'ajoût de points de terminaison doivent être faits au travers d'actions d'une autre instance (de la classe `fabric`, non représentée sur la figure), qui peut être responsable de plusieurs brassages. En revanche, les suppressions d'instances de la classe `crossConnection` sont autorisées pour un Gestionnaire OSI. La description de ces configurations est exprimée informellement dans les différents formulaires de comportement des classes et de la corrélation de noms. L'utilisation du GRM va en formaliser une partie et en homogénéiser la gestion.

4.4 Classes de relations gérées

À partir de ces modèles de gestion, nous proposons de spécifier des classes de relations qui modélisent les connexions et les brassages. Les concepts communs à ces différents types de communication sont d'une part la présence de points de terminaison émettant et recevant le trafic, d'autre part la possibilité d'initialiser la communication, de la bloquer et de la débloquer plusieurs fois, enfin, de la supprimer. L'observation des différentes configurations fait apparaître qu'il existe toujours dans chacune d'elle une instance d'objet particulière contenant les attributs d'états normalisés [ISO92f] :

– l'attribut d'état opérationnel `operationalState` dont les valeurs sont `enabled`, lorsque la

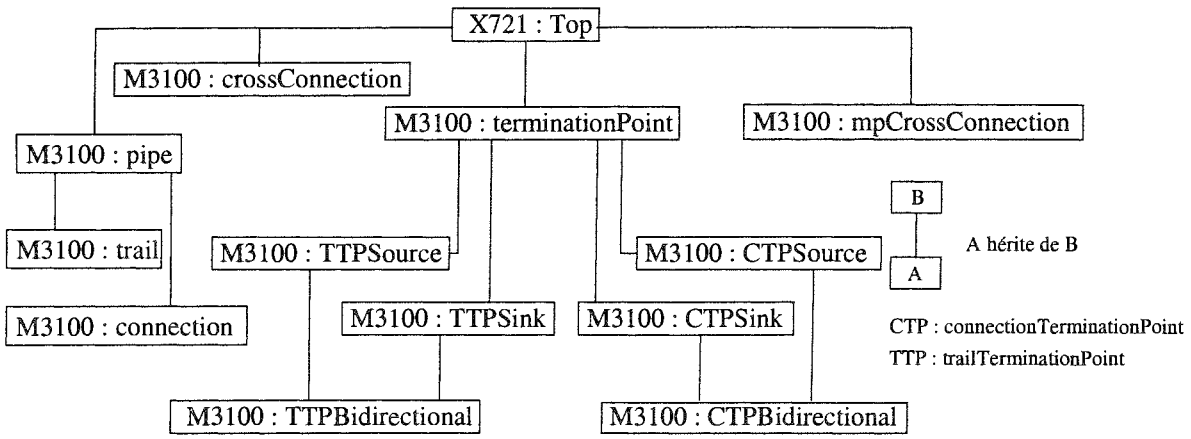


FIG. 4.11 – *Arbre d'héritage des classes*

ressource peut fonctionnner normalement et `disabled` lorsqu'elle a un problème perturbant son fonctionnement normal;

- l'attribut d'état administratif `administrativeState` dont les valeurs sont `unlocked` lorsque la ressource a l'autorisation d'un Gestionnaire OSI pour fonctionner et `locked` lorsqu'elle ne l'a pas.

La valeur de ces attributs influence le passage du trafic entre les points de terminaison. Ces instances sont issues des classes dérivées de la classe `pipe` ou de la classe `mpCrossConnection`.

4.4.1 La classe `traffic`

La classe `traffic` de la figure 4.12 modélise l'existance d'un trafic entre un point de terminaison dans le rôle `source` (lignes 10 à 13) et au moins un autre dans le rôle `destinations` (lignes 14 à 17). Comme les configurations peuvent être créées et détruites depuis un Gestionnaire OSI, nous avons doté la classe de relation des opérations `ESTABLISH` et `TERMINATE` (ligne 3). De plus des opérations permettent de récupérer les participants des différents rôles (ligne 4) et de bloquer ou débloquer le passage du trafic (lignes 6 et 7). Les états administratif et opérationnel de la relation sont donnés par les attributs qualifiants (ligne 8 et 9) dont les valeurs sont récupérables par des opérations de lecture (ligne 5). Cette classe est abstraite et les classes dérivées qui suivent sont instanciables pour les deux types de communication qui viennent d'être vues; les connexions entre éléments gérés et les brassages de connexions.

4.4.2 La classe `trafficPipe`

La classe de relation `trafficPipe` de la figure 4.13 modélise le passage du trafic entre éléments gérés, elle s'applique à la configuration entre les objets gérés `ctp1`, `ctp2` et `connect1` de la figure 4.10. Elle est dérivée de la classe `traffic` (ligne 3) et rajoute le rôle `connectivity` (lignes 6 à 9). De plus, la cardinalité du rôle `destination` est restreinte car le passage de trafic entre éléments gérés se fait d'un point de terminaison à un autre. Le sens du trafic est dépendant des classes d'objets, l'attribut qualifiant `directionality` (ligne 5) dont la valeur est soit `unidirectional`, soit `bidirectional` traduit cette caractéristique. L'opération de type `QUERY` (ligne 4) permet de récupérer cette valeur.

```
(1) traffic RELATIONSHIP CLASS
(2)   BEHAVIOUR trafficBehaviour;
(3)   SUPPORTS ESTABLISH, TERMINATE,
(4)            QUERY querySource, QUERY queryDestination
(5)            QUERY queryOpState, QUERY queryAdState,
(6)            USER-DEFINED lockTraffic,
(7)            USER-DEFINED unlockTraffic;
(8)       QUALIFIED BY "ITU-T X.721: 1992":administrativeState,
(9)                    "ITU-T X.721: 1992":operationalState;
(10)   ROLE source
(11)     COMPATIBLE-WITH "ITU-T M.3100 1995" terminationPoint
(12)     PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1
(13)    PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 0..1
(14)  ROLE destination
(15)    COMPATIBLE-WITH "ITU-T M.3100 1995" terminationPoint
(16)    PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..N
(17)    PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 0..1;
(18) REGISTERED AS {1};
```

FIG. 4.12 – *Classe de relation gérée de trafic entre points de terminaison*

```
(1)  trafficPipe RELATIONSHIP CLASS
(2)    BEHAVIOUR trafficPipeBehaviour;
(3)    DERIVED FROM traffic;
(4)    SUPPORTS QUERY queryDirectionality;
(5)    QUALIFIED BY "ITU-T M.3100: 1995":directionality;
(6)    ROLE connectivity
(7)      COMPATIBLE-WITH "ITU-T M.3100: 1995":pipe
(8)      PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1
(9)      PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 1..1
(10)   ROLE destination
(11)     PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1;
(12) REGISTERED AS {2};
```

FIG. 4.13 – *Classe dérivée de trafic sur un canal*

4.4.3 La classe `trafficMpCross`

La classe de relation `trafficMpCross` de la figure 4.14 modélise le brassage multipoint au sein d'un élément géré. Cette classe hérite de la classe `traffic` (ligne 3) et rajoute deux rôles; le rôle `mpCross` (ligne 6 à 9) et le rôle `leg` (lignes 10 à 13). Par ailleur, le rôle `destination` est étendu avec les opérations BIND et UNBIND (lignes 15 et 16) qui traduisent la flexibilité du brassage multipoint. Un attribut qualifiant (ligne 5), traduisant la disponibilité du brassage est rajouté. Sa valeur peut être soit un ensemble vide, soit contenir la valeur `degraded` lorsque le brassage multipoint est opérationnel sur une partie de ses points destinataires du traffic. L'opération `queryAvState` (ligne 4) permet de récuperer la valeur courante de cet attribut.

En ce qui concerne les formulaires de comportement référencés dans les classes de relations (ligne 2), nous les verrons dans d'autres chapitres. Dans le chapitre 6, des règles spécifiant les comportements liés à ces relations expriment des dépendances entre les valeurs des attributs d'états opérationnel ou administratif des points de terminaison et des objets dans le rôle `connectivity`. Dans le chapitre 7, nous proposeront une formalisation du comportement de la relation de brassage multipoint où la valeur de l'attribut opérationnel des objets dans le rôle `leg` influence l'état de disponibilité (attribut `availabilityStatus`) de l'objet dans le rôle `mpCross`.

```
(1)  trafficMpCross RELATIONSHIP CLASS
(2)    BEHAVIOUR trafficMpCrossBehaviour;
(3)    DERIVED FROM traffic;
(4)    SUPPORT QUERY queryAvState;
(5)    QUALIFIED BY  "ITU-T X.721: 1992":availabilityStatus;
(6)    ROLE mpCross
(7)      COMPATIBLE-WITH "ITU-T M.3100: 1995":mpCrossConnection
(8)      PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1
(9)      PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 1..1
(10)   ROLE leg
(11)     COMPATIBLE-WITH "ITU-T M.3100: 1995":crossConnection
(12)     PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..N
(13)     PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 0..1
(14)   ROLE destination
(15)     BIND-SUPPORT
(16)     UNBIND-SUPPORT;
(17) REGISTERED AS {3};
```

FIG. 4.14 – *Classe dérivée de brassage de trafic*

4.5 Les représentations de relations gérées

Les classes `trafficPipe` et `trafficMpCross` sont instanciables, cela revient à dire qu'il existe des représentations de ces classes de relations gérées. Nous avons spécifié une représentation possible pour chacune d'entre elles, correspondant aux configurations de la figure 4.10.

4.5.1 La représentation `connectionTrafficPipe`

La représentation `connectionTrafficPipe` de la figure 4.15 est une représentation de la classe `trafficPipe` (ligne 3). La partie du formulaire concernant les rôles (lignes 4 à 14) relie à chacun d'eux une classe d'objet géré. La configuration de la figure 4.10 entre les objets `ctp1`, `connect1` et `ctp2` est une instance de la relation `trafficPipe` (et donc de la relation `traffic`), où ces objets participent respectivement dans les rôles `source`, `connectivity` et `destination`. Ceci dans la mesure où les attributs de relation référencent ces objets (lignes 6 et 14). La traduction des opérations de relation en opérations système permet à un Gestionnaire OSI de gérer les relations gérées de la classe (lignes 15 à 32). On note des imprécisions dans la traduction d'opérations.

- L'opération `ESTABLISH` (ligne 16); la création d'un objet de la classe `connection` devrait se faire avec l'initialisation des attributs de relation `a-TPInstance` et `z-TPInstance` avec les valeurs des identificateurs des objets (leur *Distinguish Name*) devant participer aux rôles `source` et `destination`.
- Les opérations `lockTraffic` et `unlockTraffic` (lignes 28 à 31) ne précisent pas à quelle nouvelle valeur doit être positionné l'attribut `administrativeState`.

4.5.2 La représentation `connectionTrafficMpCross`

La dernière représentation de relation concerne la classe `trafficMpCross`. La figure 4.16 montre la représentation `connectionTrafficMpCross` de cette classe (ligne 3). La représentation des rôles (lignes 4 à 23) permet l'instanciation d'une relation gérée de la classe `trafficMpCross` à partir des objets de la figure 4.10. Les objets `ctp2`, `mpCross1`, `cross1` et `cross2`, `ctp3` et `ctp4` participent respectivement aux rôles `source`, `mpCross`, `leg` et `destination`. Tout comme la représentation précédente, la traduction des opérations de relation (lignes 24 à 43) manque de précision:

- les opérations `ESTABLISH` et `BIND` (lignes 25,26 et 42) on les valeurs initiales des attributs de relation `fromTermination` et `toTermination` qui ne sont pas spécifiées;

```
(1) connectionTrafficPipe RELATIONSHIP MAPPING
(2)   BEHAVIOUR connectionTrafficPipeBehaviour;
(3)   RELATIONSHIP CLASS trafficPipe;

(4)   ROLE source RELATED-CLASSES
(5)         "ITU-T M.3100: 1995":connectionTerminationPointSource
(6)     REPRESENTED-BY ATTRIBUTE "ITU-T M.3100: 1995":a-TPInstance;

(7)   ROLE connectivity RELATED-CLASSES
(8)         "ITU-T M.3100: 1995":connection
(9)     QUALIFY "ITU-T X.721: 1992":administrativeState
(10)            "ITU-T X.721: 1992":operationalState
(11)            "ITU-T M.3100: 1995":directionality;

(12)   ROLE destination RELATED-CLASSES
(13)        "ITU-T M.3100: 1995":connectionTerminationPointSink
(14)     REPRESENTED-BY ATTRIBUTE "ITU-T M.3100: 1995":z-TPInstance;

(15)  OPERATIONS MAPPING

(16)    ESTABLISH MAPS-TO-OPERATION CREATE OF connectivity,
(17)    TERMINATE MAPS-TO-OPERATION DELETE OF connectivity;
(18)    QUERY querySource MAPS-TO-OPERATION
(19)       GET "ITU-T M.3100: 1995":a-TPInstance OF connectivity;
(20)    QUERY queryDestination MAPS-TO-OPERATION
(21)       GET "ITU-T M.3100: 1995":z-TPInstance OF connectivity;
(22)    QUERY queryAdState MAPS-TO-OPERATION
(23)       GET "ITU-T X.721: 1992":administrativeState OF connectivity;
(24)    QUERY queryOpState MAPS-TO-OPERATION
(25)       GET "ITU-T X.721: 1992":operationalState OF connectivity;
(26)    QUERY queryDirectionality MAPS-TO-OPERATION
(27)       GET "ITU-T M.3100: 1995":directionality OF connectivity;
(28)    USER-DEFINED lockTraffic MAPS-TO-OPERATION
(29)       REPLACE "ITU-T X.721: 1992":administrativeState OF connectivity;
(30)    USER-DEFINED unlockTraffic MAPS-TO-OPERATION
(31)       REPLACE "ITU-T X.721: 1992":administrativeState OF connectivity;
(32) REGISTERED AS {2.1};
```

FIG. 4.15 – *Représentation d'une connexion*

- les opérations `lockTraffic` et `unlockTraffic` (lignes 38 à 41) modifient la valeur de l'attribut `administrativeState`, mais il n'est pas spécifié quelles valeurs sont prises suite à ces opérations;
- l'opération UNBIND (ligne 43) nécessite le *DN* du participant au rôle `destination` qui doit être enlevé de la relation gérée;
- comme le GRM ne permet pas de spécifier plus d'une représentation par rôle, la totalité des attributs de relation qui sont utilisés pour la relation (cf. fig. 4.10) n'apparaît pas dans le formulaire; il n'est par exemple pas fait mention des attributs `upstreamConnectivityPointer` ou `crossConnectionObjectPointer`.

Les tables 4.1 montrent les représentations possibles des classes de relations dans le modèle de gestion de la recommandation M.3100. La classe de relation gérée `traffic` étant une classe abstraite, les représentations des classes dérivées `trafficPipe` et `trafficMpCross` constituent l'ensemble des représentations de la classe `traffic`. Le trafic entre les points de terminaison dans une relation de la classe `trafficPipe` peut être bidirectionnel. Dans ce cas, les classes des participants des rôles `source` et `destination` doivent être bidirectionnelles. C'est le cas des représentations référencées 2.2 et 2.4 dans la table 4.1. Dans le cas des représentations de la classe `trafficMpCross`, seul un sens de transit du trafic est autorisé.

D'autres ensembles de représentations sont possibles si l'on utilise les classes de relations `traffic` et ses dérivées pour modéliser des relations sur un autre modèle de gestion.

classe trafficPipe			
Ref.	source	destination	connectivity
2.1	CTPSource	CTPSink	connection
2.2	CTPBidirectional	CTPBidirectional	connection
2.3	TTPSource	TTPSink	trail
2.4	TTPBidirectional	TTPBidirectional	trail

classe trafficMpCross		
Ref.	source	destination
3.1	CTPSink	CTPSource
3.2	TTPSource	TTPSink

CTP : connectionTerminationPoint
TTP : trailTerminationPoint

TAB. 4.1 – *Représentations possibles de la classe* `traffic`

4.6 Spécifier avec le GRM

Un même domaine de gestion (un réseau SDH, PDH, un réseau ATM, etc) peut être modélisé de plusieurs manières avec différentes classes d'objets gérés. Le même phénomène existe avec une modélisation utilisant des classes de relations et des représentations. La solution permettant d'homogénéiser les modélisations consiste à réutiliser des modèles génériques de gestion comme la recommandation M.3100 [M3195]. La même solution devrait être adoptée pour l'utilisation du GRM. Comme nous l'avons présenté sur l'exemple précédent, des relations génériques sont spécifiées sur la base de modèles objets. Les modèles qui sont dérivés peuvent réutiliser les relations ou encore les adapter à leurs configurations. Un exemple dérivé de celui présenté dans ce chapitre sera étudié au chapitre 6.

Une alternative à la spécification de relations fondées sur des configurations d'objets connues est celle de spécifier des relations indépendamment de toute classe d'objet (i.e. tous les rôles sont compatibles avec la classe `Top` [ITU92]). Ces relations génériques peuvent alors être dérivées pour s'adapter à des configurations d'objets. Le chapitre suivant détaille cet aspect du GRM.

4.7 Conclusions

Le modèle général de relation qui est défini dans la gestion OSI nous semble être celui qui offre le plus de possibilités pour modéliser toutes sortes de relations entre les objets gérés. Suivant cette idée, le prochain chapitre compare le GRM avec d'autres modèles de relation. Sa puissance d'expression doit toutefois s'accompagner d'une certaine prudence lors des spécifications afin de définir des classes de relations et des représentations qui soient le plus proche possible des relations du monde réel que l'on cherche à modéliser. L'utilisation d'un modèle de gestion existant nous semble une bonne ligne de conduite pour ces spécifications. Les interactions entre les ressources modélisées par les objets de ces modèles doivent être bien comprises pour en extraire des relations qui seront alors formalisées avec le GRM.

```
(1) connectionTrafficMpCross RELATIONSHIP MAPPING
(2)   BEHAVIOUR connectionTrafficMpCrossBehaviour;
(3)   RELATIONSHIP CLASS trafficMpCross;

(4)    ROLE source RELATED-CLASSES
(5)             "ITU-T M.3100: 1995":connectionTerminationPointSink
(6)      REPRESENTED-BY ATTRIBUTE
(7)               "ITU-T M.3100: 1995":fromTermination;

(8)    ROLE mpCross RELATED CLASSES
(9)             "ITU-T M.3100: 1995":mpCrossConnection
(10)     REPRESENTED-BY NAMING
(11)              "ITU-T M.3100: 1995":crossConnection-mpCrossConnection
(12)              USING SUPERIOR
(13)     QUALIFY "ITU-T X.721: 1992":administrativeState
(14)             "ITU-T X.721: 1992":operationalState
(15)             "ITU-T X.721: 1992":availabilityStatus;

(16)   ROLE leg RELATED CLASSES "ITU-T M.3100: 1995":crossConnection
(17)     REPRESENTED-BY NAMING
(18)              "ITU-T M.3100: 1995":crossConnection-mpCrossConnection
(19)              USING SUBORDINATE;

(20)   ROLE destination RELATED-CLASSES
(21)            "ITU-T M.3100: 1995":connectionTerminationPointSource
(22)     REPRESENTED-BY ATTRIBUTE
(23)              "ITU-T M.3100: 1995":toTermination;

(24)   OPERATIONS MAPPING

(25)     ESTABLISH MAPS-TO-OPERATION CREATE OF mpCross,
(26)                                 CREATE OF leg;
(27)     TERMINATE MAPS-TO-OPERATION DELETE OF mpCross;
(28)     QUERY querySource MAPS-TO-OPERATION
(29)       GET "ITU-T M.3100: 1995":fromTermination OF mpCross;
(30)     QUERY queryDestination MAPS-TO-OPERATION
(31)       GET "ITU-T M.3100: 1995":toTermination OF leg;
(32)     QUERY queryAdState MAPS-TO-OPERATION
(33)       GET "ITU-T X.721: 1992":administrativeState of mpCross;
(34)     QUERY queryOpState MAPS-TO-OPERATION
(35)       GET "ITU-T X.721: 1992":operationalState of mpCross;
(36)     QUERY queryAvState MAPS-TO-OPERATION
(37)       GET "ITU-T X.721: 1992":availabilityStatus of mpCross;
(38)     USER-DEFINED lockTraffic MAPS-TO-OPERATION
(39)       REPLACE "ITU-T X.721: 1992":administrativeState of mpCross;
(40)     USER-DEFINED unlockTraffic MAPS-TO-OPERATION
(41)       REPLACE "ITU-T X.721: 1992":administrativeState of mpCross;
(42)     BIND MAPS-TO-OPERATION CREATE OF leg;
(43)     UNBIND MAPS-TO-OPERATION DELETE OF leg;

(44) REGISTERED AS {3.1}
```

FIG. 4.16 – *Représentation d'un brassage de connexion*

Chapitre 5

Equivalences entre le GRM et d'autres modèles de relation

Après la description du GRM sous son aspect syntaxique dans le chapitre précédent, ce chapitre en décrit les fondements théoriques. Ceux-ci seront ensuite comparés avec d'autres modèles de relations.

5.1 Les motivations du GRM

Dans le domaine des SGBDs relationnels, hiérarchiques ou réseaux, le passage d'une spécification conceptuelle de type E/A vers les structures logiques de ces systèmes est établi dans [Che76]. Un traitement peut alors être exprimé sur la base du schéma conceptuel et profiter de l'abstraction qu'il apporte concernant la représentation des relations (comme les clés d'un SGBD relationnel). Le GRM vise le même but; à partir d'une spécification conceptuelle de classes d'objets et de relations gérés, fournir les moyens de traduire automatiquement cette spécification en une représentation logique. Celle-ci est constituée des classes d'objets et des corrélations de noms. Dans l'approche CORBA, par exemple, les relations d'un schéma conceptuel seraient directement traduites par des classes de relations et de rôles. La modélisation d'information avec le paradigme objet, comme dans [Mey88] [Boo91], conduit à un ensemble de classes d'objets. Les spécifications en GRM sont complémentaires de celles en GDMO; elles spécifient, avec les classes de relations gérées, des relations conceptuelles entre les classes d'objets gérés et précisent avec les représentations (formulaire `RELATIONSHIP MAPPING`) l'implantation logique de ces relations. Le GRM permet une meilleure compréhension entre les personnes chargées de l'analyse d'un système d'information et celles chargées de l'implanter [Kil92]. Les concepts de relations que formalisent le GRM ne laissent plus alors l'implantation être uniquement guidée par les possibilités spécifiques d'une technologie.

5.1.1 Les relations génériques

Les travaux dont est inspiré le GRM visent à isoler, dans une spécification d'information orientée-objet, les propriétés propres aux relations entre les classes d'objets. Cette recherche d'un dénominateur commun entre plusieurs classes d'objets a conduit à la notion de méta-type qui associe plusieurs classes, toutes ces classes étant vues de manière homogène [Kil91]. Des librairies de différentes variables de ce méta-type, appellées relations génériques ont été définies dans [Kil95] [KR94b] [Red96]. Le but de ces librairies est le même que pour une librairie de classes d'objets, c'est à dire fournir un ensemble de spécifications qui soient réutilisables dans des situations variées. Dans le GRM, ce méta-type correspond aux classes de relations gérées et leur visibilité des classes d'objets gérés est plus large que pour les

relations génériques grâce à la spécification de classes d'objets compatibles aux rôles de la classe de relation. Dans le chapitre précédent, nous avons illustré l'utilisation du GRM en modélisant des classes de relations gérées avec la connaissance du modèle de gestion de la M.3100. Cette démarche nous a amené à des classes de relations allant jusqu'à quatre rôles. Ces classes sont des relations génériques, mais elles sont moins génériques que celles des librairies car leurs rôles ne sont pas uniquement compatibles avec la classe `Top` (cf. §4.6 chap. 3).

Un formalisme graphique, défini dans [KR94b], représente les relations génériques. Une telle relation est schématisée (figure 5.1) par un triangle; marquant déjà qu'une relation n'est ni un objet ni une association d'un modèle E/A qui sont traditionnellement représentées par un losange ou simplement un trait[35] . Reliés à ce triangle, se trouvent les rôles. S'il s'agit d'une relation issue d'une librairie, les noms associés aux rôles, comme `composite` (fig. 5.1.a), ne sont pas des classes d'objets. En revanche, une réutilisation de ces relations dans un modèle de gestion consiste en premier lieu à choisir sur quelles classes du modèle vont s'appliquer les relations, les noms des classes peuvent alors remplacer les noms des rôles, comme `gtp` qui remplace `composite` dans la figure 5.1.a.

La figure 5.1 reprend une description graphique de quelques relations génériques issues de librairies et en propose des réutilisations dans le modèle des objets et des relations gérés, appliquées à l'architecture générique de réseaux vue au chapitre précédent. Les exemples de relations génériques choisis sont, comme la plupart des relations rencontrées, des dérivations de relations fondées sur une extension du comportement plus que de la structure. En revanche les dérivations vers des classes de relations gérées de l'architecture générique de réseaux sont fondées sur des spécialisations de structure des relations (classes de participants et cardinalité des rôles) :

- La relation de composition `C` (fig. 5.1.a) relie deux classes d'objets dans les rôles *composite* et *component*, sa sémantique est la suivante: *la présence d'une instance dans le rôle component implique la présence d'au moins une instance dans le rôle composite, la suppression de toutes les instances dans le rôle component entraîne celle des instances dans le rôle composite.* Cette classe de relation générique est réutilisée dans un exemple entre la classe d'objet géré `gtp` dans le rôle *composite* et la classe `connectionTerminationPointBidirectional` dans le rôle *component*. Une instance de la classe `gtp` modélise un groupe de points de terminaison. Ceux-ci peuvent alors être brassés au travers du `gtp` comme étant un seul point. Si le point de terminaison est bidirectionnel (i.e. de la classe spécifiée dans la figure), un point de terminaison peut faire partie d'un groupe de points pour chaque sens de trafic (nous avons spécifié cet exemple dans [FN97]);
- La relation de contenance `C'` (fig. 5.1.b) est dérivée de `C`. Le comportement qui y est défini est le suivant: *la présence d'au moins une instance dans le rôle component implique l'existance d'une unique instance dans le rôle composite, la suppression de cette instance entraîne celle des autres instances dans le rôle component.* La contrainte de l'unicité du participant au rôle *composite* renforce celle de la classe de relation `C`, celle de la suppression du participant au rôle *component* est rajoutée. La réutilisation qui en est faite s'applique à une partie de la configuration de brassage point à multipoint. La supression de l'instance dans le rôle *composite* entraîne celle de toutes les instances dans le rôle *component* (ces instances sont associées par corrélations de noms) et la supression du dernier objet géré dans le rôle *component*, signifiant la fin du brassage, entraîne par conséquent celle du *composite*;
- Une troisième relation `CO` (fig. 5.1.c) de composition ordonnée est dérivée de `C'`. Son comportement lui rajoute: *les instances dans le rôle component sont ordonnées.* La manière dont est déterminé l'ordre entre les instances du rôle *component* et la sémantique de cet ordre varient suivant les classes d'objets dans la relation. L'exemple de réutilisation modélise une connexion unidirectionnelle entre deux points de terminaison.

35. le triangle est également utilisé comme symbole de relation d'héritage, par exemple OMT

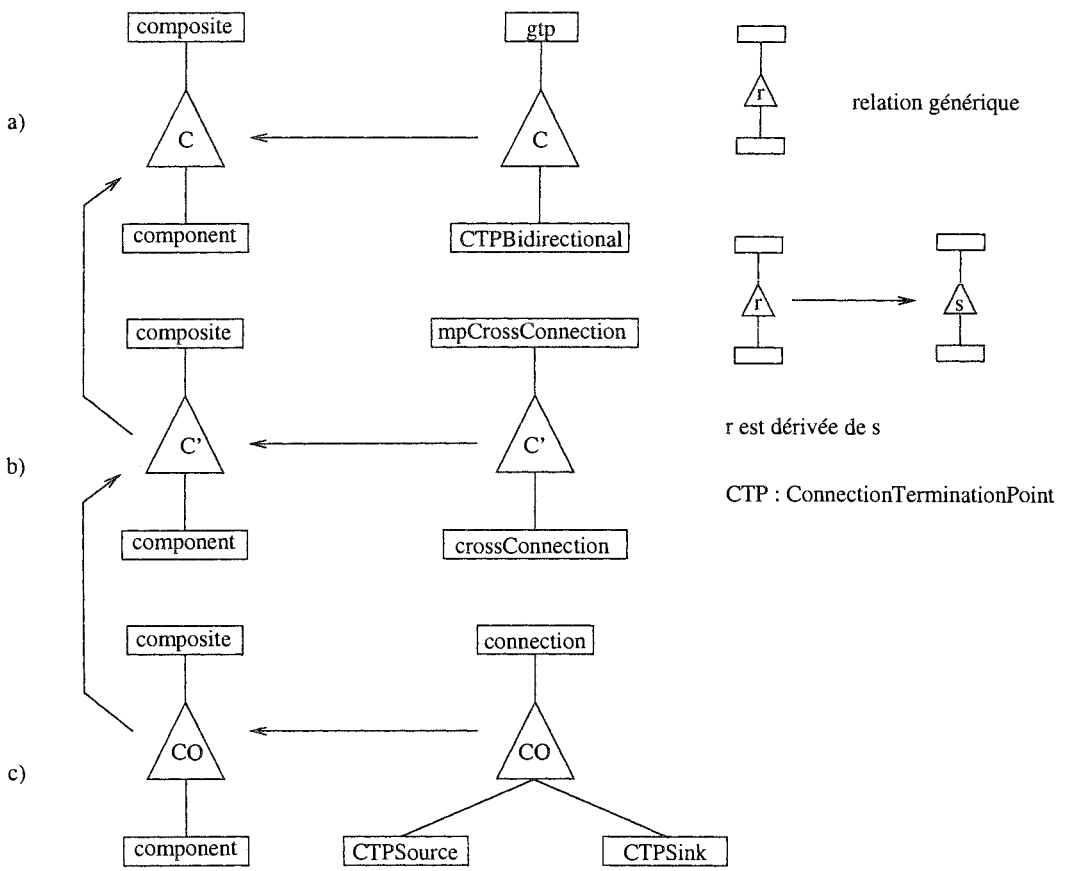


FIG. 5.1 – *Relations génériques et instanciations*

Dans l'exemple du chapitre 3, nous ne sommes pas partis de ces relations de base, mais avons adopté une démarche ascendante qui nous a permis de nous focaliser sur une fonctionnalité de plus haut niveau d'abstraction : la communication du trafic. Avec les exemples précédents, on voit que le GRM est également adapté à une modélisation plus proche des modèles E/A, où les relations binaires sont généralement préférées à des relations n-aires (comme celles du chapitre 3).

5.1.2 Le GRM et l'ODMA

Dans l'approche de l'ODMA (chap. 2 §2.5.2), il est recommandé d'utiliser des spécifications GDMO et GRM dans les points de vues suivants :

- le point de vue de l'information où sont spécifiés les objets d'information et leurs relations en utilisant un sous-ensemble de GDMO et du GRM, notamment les formulaires de représentation (`RELATIONSHIP MAPPING`) ne sont pas pris en compte;
- le point de vue de l'ingénierie prend en compte ces formulaires car ils sont spécifiques à l'infrastructure réelle des objets gérés.

Dans les objets du point de vue de l'information, il est prévu la spécification des relations potentielles qui peuvent lier ces objets. Toujours dans ce point de vue, le comportement des classes de relations gérées est spécifié avec la méthode Z; ce choix entraîne alors la nécessité de redéfinir le concept de relation avec Z (comme dans [Mar93] avec VDM).

5.1.3 Comportements des classes de relations gérées et d'objets gérés

Deux types de comportements cohabitent dans une modélisation de gestion; le comportement des objets et le comportement auquel participent les objets (i.e. celui des relations) [KR94a]. Un même objet géré peut être amené à changer d'environnement et donc de relations avec d'autres objets, par exemple dans la modélisation de réseaux de télécommunication sans fil [Din97], entraînant des comportements différents de ces objets. Le comportement d'une classe de relation gérée se traduit par une restriction du comportement des objets gérés participants. Cette restriction peut être nulle. Dans ce cas, la relation n'a pas d'influence sur les objets. Sinon, il faut vérifier que cette restriction ne soit pas en contradiction avec le comportement des objets (par ex. imposer une valeur d'attribut qui soit interdite dans la classe d'objet, mais permise dans le type de l'attribut).

Le comportement des représentations ne diffère guère de celui des classes de relations gérées. Le comportement d'une représentation ne doit pas contredire celui de la classe qu'il représente, diposant d'une information plus précise que cette dernière sur les classes des objets gérés participants, il peut exprimer des comportements supplémentaires.

Le comportement des relations gérées (classes et représentations) est une composante essentielle dans les modèles de gestion. La raison d'être des relations gérées tient plus aux conséquences qu'elles entraînent sur l'état des objets qu'aux contraintes de leur structure (rôles, opérations) et de leurs représentations (classes de participants, attributs de relation, ...). Nous montrerons dans la deuxième partie que la formalisation de ce comportement est un point essentiel pour la fiabilité des spécifications des systèmes de gestion et de leurs implantations jusqu'à leur exploitation par des applications de gestion.

5.2 Le GRM et les autres modèles de relations

5.2.1 Le modèle relationnel

Le modèle relationnel [Cod70] définit des relations comme des tables dont la première caractéristique (la première forme normale) est que le type des données soit unitaire. En cela, la représentation d'une

classe d'objet par une table du modèle relationnel échoue car les attributs de relation d'une instance d'objet peuvent désigner plusieurs instances. Les relations du GRM sont comparables avec les relations du modèle relationnel si l'on considère que le type des données est toujours un ensemble d'identificateurs d'objets gérés. Dans ce cas, chaque classe de relations gérées (ensemble des instances issues d'une même classe de relation gérée) est modélisable par une table comparable à celle du modèle relationnel. La figure 5.2 montre un exemple de deux instances de relation gérée et la table ci-dessous en reprend la composition en termes de tables de relations.

TableR		
RaRole	**RbRole**	**RcRole**
{a1}	{b1}	{c1, c2}

TableS	
ScRole	**SdRole**
{c1, c2}	{d1}

Les associations entre relations du modèle relationnel utilisent la notion de clé, qui n'est pas présente explicitement dans une relation gérée. Cette notion est indispensable pour permettre les associations entre les tables du modèle relationnel. Toutefois, des associations existent entre les tables modélisant les relations gérées, comme dans l'exemple où les objets gérés `c1` et `c2` participent aux relations gérées `r1` et `s1`. Une association existe entre les tables `TableR` et`TableS`. Elle est représentée par l'égalité entre les occurences des rôles `RcRole` et `ScRole`.

Il reste néanmoins le problème lié à l'héritage, où la structure d'une table modélisant une classe de relation gérée peut varier car cette table modélise également des instances issues de sous-classes qui peuvent rajouter des rôles et donc des colonnes dans les tables. Une proposition de mise en œuvre de certaines propriétés liées au modèle relationnel et adaptées au modèle des relations gérées sera vue au chapitre 8.

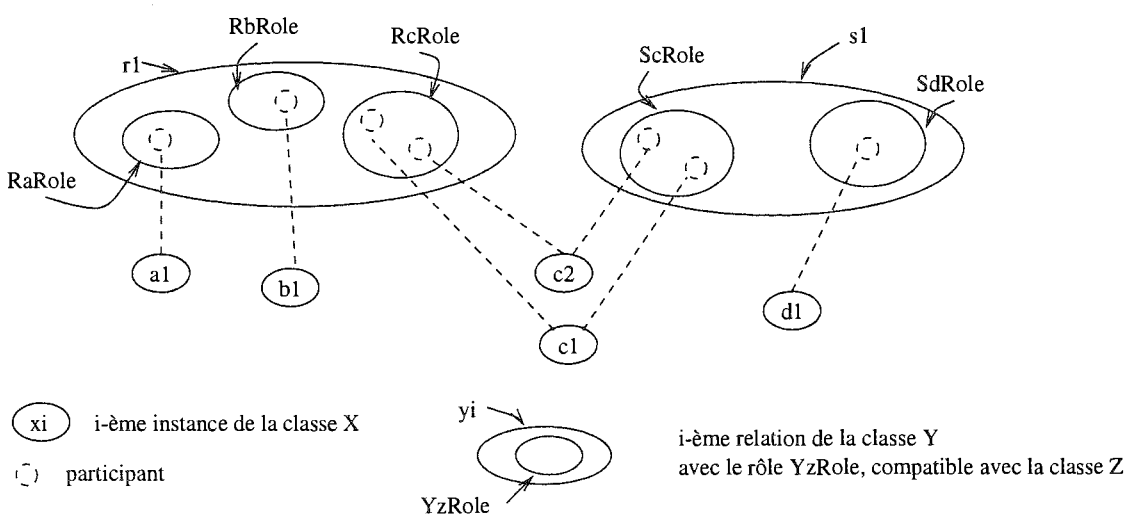


FIG. 5.2 – *Instances de relation gérées*

5.2.2 Les modèles sémantiques de données

Le modèle Entité Association [Che76] (E/A) a largement prouvé son efficacité; le niveau d'abstraction qu'il offre permet d'adapter une spécification E/A à différentes implantations logiques. A la différence du modèle relationnel, les associations entre les entités y sont explicitement séparées. Le modèle E/A fut le premier modèle sémantique fondé sur les associations au lieu des attributs [HK87], comme dans les modèles hierarchiques ou réseaux. Avec la prise en compte du paradigme objet, les modèles

Objet / Relation (O/R) ont accentué le besoin d'exprimer les relations entre classes d'objets avec des constructions au même niveau d'abstraction que les classes [Rum87]. Des opérations et des attributs peuvent y être définis au lieu de les disperser dans les classes. Ces dernières s'en trouve allégées et la sémantique est alors clairement cloisonnée entre celle des classes d'objets et celles des relations.

Comparaison avec le GRM

Les associations (i.e les relations[36] du modèle O/R) et les relations du GRM relient des classes d'objets dans des rôles, que nous appelerons des extrémités pour les associations. Si l'on se limite à cette comparaison, les relations du GRM, avec la notion de classes compatibles à un rôle, sont réutilisables sur plusieurs classes d'objets indépendamment de leurs liens d'héritages. Les associations sont, elles, définies pour chaque configuration de classes d'objets à associer. Avec le GRM, les représentations des classes de relations devront toutefois être spécifiées pour chaque configuration de classes d'objets à relier. Une spécification en GRM et GDMO est donc plus détaillée qu'avec GDMO et un schéma O/R. Les informations supplémentaires concernant les représentations des relations dans les classes sont formalisées, alors que les spécifications en GDMO décrivent informellement les représentations des relations.

La notion d'instance de relation (ou relation gérée) est une caractéristique importante dans le GRM. Une relation gérée accepte des requêtes d'opérations dont la portée est située dans l'ensemble de ses objets participants. Dans un modèle O/R, des opérations peuvent être définies pour une association, mais la portée en est l'ensemble des instances des classes aux extrémités. Cela correspondrait, pour le GRM, à invoquer une même opération sur toutes les relations gérées de la classe de relation.

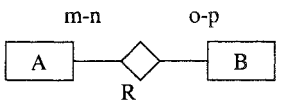
La sémantique des cardinalités aux extrémités des associations des modèles O/R n'est pas la même que celle des rôles des relations gérées. Dans les premiers, la cardinalité d'une extrémité donne le nombre d'instances de la classe de cette extrémité qui peuvent être associées avec une instance de chaque classe des autres extrémités de l'association. La cardinalité d'un rôle du GRM ne spécifie que le nombre d'instances pouvant participer au rôle. Concernant la cardinalité de relation d'un rôle, cette notion n'est pas directement transposable sur les modèle O/R car ils n'ont pas la notion d'instance de relation.

Malgrès ces différence, il est possible de modéliser une association par une classe de relation, moyennant quelques contraintes sur l'association. Ces contraintes imposent une configuration particulière des instances d'objets de telle sorte que l'on puisse en inférer des instances de la classe de relation. Elle peut être construite automatiquement à partir d'une association en suivant les règles suivantes (voir la figure 5.3):

- chaque extrémité de l'association représente un rôle de la classe de relation gérée, la classe d'objet devient la classe compatible au rôle;
- les cardinalités aux extrémités des associations sont transformées en cardinalités de rôle (`PERMITTED-ROLE-CARDINALITY-CONSTRAINT`);
- si la cardinalité minimale d'une extrémité est nulle, alors la cardinalité minimale de la relation (`PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT`) doit l'être aussi;
- la cardinalité maximale de la relation (`idem`) est toujours à 1.

Une conséquence du dernier point est que les valeurs de `m'` et `o'` sont soit 0, soit 1. Elles sont respectivement égales à `m` et `o` si ces valeurs sont dans l'ensemble {0, 1}. Lorsqu'une association a un 0 en cardinalité minimale de l'une de ses extrémité, cela signifie qu'une instance de la classe peut exister en étant reliée avec aucune instance des classes associées aux autres extrémités. Cette sémantique s'accorde avec celle de la cardinalité de relation dans un rôle; si elle est à 0, une instance d'objet géré peut exister sans participer à une instance de relation gérée de la classe de relation. Par ailleurs, les contraintes que nous avons imposées sur les instances d'objets reliés par l'association ne nous permettent pas de faire

36. le terme "association" est conservé pour éviter une ambiguité avec le terme "relation", lié au GRM




```
R RELATIONSHIP CLASS
  ...
  ROLE aRole
    COMPATIBLE-WITH A
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT m..n
    PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT m'..1
  ROLE bRole
    COMPATIBLE-WITH B
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT o..p
    PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT o'..1;
REGISTERED AS {...}
```

FIG. 5.3 – *Modèles O/R et GRM*

participer chaque objet à plus d'une relation gérée par instance d'objet. Ces contraintes, qui permettent d'isoler des instances de relation sont les suivantes:

- tous les groupes d'instances d'objets associés doivent pouvoir être sub-divisés, suivant leur classe. Le nombre d'instances d'objets doit être dans les bornes de la cardinalité de l'extrémité (correspondant au rôle respectif de chaque sous-groupe);
- toutes les instances d'un sous-groupe doivent être associées avec toutes les instances d'un autre sous-groupe (les deux sous-groupes appartenant au même groupe initial);

La figure 5.4 illustre ces contraintes par plusieurs instanciations possibles d'une même association, un exemple d'association ternaire est également donné. Les groupes entourés (en traits hachurés) représentent des instances de relation gérée de la classe de relation générée en suivant les règles précédentes. La figure 5.4.a montre trois instances possibles. En 5.4.b, on voit une relation saturée en objet, en revanche la configuration d'instances située en dessous n'est pas une relation gérée car il lui manque une association entre les instances `a4` et `b4`. La figure 5.4.c montre une configuration qui ne permet pas d'associer toutes les instances entre deux rôles, il y aurait sinon une violation des cardinalités aux extrémités de l'association. Enfin la figure 5.4.c montre une instanciation d'une association ternaire où l'on voit que les associations sont faites par couples de rôles; le rôle correspondant à la classe `A` avec celui de la classe `B` et celui de la classe `C` avec `B`. En fonction de la sémantique des associations, les classes de relations générées pourront être pourvues d'attributs et d'opérations de relation.

Les modèles sémantiques de données comme OMT[37] [Rum91] ou UML[38] [Mul97] sont nombreux à utiliser un symbolisme particulier pour exprimer une relation d'agrégation. La figure 5.5 illustre une telle relation. Une instance de la classe `A` doit être en relation avec un nombre indéterminé d'instances de la classe `B`, avec au plus une instance de `C` et exactement une instance de `D`. Une classe de relation du GRM, appellée `R` dans l'exemple, peut être générée dans laquelle le rôle correspondant à la classe agrégée (la classe `A` dans l'exemple) doit avoir une cardinalité de rôle limitée à 1; les autres rôles ont la même cardinalité que celle définie pour leur relation avec la classe agrégée. Les cardinalités des rôles de relation suivent la même règle que dans la figure 5.3.

37. Object-oriented Modelling and Technic
38. Unified Method Langage

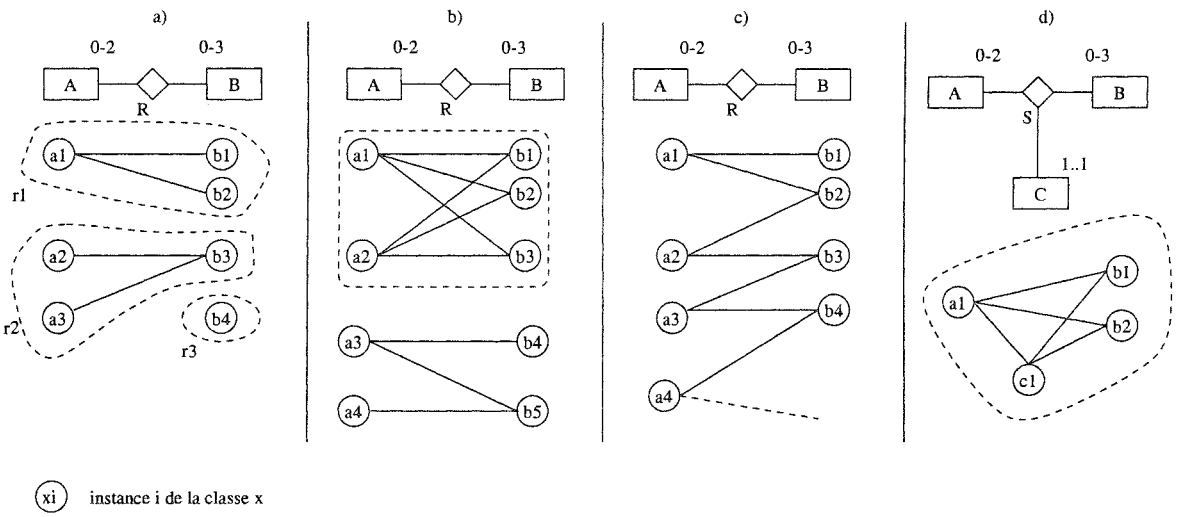
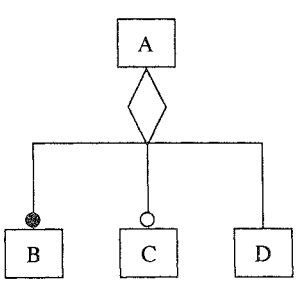


FIG. 5.4 – *Instanciations possibles d'une association*




```
R RELATIONSHIP CLASS
  ...
  ROLE aRole
    COMPATIBLE-WITH A
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1
  ROLE bRole
    COMPATBLE-WITH B
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT 0..N
  ROLE cROLE
    COMPATIBLE-WITH C
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT 0..1
  ROLE dROLE
    COMPATIBLE-WITH D
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1
  ...
```

FIG. 5.5 – *Relation d'agrégation*

5.2.3 La théorie des relation

Il serait incomplet de parler des relations entre des objets qui modélisent des ressources de réseaux sans citer la théorie des relations énoncée par S. Bapat [Bap94]. Cette théorie se fonde sur le modèle O/R (Objet / Relation) et s'intéresse plus particulièrement à la prise en compte, dans des sous-classes d'objets, des relations définies entre leurs super-classes. Lorsque des classes d'objets ont des relations, leurs sous-classes ont les mêmes relations et éventuellement en redéfinissent de nouvelles. Les relations "héritées" (ou initiales) peuvent être redéfinies pour s'adapter aux caractéristiques des sous-classes. Cette redéfinition peut s'opérer de différentes manières, en respectant certaines contraintes qui garantissent le respect des cardinalités des relations initiales. La figure 5.6 illustre par un exemple les règles énoncées dans la suite.

Une super-classe d'objet (dite *Subject*) et une classe d'objet (dite *Relatant*) sont en relation avec une cardinalité qui indique l'intervalle des nombres possibles d'instances de la classe *Relatant* (dite cardinalité du *Relatant*) en relation avec une instance de la classe *Subject*. Une sous-classe de la classe *Subject* ne doit pas contredire cette cardinalité en admettant des relations avec plus, ou moins, d'instances de la classe *Relatant*. La cardinalité du *Relatant* dans les sous-classes doit par conséquent être un sous-ensemble (non strict) de la cardinalité du *Relatant* de leur super-classe. Ce phénomène est appellé la restriction du *Relatant*. Ainsi (fig. 5.6), une instance de la classe `serveur` peut être en relation avec au plus 20 instances de la classes `client`; les classes `serveurDepartement` et `serveurEntreprise` qui en sont dérivées pourront être en relation avec respectivement, au plus 10 clients et au moins 10. Les premiers sont des serveurs moins puissants que les seconds et la restriction des cardinalités traduit ce fait.

La cardinalité du *Subject* est la cardinalité qui définit le nombre d'instance de la super-classe *Subject* qui peuvent être en relation avec une instance de la classe *Relatant*. Elle peut également être redéfinie dans la même situation que précédemment où la classe *Subject* est dérivée. Pour cela, il faut considerer qu'une instance de la classe *Relatant* peut être simultanément en relation avec des instances de la super-classe *Subject* et des sous-classes. Dans tous les cas (aucune instance de sous-classe, des instances de sous-classes et des instances de la super-classe, uniquement des instances de sous-classes) une instance de la classe *Relatant* ne devra pas être en relation avec plus d'instances que ne le permet la cardinalité du *Subject*. La borne supérieure de celle-ci doit donc être inchangée ou rabaissée dans les relations avec les sous-classes. La borne inférieure de la cardinalité du *Subject* suit une règle particulière dans le cas où plusieurs sous-classes sont définies; si on la laisse inchangée dans les relations avec les sous-classes et qu'elle est supérieure à 0 (disons égale à n et $n \geq 0$), une instance de la classe *Relatant* devra être en relation avec au moins n instances de chaque sous-classe (disons x sous-classes avec $xgeq0$), faisant passer la cardinalité minimale de n à $x \times n$. Dans l'exemple de la figure 5.6, la cardinalité minimale du *Subject* serait à 4. Pour remédier à ce problème, la théorie des relations impose une cardinalité minimale à 0 pour toutes les relations avec les sous-classes. Ce phénomène est appelé la 0-Regression du *Subject*. La borne inférieure de la cardinalité du *Subject* peut cependant être supérieure à 0 (comme la cardinalité de la classe `serveurDepartement`) dans une redéfinition de relation, mais dans ce cas, toutes les bornes supérieures des autres relations doivent être abaissées d'autant. Dans l'exemple, cela revient à dire que si une instance de `client` doit être en relation avec au moins une instance de `serveurDepartement`, alors il ne peut pas être en relation avec 5 `serveurEntreprise` car il pourrait se retrouver avec 6 serveurs et contredirait la cardinalité de la relation initiale. Dans une classe de relation gérée du GRM, les classes d'objets gérés compatibles aux rôles peuvent être dérivées et nécessiter une redéfinition de la sémantique (notamment les cardinalités des rôles) de la relation. Dans ce cas, une nouvelle classe de relation est dérivée et les cardinalités sont modifiées suivant le GRM. Dans le contexte de la théorie des relations, la situation étudiée plus haut se traduit donc, dans le contexte du GRM, par une dérivation de la classe de relation générée suivant le §5.2.2. Alors, si une classe de relation gérée est dérivée en dérivant

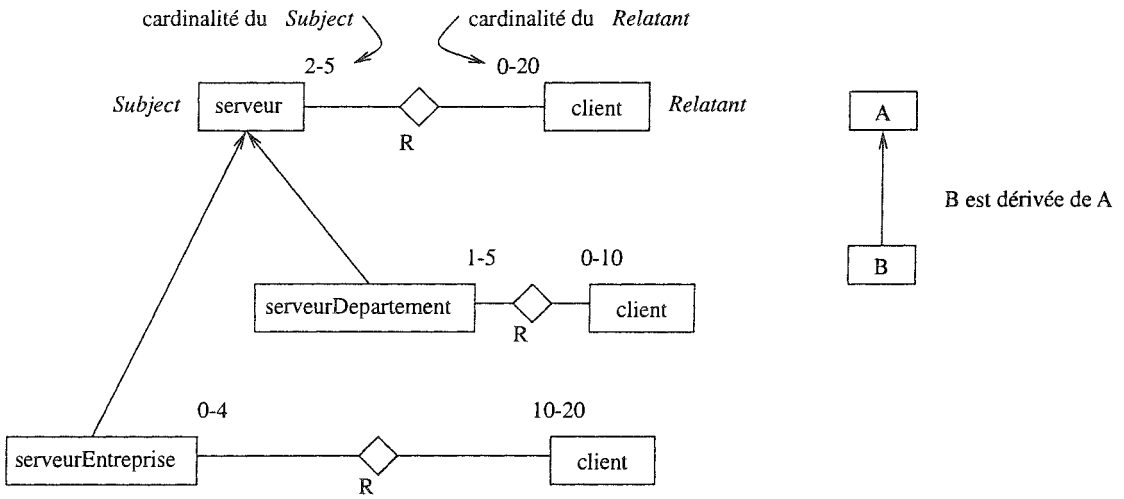


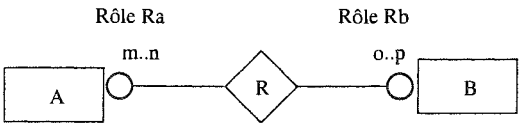
FIG. 5.6 – *Cardinalités de la théorie des relations*

une classe d'objet géré compatible à un rôle (rôle *Subject*), les cardinalités des autres rôles (rôles *Relatant*) peuvent être redéfinies. Les règles de l'héritage entre classes de relations sont alors concordantes avec la régression du *Relatant*: les nouvelles cardinalités sont des sous-ensembles des cardinalités initiales. La sémantique de la théorie des relations revient à dire, dans le modèle du GRM, qu'une relation gérée d'une sous-classe, étant également une relation de ses super classes, ne peut accepter plus, ou moins, de participant dans ses rôles. En revanche, la 0-Régression du *Subject* n'est pas acceptée par le GRM car elle enfreint l'héritage. La sématique de la théorie des relations se fonde pour la 0-Régression, sur le nombre d'instances du *Subject* en relation avec une instance du *Relatant*. Elle est traduite en GRM par la restriction des cardinalités de relation dans les rôles (et particulièrement ici, dans celle du rôle *Subject*) qui limite le nombre de relations par instance à 0 ou 1 (cf. §5.2.2).

5.2.4 Le GRM et le service de relation de CORBA

L'intégration des technologies CORBA et de la gestion OSI est un sujet traité dans plusieurs directions. Par exemple, les objets CORBA peuvent servir d'intermédiaires entre des applications de gestion et les objets gérés provenant d'approches telles l'OSI ou SNMP [FMB97] . Les mêmes objets CORBA peuvent être accessibles au travers d'objets Java [CK97]. Pour de telles solutions d'implantations, des passerelles entre différents modèles de l'information et de communication sont nécessaires [RS97] [SH97]. Enfin, ce sont les objets gérés qui peuvent être implantés en utilisant CORBA [CW97].

Dans les propositions de traduction entre GDMO et IDL, le GRM n'est pas pris en compte. Nous proposons ici de faire une correspondance entre une relation du service CORBA et une classe de relation du GRM comme l'indique la figure 5.7. Il s'agit de la traduction inverse à celles des modèles E/A de la figure 5.3 car la sémantique de la cardinalité d'un rôle de relation CORBA est le nombre des autres rôles avec lesquels une instance de ce rôle peut être reliée, alors que dans le modèle E/A, la cardinalité d'un rôle indique le nombre d'instances de ce rôle avec lesquelles une instance de chaque autre rôle peut être reliée. Ainsi, la cardinalité `m..n` du rôle `Ra` indique qu'au moins `m` objets de la classe `B` et au plus `n`, sont en relations avec un objet de la classe `A`.




```
R RELATIONSHIP CLASS
  ...
  ROLE Ra
    COMPATIBLE-WITH A
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT o..p
    PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT o'..1
  ROLE Rb
    COMPATIBLE-WITH B
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT m..n
    PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT m'..1;
REGISTERED AS {...}
```

FIG. 5.7 – *Relations CORBA et GRM*

5.3 Conclusions

Ce chapitre a montré quelles différences et quelles ressemblances existent entre le modèle de relation du GRM et d'autres modèles de relation. Le GRM est fondé sur le modèle des relations génériques avec lequel une correspondance entre spécifications est possibles quelque soit les relations modélisées. Il apparaît que les modèles inspirés du modèle E/A n'ont pas la même notion de relation que celle du GRM. Ce dernier étant orienté vers la description de groupes d'objets distincts ayant des interactions, alors que le GRM vise une description générale de tous les objets d'un système d'information.

Le modèle relationnel peut être utilisé pour organiser les objets de sorte à modéliser les relations du GRM. Par ailleur, nous avons établi une équivalence entre des spécifications de relations avec le modèle E/A et des classes du GRM.Les schémas E/A sont toutefois traduisibles en classes de relations gérées lorsque certaines contraintes sont imposées aux occurences de ces schémas. en revanche, la sémantique de l'aggrégation, souvent représentée par un symbole graphique particulier dans les modèles E/A est facilement modélisable avec le GRM. La théorie des relations qui complète les modèles E/A étendus à la notion de classes d'objets est conforme avec les contraintes du GRM. Enfin, une équivalence entre les relations définies par CORBA et celles du GRM a été donnée.

Deuxième partie

Intégration et exploitation du comportement dans la gestion des relations

Cette partie présente les contributions de cette thèse. Les relations entre les objets des MIBs entraînent des comportements globaux complexes qui nécessitent d'être validés. Le comportement propre des relations est formalisé dans cette partie, en accord avec le formalisme du comportement des objets gérés. Le dernier concept manquant aux relations de la gestion OSI : leur exploitation dans un environnement de gestion, a donné lieu à la conception d'une architecture pour la gestion des relations.

Chapitre 6

Formalisation du comportement global des MIBs

Les relations sont omniprésentes entre les objets gérés d'une MIB mais elles sont généralement exprimées de manière informelle. Ce chapitre présente des expérimentations visant à formaliser le comportement global d'une MIB, c'est à dire les comportements internes aux objets et les comportements liés aux relations entre ces objets. Les MIBs tendant à être de plus en plus complexes, la formalisation par une méthode nécessite une comparaison poussée entre les concepts de la gestion et ceux de la méthode choisie. L'existance d'outils logiciels pouvant traiter les spécifications formelles est également une nécessité.

6.1 Conception des MIBs

6.1.1 L'importance de l'approche OSI

L'approche de l'ISO pour la gestion des réseaux (qui suivent le modèle OSI) connaît un essort considérable depuis quelques années. Trois faits sont significatifs de ce phénomène. Celui qui porte le plus de conséquences est l'adoption de cette approche dans le modèle du RGT. Il en découle une grande activité dans des consortiums tels que le *Network Management Forum* (NMF) ou l'*ATM forum* autour de la modélisation des ressources de gestion fondée sur l'approche OSI. Le deuxième fait est que cette modélisation conduit à la définition de nombreux modèles de gestion. Enfin, le dernier fait est la croissance du développement de plate-formes d'administration offrant un support pour la gestion OSI. La plupart des fournisseurs de ces plate-formes offrent des interfaces OSI et de nombreux outils de développement d'agents sont disponibles sur le marché.

6.1.2 Besoins formels

Comme cela est souvent le cas dans les normes de l'ISO, une grande liberté est laissée pour leurs implantations. Dans le cas du comportement des objets et des relations gérés, une implantation sera conforme si elle se comporte de la manière indiquée dans les formulaires de comportement (`BEHAVIOUR`). La nature informelle de ces derniers fait que certaines interprétations peuvent différer d'une personne à l'autre, surtout lorsque le nombre de relations et d'objets est grand ou lorsque le volume de ces formulaires est de l'ordre de plusieurs dizaines de pages. Le besoin de décrire formellement le comportement des objets gérés a été exprimé à plusieurs reprises [Kil92] [Cle93] [CF93] et quelques études de faisabilité ont été menées avec différentes méthodes formelles [Mar93] [DLT95] mais peu

d'entre elles ont porté sur des cas réels de catalogues d'objets gérés. Parmi celles ci, il faut citer le développement d'un environnement de simulation des modèles de gestion OSI [Sid95] [EMS97] dont nous détaillons les principes plus loin.

La conception des modèles de gestion est une activité qui nécessite la mise en œuvre d'une ingénierie similaire à celle des protocoles telle que présentée dans [Raf95]. On retrouve notamment lors du développement de MIBs les besoins de conception, spécification formelle, validation, test de conformité et d'interopérabilité. Il existe cependant quelques différences par rapport aux approches utilisées pour les protocoles, comme la base orientée objet et de nouvelles données pour les tests de conformité et d'interopérabilité [KS96]. Dans notre étude, nous nous limitons aux spécifications formelles, à la simulation et la validation de base d'information de gestion.

6.1.3 Objectifs

Ce chapitre présente les résultats d'une étude et d'une mise en œuvre de méthodes formelles disposant d'une sématique opérationnelle et d'outils d'animation et/ou de verification pour la modélisation d'une MIB.

Nous nous interessons particulièrement à ce type de modélisation car une MIB n'est jamais un ensemble de classes d'objets gérés dont les instances sont indépendantes les unes par rapports aux autres. Au contraire, une MIB est la modélisation d'une ensemble de ressources qui sont dépendantes et fonctionnent dans des buts communs.

Cette étude a trois objectifs. Le premier est d'évaluer l'adéquation de plusieurs méthodes formelles à la modélisation des systèmes de gestion. Le second vise à isoler, pour chaque méthode formelle utilisée, ce qui concerne la modélisation des objets gérés pris individuellement de la modélisation des relations inter-objets et des conséquences qu'ont ces relations sur le comportement des objets. Le dernier objectif est de cerner l'apport d'outils logiciels disponibles pour les différentes méthodes dans des activités telles que l'animation et la validation des modèles de gestion.

Les trois méthodes formelles que nous avons retenues sont :

1. les systèmes de transitions étiquetées utilisés dans le logiciel MEC [ABC94],
2. le langage LOBSTERS [39] [Fes95] conçu spécifiquement pour la description du comportement des objets gérés dans GDMO,
3. le langage normalisé LDS [40] [CZ92] largement utilisé dans les télécommunications pour la spécification des protocoles de communication.

Les deux premières méthodes sont contenues dans ce chapitre, l'étude de la troisième est donnée dans [AFN+97] [TAF97].

6.2 Le simulateur TIMS

Cette section présente une étude récente dans le domaine de la formalisation du comportement dans les MIBs. La réalisation produite de cette étude est un simulateur de comportement, nommé TIMS [41] [Sid95] [EMS97], qui permet en outre de faire du prototypage rapide de systèmes de gestion RGT, de la génération de tests et de la validation de propriétés.

Les MIBs sont modélisées avec les langages normalisés pour la gestion OSI; GDMO, GRM et ASN.1. Le comportement des objets et des relations est formalisé avec le langage *Scheme* [CR91]. Les spécifications de comportement des relations sont organisées en événements, condition et action (règles ECA);

39. Language for Object Behaviour Specification based on Templates and Extended Rule Systems
40. Langage de Description et de Spécification
41. TMN-based Information Model Simulator

un événement détecté par une relation, entraîne l'évaluation de la condition qui lui est associée. Si l'évaluation est positive, l'action associée doit alors se dérouler. Les actions sont elles mêmes associées à des pré-conditions et des post-conditions devant être évaluées positivement avant et après l'exécution de l'action.

Les relations entre les objets sont le contexte le mieux adapté pour exprimer la propagation d'événements entre ces objets. L'événement d'une règle ECA dans une relation peut être détecté par l'occurence d'un événement dans des objets liés à la relation. L'action de cette même règle ECA peut alors entraîner le déclanchement d'événements dans des objets liés à la relation. La propagation peut alors se poursuivre, notamment si ces derniers objets sont également liés à une autre relation.

Les comportements sont analysés pour former un arbre de comportements où les noeuds sont des états du système avec un comportement à exécuter. Les noeuds immédiatement en dessous d'un noeud sont les différents états provoqués par l'exécution du comportement du noeud supérieur. L'arbre de comportement est obtenu à partir d'un noeud initial qui formera la racine de l'arbre. L'algorithme de recherche des noeuds s'arrête lorsqu'il n'y a plus d'état produit. Toutefois, la taille de l'arbre croit de manière exponentielle et il n'est pas possible de générer exhaustivement tous les états possibles du système. Plusieurs modes de simulation sont utilisables; un mode pas à pas où l'utilisateur choisit le comportement à exécuter parmi ceux qui sont en dessous du noeud qu'il vient de faire s'exécuter, un mode en profondeur et en largeur, suivant l'ordre de parcours de l'arbre et un mode aléatoire.

La génération de tests de comportements à partir de l'arbre de comportement et de l'outil TGV[42] [FJJV96] est détaillée dans [Maz97]. Les possibilités en matière de validation de propriétés à partir des arbres de comportement sont données dans [Sid97].

Notre étude diffère des travaux cités dans le sens où nous nous sommes intéressés à l'utilisation de méthodes formelles qui intègrent une description des données et de leur comportement dans une même approche, les FDTs ayant des outils déjà existant. Les différents résultats de ces outils, suivant les FDTs, apportent chacun une explication du modèle formalisé, focalisée sur les propriétés que la FDT manie le mieux.

6.3 Un cas d'étude

Les différentes FDTs qui sont vues dans la suite de ce chapitre ont été utilisées pour formaliser un modèle de gestion produit par le *Network Management Forum* dans le cadre de ses ensembles de solutions [NMF92]. Le modèle contient une description de la gestion de la configuration de l'interconnexion de commutateurs de télécommunication. La figure 6.1.a positionne les commutateurs dans l'architecture du TMN. La figure 6.1.b schématise les classes d'objets gérés qui sont définies dans le document `SIM-CM`[43] [NMF94] ou empruntées à d'autres normes pour modéliser les ressources (les classes `X721` [ITU92], `M3100` [M3195] et `ETSI GOM` [DI/96]). La classe `managedElement` modélise un commutateur. Il peut être composé d'un certain nombre d'équipements (des baies, des cartes, etc) modélisés par la classe `equipment`. Les connexions qui entrent en jeux au niveau réseau sont modélisées par les classes `interExchangeCircuit` et `circuitSubGroup`. Les extrémités physiques et logiques des connexions sont respectivement modélisées par les classes `trailTerminationPointBidirectional` et `circuitXtp`, `circuitSubGroupTerminationPoint`. Les relations d'héritages qui lient ces classes avec celles définies dans les autres documents sont données dans la figure 6.1.c.

La gestion de l'interconnexion des commutateurs est une fonction de gestion qui relève de l'aire fonctionnelle de la configuration [ISO92a]. On s'interesse donc particulièrement aux états des objets gérés (c.a.d essentiellement les valeurs de leurs attributs), en décrivant ceux qui sont acceptables et ceux

42. Test Generation by Verification Technology
43. Switch Interconnection Management - Configuration Management

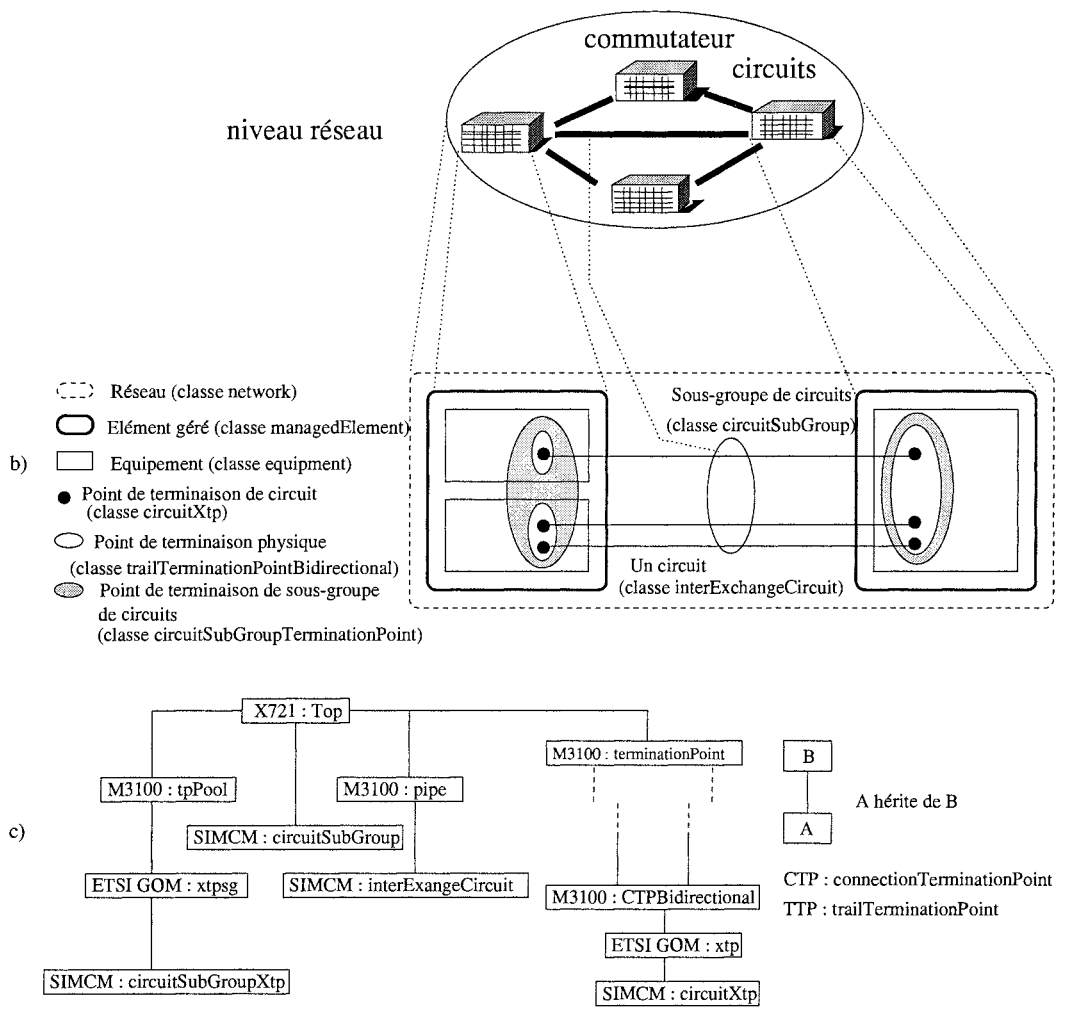


FIG. 6.1 – *Abstraction du SIM-CM*

qui ne le sont pas. Les transitions possibles entre les états sont également des caractéristiques importantes du modèle. Le moyen choisi dans le document du NMF est d'exprimer de tels états et transitions avec des règles, informelles, mais fondées sur l'existance de différentes relations de dépendances. Elles sont reprises dans les paragraphes suivants.

Dépendances de connectivité

Les dépendances de connectivité existent entre les objets gérés modélisant des connexions et les points de terminaison de ces connexions dans les commutateurs. Ces dépendances ont été spécifiées en partie par la classe de relation `trafficPipe` (cf. §4.4 chap. 3). Nous en verrons une classe dérivée qui traduit exactement ces dépendances. Dans les définitions des classes de ces objets, on retrouve les deux attributs d'état (cf. §4.4). Dans un même objet possédant ces attributs, tous les couples de valeurs sont admissibles. Seul l'attribut d'état administratif peut être modifié par un Gestionnaire OSI. L'attribut d'état opérationnel varie en fonction de la ressource modélisée. Les dépendances sont exprimées entre un objet modélisant une connexion (ou un sous groupe de connexions) et les deux objets qui modélisent les deux points de terminaison (ou sous-groupes de points), dans des commutateurs distincts. Les règles stipulent que :

R1 lorsque l'attribut d'état administratif d'un point de terminaison passe à la valeur `locked`, les attributs d'état opérationnel de la connexion et du point de terminaison situé à l'autre extrémité de la connexion doivent passer à la valeur `disabled`;

R2 lorsque l'attribut d'état administratif d'une connexion passe à la valeur `locked`, les attributs d'état administratif et d'état opérationnel des deux points de terminaison doivent passer aux valeurs `locked` et `disabled`;

R3 lorsque l'attribut d'état administratif d'une connexion passe à la valeur `unlocked`, les points de terminaison doivent faire passer leurs attributs d'états opérationnel et administratif aux valeurs `enabled` et `unlocked`. La connexion doit également faire passer (ou conserver) son état opérationnel à la valeur `enabled`.

Dépendances de contenance

Les dépendances de contenance relient certaines classes par couples. Pour chacun de ces couples, les instances d'une des classes, dite *contenant* contiennent des instances de l'autre classe dite *contenu*. La règle (**R4**) décrivant cette dépendance stipule qu'une modification de l'attribut d'état opérationnel d'un objet *contenant* doit se repercuter par une modification identique sur chaque attribut d'état opérationnel (s'il était à la même valeur) dans les objets *contenu*. Chaque objet *contenant* ou *contenu* possède donc cet attribut.

Dépendance de groupe

La dépendance de groupe relie également certaines classes par couples. Pour chacun des couples, les instances d'une classe *fédérateur* représentent des instances de l'autre classe *fédéré*. Les cardinalités du couple *fédérateur* / *fédéré* sont les même que pour le couple *contenant* / *contenu*. Les règles de comportement associées sont:

R5 la mise à la valeur `locked` d'un objet *fédérateur* entraîne une mise à la valeur `disabled` de l'objet *fédéré* correspondant.

R6 la mise à la valeur `unlocked` d'un objet *fédérateur* entraîne la mise à la valeur `enabled` de l'objet *fédéré* correpondant.

Règle supplémentaire

Une dernière règle (**R7**) est spécifiée de façon transversale par rapport aux dépendances qui viennent d'être vues. Chaque objet possède également un attribut de disponibilité (`availabilityStatus`) qui comptabilise le nombre de dépendances d'un objet. Cette règle énonce que le changement de la valeur de l'attribut d'état opérationnel à la valeur `disabled`, en conséquence de l'application d'une des règles précédentes, doit entraîner l'ajoût d'une valeur `dependency` à la valeur ensembliste[44] de l'attribut de disponibilité. Un passage à la valeur `enabled` enlève une valeur `dependency`.

Vue globale

Le document du NMF donne un schéma de type E/A dont s'inspire la figure 6.2 pour schématiser les relations de dépendances entre les classes. Sur ce schéma, les classes GDMO sont représentées par les rectangles et les différentes relations par des traits ou des flèches différents. Les relations de dépendances de contenance donnent à juste titre une forme arborescente au schéma car ces relations sont toujours représentées par des corrélation de noms (`NAME BINDING`). Les autres relations sont représentées par des attributs de relation [ISO92b].

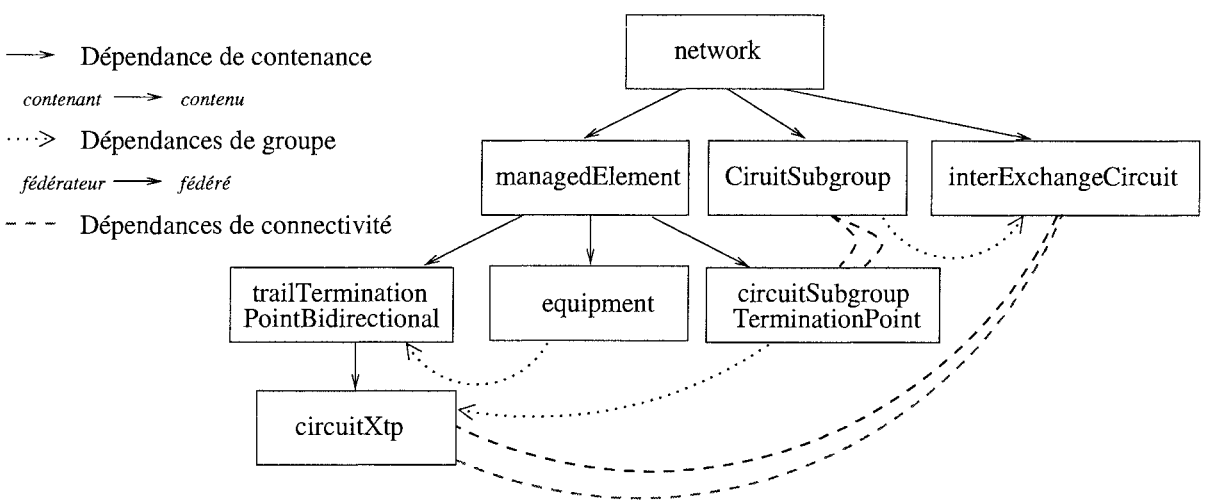


FIG. 6.2 – *Dépendances*

6.3.1 Critiques du document

La première critique porte sur l'aspect informel des règles, plusieurs interprétations sont possibles pour des situations moyennement complexes. Les règles sont assez simples, mais elles interfèrent quasiment toutes entre elles. La deuxième critique concerne l'utilisation d'exemples pour illustrer le déroulement des règles. Ces exemples prennent en compte des configurations réduites, avec un sous ensemble des objets, qui ne font pas apparaître toutes les interférences possibles entre les règles. La dernière critique du document est liée à la spécification semi-formelle qui est faite pour décrire des scénarios de gestion (constitution de connexions et de sous groupes de connexions). Ces spécifications ne prennent pas en compte l'état des objets manipulés ou ne donnent pas de précision sur des valeurs d'initialisation.

44. du type ASN.1 `SET OF` à considérer comme un multi-ensemble

6.4 Automates synchronisés

Si l'on modélise l'état d'un objet géré par la valeur de ses attributs à un instant donné, les règles de la section précédente font apparaître que l'état d'un objet géré est dépendant de l'état d'autres objets gérés, au même instant. Le nombre de ces attributs est faible (on en compte trois) et ils sont présents dans les différentes classes d'objets gérés. La modélisation de ces seuls attributs est donc suffisante pour modéliser le système. La plupart de ces attributs ont un domaine de valeur peu complexe (deux valeurs pour les attributs d'état administratif et opérationnel) et les règles sont exprimées en fonction des transitions de ces valeurs. Ces constatations nous ont conduits à modéliser les classes d'objets gérés par des Systèmes de Transitions Etiquetés (STE).

Les règles donnent des relations de cause à effet entre des transitions de différents attributs. La notion de temps de propagation n'y est pas prise en compte. Cette contrainte existe mais une analyse de la cohérence des règles peut déjà être menée sans la prise en compte du temps, qui pourra par la suite être exprimée suivant les caractéritiques de l'implantation (répartition des objets gérés, bande passante de la gestion, etc). Les relations entre transitions peuvent dans cette hypothèse être prises en tant que synchronisation entre transitions, c'est à dire que la cause se produit dans le même temps que les effets.

La modélisation des attributs en STEs et la modélisation des règles par synchronisation des transitions permet alors de construire un STE représentant l'ensemble du système. Ce *produit synchronisé* [Arn92] peut alors être analysé pour en extraire des propriétés. L'outil MEC [ABC94] a été conçu pour formaliser des STEs et en synchroniser les transitions. Il fournit de plus un langage de spécification permettant d'exprimer des propriétés à valider. L'utilisation de MEC dans des spécifications liées au domaine des télécommunications [BA93] [MAG97] en font un outil reconnu pour son application à des études de spécification de ressources de réseaux. Cette section reprend les points essentiels de notre modélisation du document du NMF. La version complète a donné lieu à un rapport de recherche [NFS96] et [NFSA96] en donne une version plus concise.

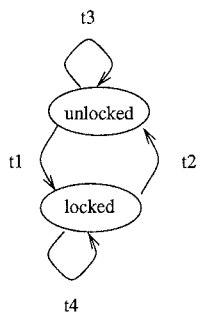
6.4.1 La spécification des STEs

Un attribut peut être modélisé par un STE, c'est à dire : un quintuplé $\langle S, T, \alpha, \beta, \lambda \rangle$ où $\langle S, T, \alpha, \beta \rangle$ est un système de transitions d'ensemble d'états S, d'ensemble de transitions T et de fonctions α, β associant à chaque transition $t \in T$, un état d'origine $\alpha(t)$ et un état de destination $\beta(t)$. La fonction λ associe à toute transition de T une étiquette $\lambda(t)$ indiquant l'action ou l'événement provoquant la transition [Arn92]. Les étiquettes sont construites sur un alphabet donné A. Pour des attributs dont le domaine de valeur est fini, il est possible de modéliser chaque valeur par un état et de spécifier les transitions nécessaires. Ceci a été très facilement réalisé pour l'attribut d'état administratif ne contenant que deux états `unlocked` et `locked`. La figure 6.3 montre la définition des fonctions, un schéma du STE et son expression dans MEC (`transition_system`).

En revanche, le domaine infini de l'état de disponibilité nous a contraint à ne modéliser qu'un sous ensemble des valeurs et des transitions possibles, la figure 6.4 schématise une partie du STE, la spécification complète nous a conduit à aller jusqu'à dix états. Pour définir ce chiffre, nous avons modélisé dans un premier essai le STE pour aller jusqu'à 13 états, l'examen du produit syncronisé nous a alors montré que seul les dix premiers états étaient accessibles (l'incrémenation maximale étant de 3 valeurs `dependency`).

Il s'est avéré inutile de modéliser séparément l'attribut d'état opérationnel car sa valeur est totalement couplée avec celle de l'état de disponibilité; l'état `empty` correspondant à la valeur `enabled`, tout autre état (`1_dep`, `2_deps`, ...) signifiant une valeur à `disabled`.

$S = \{unlocked, locked\}$
$T = \{t1, t2, t3, t4\}$
$A = \{lock, unlock, e\}$
$\alpha(t1) = unlocked \quad \beta(t1) = locked$
$\alpha(t2) = locked \quad \beta(t2) = unlocked$
$\alpha(t3) = unlocked \quad \beta(t3) = unlocked$
$\alpha(t4) = locked \quad \beta(t4) = locked$
$\lambda(t1) = lock$
$\lambda(t2) = unlock$
$\lambda(t3) = \lambda(t4) = e$




```
transition_system
   AdministrativeState;

unlocked |-- e -->unlocked,
         lock -->locked;
locked   |-- e -->locked,
         unlock -->unlocked;
<initial = {unlocked}>.
```

FIG. 6.3 – *Le STE de l'attribut d'état administratif*

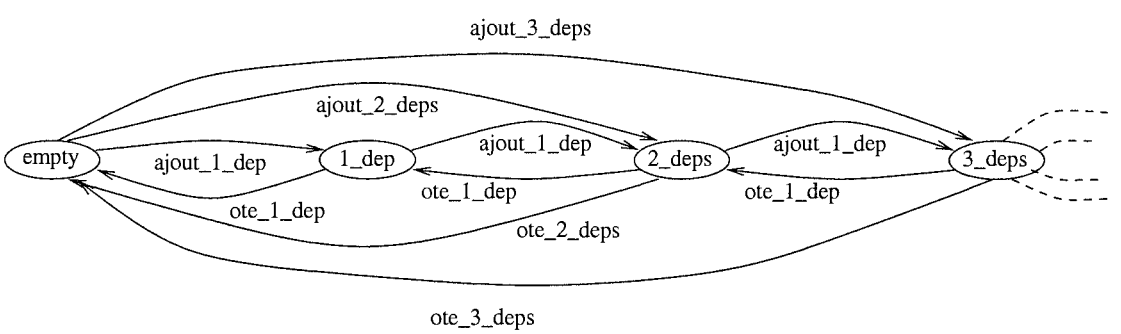


FIG. 6.4 – *Une partie du système de transitions étiquetées de l'état de disponibilité*

6.4.2 La synchronisation

Avec les deux STEs précédents, nous pouvons modéliser toutes les classes d'objet gérés du système. La spécification des synchronisations ne s'est pas faite sur l'ensemble de ces classes dans un seul modèle; nous avons scindé la configuration de la MIB en deux parties. Les raisons de cette division sont d'une part la complexité et d'autre part la limitation de l'outil MEC vue l'explosion combinatoire du produit synchronisé. La configuration de la figure 6.5 prend en compte deux types de dépendances et six instances d'objets gérés. Nous avons donc modélisé les six objets avec, pour chacun d'eux, un STE pour l'attribut d'état administratif et un STE pour l'attribut d'état de disponibilité. Aussi, dans la figure 6.6, on définit le

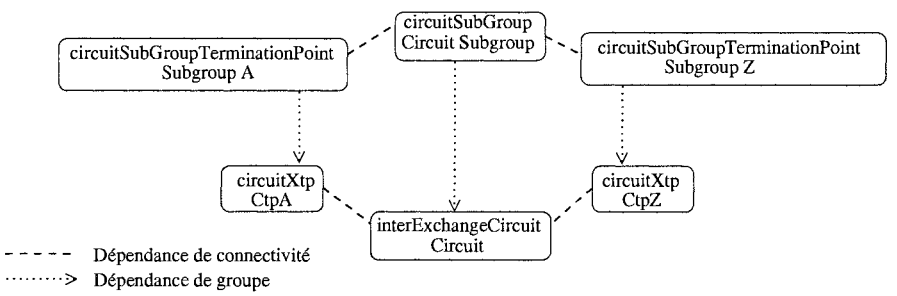


FIG. 6.5 – *Configuration modélisée*

système de synchronisation `circuits` par la liste des STEs utilisés : `administrativeState` pour l'état administratif et `operationalStateDependency` pour l'état de disponibilité. Les instances modélisées sont indiquées par couple de STEs et correspondent à celles de la figure 6.5.

```
synchronization_system circuits
 <width=12;
    list=(AdministrativeState, OperationalStateDependency,    /* circuitSubGroupTerminationPoint SubGroup A */
          AdministrativeState, OperationalStateDependency,    /* circuitXtp CtpA */
          AdministrativeState, OperationalStateDependency,    /* circuitSubGroup CircuitSubGroup */
          AdministrativeState, OperationalStateDependency,    /* interExchangeCircuit Circuit */
          AdministrativeState, OperationalStateDependency,    /* circuitXtp  CtpZ */
          AdministrativeState, OperationalStateDependency)>;  /* circuitSubGroupTerminationPoint SubGroup Z */
```

FIG. 6.6 – *Le système de synchronisation*

Les quatres vecteurs de transitions donnés dans la figure 6.7 modélisent l'action de blocage administratif d'un circuit. Les transitions d'un vecteur sont séparées par un point. Chaque transition est appliquée au STE du même rang dans le système de synchronisation. Toutes les situations possibles concernant les points de terminaison CtpA et CtpZ sont à prendrer en compte car elles influencent les transitions à synchroniser. La formalisation de chaque vecteur de synchronisation est le résultat d'une démarche systématique :

1. pour chaque objet de la configuration, le gestionnaire peut appliquer soit une action visant à bloquer l'état administratif, soit à le débloquer (transitions `lock, unlock`);
2. pour chacune de ces actions, il faut considérer la valeur de l'état administratif des autres objets présents dans la configuration, lorsque celle-ci a une influence, si elle est déjà à `locked` par exemple, il n'y a pas de transition vers l'état `locked` et donc pas de propagation (cf **R1** à **R3** et

R5, R6);

3. lors de l'application d'une action du gestionnaire, il faut appliquer les règles de comportement existantes.

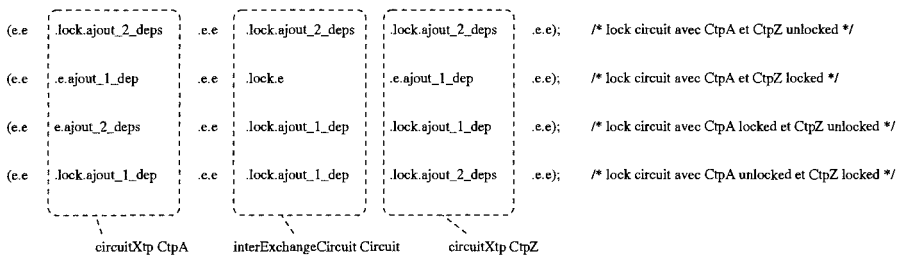


FIG. 6.7 – *Vecteurs de synchronisation*

La principale difficulté dans cette modélisation a été de condenser en un seul vecteur de synchronisation, le déclenchement de plusieurs règles découlant d'une seule. Par exemple, dans le premier vecteur de la figure 6.7, le blocage administratif du circuit entraîne celui des deux points de terminaison CtpA et CtpZ et l'ajoût d'une dépendance à chacun de ces points. Les blocages administratifs sont modélisés par les trois actions `lock` du vecteur. Mais le blocage administratif d'un point de terminaison entraîne également l'ajoût d'une dépendance aux deux objets associés, ce blocage s'effectuant sur chaque point de terminaison, on doit ajouter deux dépendances pour chaque objet : l'une provenant du blocage du circuit et se propageant sur les deux points de terminaison, l'autre provenant, d'une part pour chaque point de terminaison, du point de terminaison opposé, d'autre part pour le circuit, de chaque point de terminaison. La figure 6.8 illustre la construction du premier vecteur de la figure 6.7. Les flèches indiquent une propagation due à la règle **R1**, le vecteur final est obtenu par "addition" des vecteurs individuels.

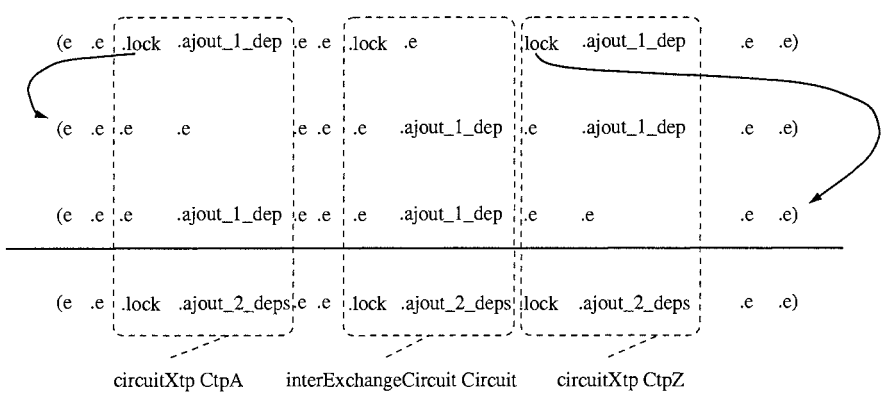


FIG. 6.8 – *Construction d'un vecteur de synchronisation*

La figure 6.9 reprend les différentes étapes de cette étude. Le document du NMF nous a fourni les classes GDMO et les règles, triées par relation de dépendances. Les classes nous ont facilement conduits à spécifier des STEs pour chacune d'entre elles. Les relations de dépendances ont permis de rassembler plusieurs instances d'objets gérés dans des configurations formalisées par des systèmes de transitions. Les règles associées aux dépendances ont guidé l'écriture des vecteurs de synchronisation. Ces trois éléments sont alors suffisants pour générer un produit synchronisé. Ce STE est utilisé comme source de données sur laquelle on peut valider des propriétés spécifiées avec la syntaxe définie par MEC. Lorsque

les résultats de la validation n'étaient pas satisfaisants, il a fallu modifier le comportement au niveau des vecteurs de synchronisation, jusqu'à obtenir statisfaction.

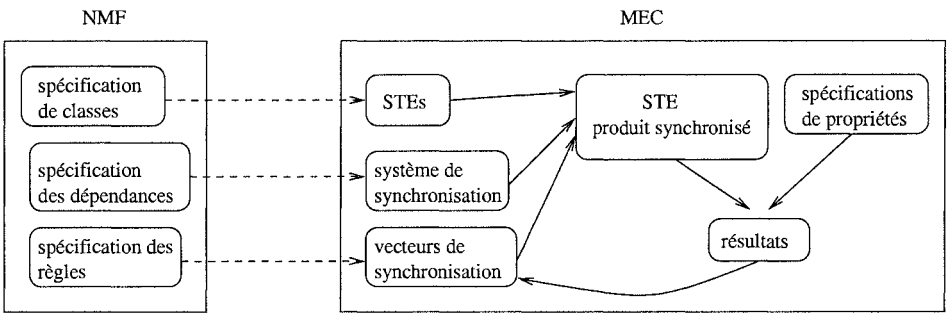


FIG. 6.9 – *Modélisation par synchronisation de STEs*

6.4.3 Les résultats

Le premier résultat est la formalisation du comportement lié aux dépendances entre objets gérés. Après avoir traduit les règles de comportement en un système sychronisé, nous nous sommes interessés à la vérification de propriétés de la configuration d'objets modélisée:

- absence de situation de blocage (*deadlock*),
- retour à un état initial toujours possible,
- impossiblité d'avoir un circuit opérationnel relié à au moins un point de terminaison non opérationnel,
- impossiblité d'avoir un circuit non opérationnel lié à des points de terminaison opérationnels.

Nous avons également identifié un cas particulier de propagation qui n'était pas spécifié. Le cas en question se produit sur la configuration où un équipement contient un point de terminaison physique qui contient lui même un point de terminaison de circuit, ce dernier étant associé avec un circuit. Le blocage administratif de l'équipement entraîne ceux des points (physique et circuit), mais le point de terminaison de circuit peut alors être débloqué par l'intermédiaire du circuit, sans prendre en compte le blocage de l'équipement et du point de terminaison physique.

Le modèle de la figure 6.5 a conduit à 24 vecteurs de synchronisation. Le produit synchronisé résultant a 64 états et 544 transitions. Un autre modèle contenant deux instances supplémentaires a donné un STE de 2048 états et 25088 transitions. La conclusion que nous pouvons donner de cette expérience est positive. Ce type de modélisation force le spécifieur à penser complètement son modèle et l'outil logiciel MEC permet d'en valider les propriétés désirées. Toutefois, le niveau de complexité d'un système d'objets peut rapidement submerger le programme. Il ne reste plus alors qu'à couper son modèle en plus petites parties qui soient traitables. Dans ce cas, la division doit s'efforcer de produire des sous-groupes d'objets les plus indépendants possibles.

La principale difficulté de la modélisation avec MEC a été de traduire sur un seul vecteur de transition, l'ensemble des règles de dépendance qui s'appliquent aux objets gérés modélisés. En outre, la formalisation des vecteurs n'est pas facile à relire, la distinction entre les différents objets se fait grâce aux positions des systèmes de transitions dans le système de synchronisation. Une mise en forme des spécifications (identation, commentaires) s'avère alors indispensable pour corriger le modèle.

6.5 Automates étendus communicants

Parallèlement à l'expérience menée avec MEC, nous avons modélisé le même document du NMF [NMF94] avec une FDT dont les caractéristiques sont plus proches du domaine de la gestion OSI. Le langage LOBSTERS [CF93] [Fes94] [Fes95] permet d'étendre GDMO avec une description formelle du comportement. Les spécifications LOBSTERS sont intégrées dans GDMO en se substituant aux formulaires de comportement informel (`BEHAVIOUR`). Une autre extension définie par LOBSTERS consiste en la définition d'interfaces de communication. Une classe GDMO déclare une ou plusieurs de ces interfaces et plusieurs classes GDMO peuvent partager les mêmes interfaces de communication. Dans ce cas, des messages peuvent circuler entre les objets gérés modélisés.

Concepts de LOBSTERS

La sémantique de LOBSTERS est formalisée par le langage CRS[45] [MNM87] [Sch92] (dans lequel les spécifications LOBSTERS sont traduites) qui permet la modélisation d'ensembles de règles, regroupées en systèmes pouvant communiquer entre eux par des interfaces de communication (appelées des portes). Un système est la modélisation d'un automate à états finis étendu par des variables. Les règles qui déclenchent les transitions sont de la forme Condition/Action.

En regroupant les spécifications LOBSTERS suivant l'algorithme donné dans [Fes95], on parvient à modéliser chaque classe GDMO par un système de règles CRS. Les interfaces de communication LOBSTERS sont traduites par des portes de communication entre systèmes de règles. Les systèmes peuvent être instanciés plusieurs fois et toutes les instances d'un même système partagent alors les mêmes portes.

6.5.1 Les interfaces de communications

La deuxième étape de la modélisation du système, après avoir modélisé les classes GDMO en systèmes de règles CRS, a été la spécification des interfaces de communication entre les classes GDMO. Pour cela, nous avons répertorié ces classes suivant leur participation à des relations de dépendance et leur avons associé une interface de communication pour chaque participation. La figure 6.10 montre l'ensemble des classes GDMO modélisées et leurs interfaces, représentées par un trait reliant deux classes au travers des interfaces de communication qui y sont déclarées. On voit par exemple que la classe `circuitXtp` modélisant un point de terminaison participe à trois relations : une avec le circuit `interExchangeCircuit` pour la dépendance de connectivité (interface *CircuitIface*); une avec le point de terminaison physique `trailTerminationPointBidirectional` pour la dépendance de contenance (*TtpIface*) et une avec le sous groupe de points de terminaison `circuitSubGroupTerminationPoint` pour la dépendance de groupe (*CsTpBelongIface*). Une classe particulière a été définie pour modéliser un opérateur de gestion. Sa fonction est de simuler les indications du service CMIS, limitées à celles visant à positionner la valeur de l'attribut d'état administratif des objets. Toutes les classes GDMO ont donc une interface avec l'objet `operator` (interface nommée *MgmtIface*), c'est par ces interfaces que l'on initialisera les simulations.

6.5.2 La modélisation des règles de comportement

Les règles de comportement liées aux dépendances sont spécifiées dans les classes GDMO. Elles sont rajoutées aux règles de comportement propres aux classes GDMO et indépendantes des relations de

45. Communicating Rule Systems

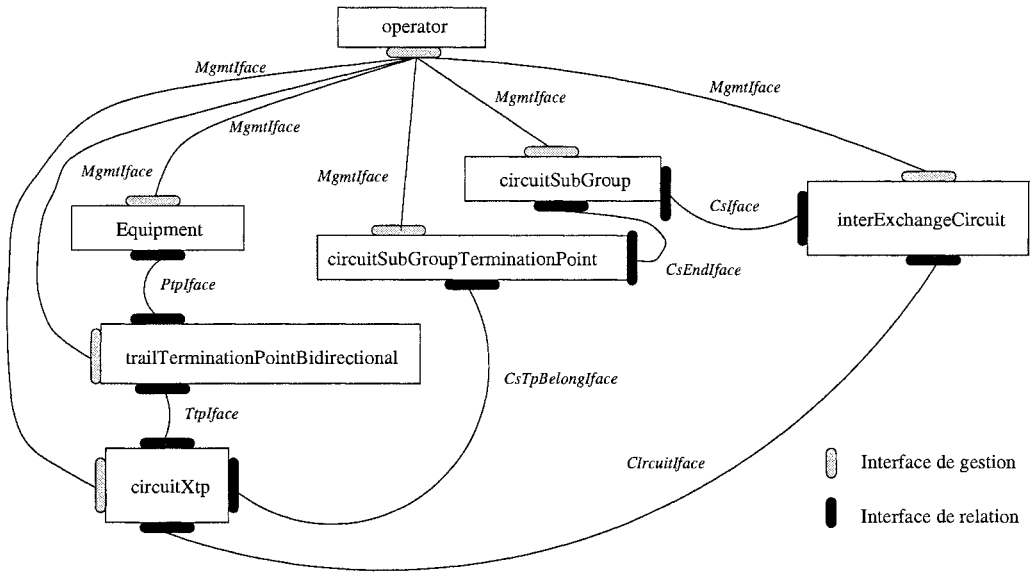


FIG. 6.10 – *Les objets LOBSTERS du modèle*

dépendance. La figure 6.11 explicite quelques règles de la modélisation de la classe `interExchangeCircuit`. Une instance du système de règles CRS qui la modélise est en relation avec deux intances du système de règles CRS modélisant la classe `circuitXtp`. Cette relation est réalisée par l'intermédiaire d'une porte nommée `porte_t_ctp` qui véhicule des messages du circuit vers les points de terminaison. La règle `glock` est évaluée lorsque l'interface de gestion porte le message `lock(id)` avec `id` étant l'identificateur du circuit dans lequel est spécifiée cette règle (ligne 3) Les effets sont de mettre l'état administratif à la valeur `locked` (affectation de variable à la ligne 5) puis de positionner des variables (lignes 6 à 9) qui vont déclencher d'autres règles. Parmi elles, les règle `az_lock_prop` et `az_disable_prop` sont déclenchées par les lignes 7 et 8 qui valident les conditions des lignes 11 et 16. Dans la première de ces règles, on propage le changement de valeur de l'état administratif aux points de terminaison associés (ligne 13) . La seconde règle demande la mise à un état non opérationnel de ces mêmes points (ligne 18).

Au niveau des points de terminaison (figure 6.12), la porte de communication véhiculant les messages depuis le circuit est nommée `porte_f_circuit`, elle correspond à la porte `porte_t_ctp` déclarée dans le circuit. Une autre porte communique les messages vers le circuit (les portes sont unidirectionnelles); la porte `porte_t_circuit`. La règle `lockProp` est déclenchée lors de la propagation du blocage administratif du circuit (ligne 13 de la figure 6.11), les effets sont d'assurer que l'état administratif sera bien à la valeur demandée, lignes 4-5 et 6, et d'autoriser la propagation d'une dépendance (ligne 7). Cette autorisation déclenche la règle `issue_disable` qui va demander au circuit de passer dans un état non opérationnel (ligne 18). Enfin, l'émission du message de propagation de dépendance venant du circuit (ligne 18 figure 6.11) est traitée par la règle `disable` qui ajoute une dépendance au point de terminaison.

La spécification traitant complètement l'exemple du §6.3 est décrite dans [FNA96b] Les spécifications CRS ont été traitées par l'outil CRUSADE qui permet une simulation du modèle. L'outil permet de déclencher interactivement, dans une configuration de plusieurs systèmes de règles, une transition parmi celles dont les conditions de déclenchement sont réunies. Nous avons simulé les scénarios relatifs à la constitution de connexions et de groupes de connexions donnés dans le document du NMF . Ces simulations ont mis à jour les manques du document concernant l'initialisation de variables et nous avons pu

```
(1)   glock;
(2)   COND:mgmt.lock(id)
(3)      AND attached = 1
(4)      AND identifier = id;
(5)   EFFECTS:administrativeState = locked
(6)      AND vz_to_lock = 1
(7)      AND va_to_lock = 1
(8)      AND va_to_disable = 1
(9)      AND vz_to_disable = 1;

(10)  az_lock_prop;
(11)  COND:va_to_lock = 1;
(12)  EFFECTS:va_to_lock = 0
(13)     AND porte_t_ctp.lock(ep)
(14)     AND ep = 'aCtp;

(15)  az_disable_prop;
(16)  COND:va_to_disable = 1;
(17)  EFFECTS:va_to_disable = 0
(18)     AND porte_t_ctp.disable(ep)
(19)     AND ep = 'aCtp;
```

FIG. 6.11 – *Les règles de propagation de message dans le circuit*

```
(1)  lockProp;
(2)  COND: porte_f_circuit.lock(id)
(3)        AND id = identifier;
(4)  EFFECTS: IF 'administrativeState = locked
(5)           THEN TRUE
(6)           ELSE administrativeState = locked
(7)                AND propagate_disable = 1
(8)           FI;

(9)  disable;
(10) COND: porte_f_circuit.disable(id)
(11)       AND id = identifier;
(12) EFFECTS: operationalState = disabled
(13)       AND availabilityStatus = 'availabilityStatus + 1;
(14)
(15) issue_disable;
(16) COND: propagate_disable = 1;
(17) EFFECTS: propagate_disable = 0
(18)       AND porte_t_circuit.disable(id)
(19)       AND id = identifier;
```

FIG. 6.12 – *Les règles de propagation de message dans les points de terminaisons*

pallier ces manques dans des règles CRS déclanchées à la création des instances.

6.6 L'étude avec LDS

L'utilisation de LDS pour formaliser les spécifications GDMO [BLR93], ASN.1 [FS93b] [Kar93] et le comportement des objets gérés [MMP93].

Une troisième expérience de modélisation du document du NMF est menée avec LDS dans [AFN+97] [TAF97]. La spécification du modèle de gestion de l'interconnexion des commutateurs avec LDS a été faite dans un premier temps d'une manière comparable à LOBSTERS. Concernant les relations entre objets, les portes de communication de ce formalisme ont leur équivalent avec les canaux de communication de LDS. Les objets sont modélisés par des processus LDS, ayant une sémantique d'automate à états finis étendu. Un canal défini entre deux processus LDS est partagé entre toutes les instanciations de ces processus. Il transmet des événements reconnus par ces derniers. La version orientée objet de LDS, LDS'92, rajoute le concept d'appel de procédure à distance entre processus. La spécification, dans un deuxième temps, du modèle de gestion en LDS'92 a été allégée des canaux et des événements. Les mêmes résultats ont été obtenus. Afin de faciliter la traduction, un outil de compilation GDMO/GRM vers des squelettes LDS a été développé [Tat97].

L'outil ObjectGeode[46] accepte les spécification LDS et en permet l'animation, notamment par une méthode de simulation exhaustive où sont déclanchées toutes les transitions qui peuvent l'être. Une telle simulation sur une modélisation en LDS de la configuration du NMF s'est déroulée sans mener à des situations de blocages ou incohérentes.

6.7 Conclusions

Les conclusions que nous tirons de ces expérimentations sont multiples. D'une part, les objectifs que nous nous étions fixés sont atteints; le même modèle a pu être formalisé avec trois FDTs, en prenant en compte la notion d'objet, de relation et de gestionnaire. La notion de relation à été exprimée différemment suivant les FDTs, étant plus ou moins intégrée dans celle des objets. Enfin, l'utilisation d'outils s'est dans tous les cas avérée être une aide précieuse.

D'autre part, l'utilisation de différentes FDT nécessite d'aborder le même problème avec des concepts différents. Ces différentes vues ne sont pas sans relation entre elles, par exemple la validation avec MEC nous a fait découvrir qu'une propagation n'était pas spécifiée, celle ci a donc été intégrée dans les règles LOBSTERS. Les outils que nous avons choisis permettent par leur relative facilité d'utilisation de faire un prototypage rapide du modèle. Cette facilité tient au fait que les règles de comportement étaient simples, l'outil prenant véritablement sa valeur lorsqu'il s'agit de valider ou de simuler la cohabitation de plusieurs de ces règles.

Au regard de plusieurs catalogues d'objets gérés, bon nombre de comportements sont liés à des relations et reposent sur le même principe que dans le document du NMF, c'est à dire des variations de valeurs d'attributs qui entraînent d'autres variations sur d'autres attributs. Nos travaux montrent que la modélisation de ces système simples par une FDT nécessite la traduction des concepts de classe, de relation et de comportement de la gestion OSI en des concepts de la FDT. Ces concepts ne sont généralement pas explicites dans les FDT et il faut les rechercher. Nous avons travaillé sur trois FDTs et il en existe probablement d'autres qui auraient pu être utilisées (LOTOS, ESTELLE, Z, π-calcul, ...).

46. de la société Verilog

Chapitre 7

Une syntaxe pour normaliser le comportement des relations gérées

Ce chapitre décrit notre participation au processus de normalisation d'un langage formel pour exprimer le comportement des relations gérées. Les spécifications dans le formalisme de ce chapitre peuvent être une source formelle pour les validations expérimentées dans le chapitre précédent. La motivation de ce travail tient également à l'apparition dans le cadre de la normalisation d'une syntaxe pour le comportement des objets gérés. Notre contribution porte sur l'extension du GRM avec un ensemble de constructions pour la spécification du comportement des relations gérées.

7.1 Le comportement des objets gérés avec GDMO+

GDMO+ [ISO96] est un amendement à la norme de GDMO [CCI92b] qui propose une syntaxe pour exprimer le comportement des objets gérés. GDMO+ est défini comme étant une extention de GDMO [Kel96], ce qui signifie que la syntaxe de GDMO+ englobe celle de GDMO et n'y apporte aucun changement. Sa conception s'est faite avec la volonté de conserver certains principes de GDMO, comme la classification des spécifications en formulaires ou l'utilisation de la syntaxe ASN.1. L'intégration des deux formalismes se fait en substituant les formulaires `BEHAVIOUR` de GDMO par des formulaires de GDMO+ suivants :

- CREATE BEHAVIOUR pour spécifier le comportement (pré-conditions, post-conditions et invariants) lié à la création d'un objet géré, les `PACKAGE`s présents et des valeurs intiales pour leurs attributs;
- DELETE BEHAVIOUR pour spécifier le comportement liée à la suppression d'une instance d'objet géré;
- ATTRIBUTE BEHAVIOUR pour l'évaluation d'un attribut (vu plus loin);
- ACTION BEHAVIOUR pour spécifier le comportement lié à une invocation d'action;
- MULTIPLE REPLY ACTION BEHAVIOUR pour spécifier le comportement lié aux réponses multiples, suite à l'invocation d'une action. Les conditions sont spécifiées pour la première réponse, celles qui suivent et pour la dernière;
- NOTIFICATION BEHAVIOUR pour spécifier le comportement lié à l'émission d'une notification;
- OBJECT BEHAVIOUR pour spécifier le comportement lié à la réception d'événements par un objet géré;
- PARAMETER BEHAVIOUR pour spécifier des conditions qui doivent être vérifiées lors de l'insertion d'un paramètre dans une requête de service CMIS (précisée également dans le formulaire).

Se rajoutent à ces formulaires, deux autres qui ne peuvent être référencés que depuis un formulaire GDMO+; le premier est pour la description d'événements (formulaire EVENT) émis ou perçus par un objet géré, par exemple `m-Action` signifie que l'objet géré a reçu un appel pour une de ses ACTION. Le second permet la définition de fonctions diverses (formulaire de MACRO), dont les paramètres d'appel et de retour sont typés en ASN.1. Par exemple la MACRO `SuperiorObject` retourne l'objet géré[47] supérieur à celui qui utilise cette MACRO dans un de ses formulaire GDMO+. La table 7.1 donne les types de formulaires GDMO+ qui peuvent être référencés comme BEHAVIOUR dans les formulaires GDMO. Un inconvénient d'intégrer GDMO+ de cette manière est que les classes d'objets gérés ne peuvent pas référencer directement du comportement. L'intérêt en serait de spécifier des conditions entre les PACKAGEs – et leurs caractéristiques – d'une même classe.

GDMO	GDMO+
NAME-BINDING	CREATE BEHAVIOUR DELETE BEHAVIOUR
ATTRIBUTE	ATTRIBUTE BEHAVIOUR
ACTION	ACTION BEHAVIOUR MULTIPLE REPLY ACTION BEHAVIOUR
NOTIFICATION	NOTIFICATION BEHAVIOUR
PACKAGE	tous sauf CREATE et DELETE BEHAVIOUR, plus OBJECT BEHAVIOUR et PARAMETER BEHAVIOUR

TAB. 7.1 – *Formulaires GDMO+ référencés par les formulaires GDMO*

A titre d'exemple de formulaire GDMO+, la figure 7.1 montre celui qui est dédié à la spécification du comportement des attributs. La première clause du formulaire (COMMENT ligne 2) laisse une place

```
(1)<attribute-behaviour-label> ATTRIBUTE BEHAVIOUR
(2)       [COMMENT delimited-string ;]
(3)
(4)       [ATTRIBUTES <attribute-label> [, <attribute-label>]*;
(5)       ]
(6)       SELF VARIABLE <variable-name>;
(7)
(8)       [COMPARISON VARIABLE <variable-name>;
(9)
(10)         [EQUALITY condition-definition;]
(11)         [ORDERING condition-definition;]
(12)         [SUBSTRINGS      [INITIAL condition-definition ]
(13)                          [ANY condition-definition]
(14)                          [FINAL condition-definition];
(15)         ]
(16)         [SET-COMPARISON condition-definition;]
(17)         [SET-INTERSECTION condition-definition;]
(18)      ]
(19)      [INVARIANTS  condition-description
(20)         [, condition-description]*];
(21)      ];
```

FIG. 7.1 – *Le formulaire de comportement d'attribut*

à un commentaire informel (cette clause est présente dans tous les formulaires de GDMO+). La liste d'attributs à la ligne 4 est utilisée lorsque ce fomulaire est référencé depuis un formulaire GDMO de

47. type ASN.1 ObjectInstance

PACKAGE, pour spécifier quels attributs du PACKAGE sont concernés par le comportement. La ligne 6 permet de donner un nom unique représentant l'attribut, la SELF VARIABLE. Les lignes 8 à 18 décrivent comment doivent être évaluées les différentes comparaisons possibles entre deux valeurs d'attributs. Les comparaisons sont exprimées entre la SELF VARIABLE et la COMPARAISON VARIABLE (ligne 8). L'évaluation des comparaisons donne un résultat booléen qui traduit si la première est égale à la seconde (ligne 10), supérieure (11), une sous chaîne (lignes 12 à 15) et pour les attributs à valeur ensembliste, s'il y a égalité (ligne 16) ou si l'intersection est non vide (ligne 17). Enfin, la clause INVARIANTS (lignes 19 et 20) permet de spécifier des conditions portant sur les valeurs de la SELF VARIABLE et devant toujours être vraies.

Les expressions qui doivent remplacer les condition-description de la figure 7.1 ont une syntaxe qui est définie dans le cadre de l'amendement GDMO+. L'utilisation de cette syntaxe se fait toujours depuis un formulaire GDMO+, essentiellement par une condition-description dont l'évaluation retourne un résultat booléen. Cette syntaxe est particulièrement intéressante car elle possède des constructions pour manipuler des valeurs ASN.1. On y trouve des expressions qui permettent par exemple de construire des valeurs de type SET, SET OF ou de décomposer ces valeurs en types plus simples. Afin d'illustrer cette syntaxe et le formulaire de la figure 7.1, le comportement de l'atttribut eventTime [ITU92] est formalisé. Le type ASN.1 en est le suivant :

```
EventTime ::= SEQUENCE {
      year   INTEGER
      month  INTEGER{1..12}
      day    INTEGER{1..31}
      hour   INTEGER{0..23}
      minute INTEGER{0..59}
      second INTEGER{0..59}
}
```

Le comportement informel de l'attribut stipule que la comparaison entre deux valeurs doit se faire en commençant par l'année, puis si les années sont identiques par le mois, etc. L'expression formelle avec GDMO+ de la figure 7.2 spécifie comment évaluer l'égalité (lignes 4 à 6) et la comparaison (lignes 7 à 15), les mots clés et symboles AND, IF, THEN, ELSE, ENDIF, <, >, =, (,) ont la sémantique usuelle. On notera l'accès aux champs du type ASN.1 de l'attribut par une notation pointée. La condition portant sur l'ordre entre deux valeurs de ce type d'attribut retourne vrai (TRUE lignes 7 et 9) si la valeur de l'attribut courant est supérieure à celle de l'attribut avec lequel on le compare. La syntaxe d'expression

```
(1) EventTimeBehaviour ATTRIBUTE BEHAVIOUR
(2)   SELF VARIABLE eTime;
(3)   COMPARISON VARIABLE cTime;
(4)     EQUALITY eTime.year = cTime.year AND
(5)              eTime.month = cTime.month AND
(6)              ...;
(7)     ORDERING IF (eTime.year > cTime.year) THEN TRUE
(8)              ELSE IF (eTime.year < cTime.year) THEN FALSE
(9)                   ELSE IF (eTime.month > cTime.month) THEN TRUE
(10)                       ELSE IF (eTime.month < cTime.month) THEN FALSE
(11)                            ...
(12)                            ENDIF
(13)                       ENDIF
(14)                  ENDIF
(15)             ENDIF;
```

FIG. 7.2 – *Le comportement de l'attribut* eventTime

de GDMO+ à été formalisée en Z [Kel96]. Elle repose sur d'autres spécifications Z qui modélisent les formulaires GDMO [KD94].

7.2 Le Comportement des relations gérées avec GRM+

7.2.1 Comportement normalisé du GRM

La norme [CCI95] propose d'utiliser des formulaires de type BEHAVIOUR, définis dans GDMO, pour spécifier le comportement des classes et des représentations de relations gérées. Une instance (ou relation gérée) est issue à la fois d'une classe et d'une représentation. Son comportement est donc exprimé dans ces deux contextes. Bien qu'étant de nature informelle, le comportement des relations gérées doit pouvoir être divisé en:

- comportement invariant qui est toujours vérifié (i.e. vrai) durant l'existance de la relation gérée;
- comportement de pré-condition qui est d'une part lié à une opération de relation et d'autre part vérifié à la suite de sa requête, avant son exécution;
- comportement d'invariant d'opération, lié à une opération et devant être vérifié durant son exécution;
- comportement de post-condition également lié à une opération de relation, mais vérifié après son exécution.

La table 7.2 reprend de la norme les définitions comportementales concernant les différents prototypes d'opérations de relation (sauf USER-DEFINED qui est totalement dépendant des définitions des opérations des utilisateurs). Toute opération de relation récupère le comportement de son prototype et définit

opérations	invariant	pré-condition	post-condition
ESTABLISH	les cardinalités sont respectées	la MR[a] n'existe pas et les MOs[b] spécifiés sont des classes appropriées	la MR existe, les MOs spécifiés existent et sont liés à la MR
TERMINATE	-	la MR existe	la MR n'existe plus et les MOs qui étaient liés à la MR ne le sont plus
BIND	la MR existe et les cardinalités sont respectées	les classes des MOs spécifiés sont celles permises par le rôle et ce dernier supporte l'opération BIND	les MOs spécifiés existent et sont liés à la MR
UNBIND	la MR existe et les cardinalités sont respectées	les MOs spécifiés existent et sont liés à la MR et le rôle supporte l'opération UNBIND	les MOs spécifiés ne sont plus liés avec la MR
QUERY	-	toujours vrai	la MR est inchangée
NOTIFY	-	toujours vrai	la MR est inchangée

[a] Managed Relationship : une relation gérée
[b] Managed Object : un objet géré

TAB. 7.2 – *Définition générique des opérations de relation*

le sien. Toutefois, le terme de pré-condition utilisé par la norme ne doit pas faire croire que les pré-conditions d'un prototype puissent être affaiblies lors d'une redéfinition d'une opération dans une sous-classe. Au contraire, les pré-conditions et tous les autres comportements de prototypes d'opérations, sont les comportements minimaux que les opérations ne peuvent que renforcer.

Avant de formaliser le comportement par une extention du GRM, il faut prendre en compte celui qui est formellement exprimé avec les notations normalisées. Les comportements des prototypes d'opérations

précédents en sont une partie, les cardinalités en sont une autre. En effet le maintien du nombre de participants dans les valeurs permises et requises de la cardinalité d'un rôle est une composante de l'invariant de la relation. Il en résulte que les opérations de type ESTABLISH, BIND ou UNBIND peuvent être utilisées sans craindre de violer les cardinalités des rôles.

La cardinalité de relation d'un rôle est particulière car elle peut imposer un comportement aux objets gérés lorsque sa valeur minimale est de 1. Dans ce cas, la relation gérée ne peut être supprimée (TERMINATE) qu'en impliquant la destruction (service M-DELETE) des objets participants à ce rôle.

Enfin, les attribut qualifiants ont eux même un comportement qui fait partie intégrante du comportement invariant de la relation.

Le principe adopté par GDMO+ pour intégrer ses formulaires dans GDMO est également valable pour le comportement du GRM. Nous avons donc défini de nouveaux formulaires de spécifications dédiés au comportement. Un pour les classes de relations et un pour les représentations que nous regroupons sous le terme de GRM+. Les spécifications avec ces formulaires sont référencées dans les classes et les représentations de relations gérées, à la place des comportements informels. Nous n'avons pas prévu de formulaire spécifique pour les rôles bien qu'ils puissent être définis séparément des classes de relations. Il n'ont pas de construction pour le comportement mais surtout, au vu des exemples existants et de notre propre expérience, la réutilisation des rôles n'est pas justifiée.

Du point de vue de la normalisation, il a fallu intégrer GRM+ comme un ajoût (un *addendum*) à l'amendement de GDMO que constitue GDMO+. Ce choix permettait de se positionner dans le groupe de travail déjà créé pour GDMO+, au lieu de demander la création d'un nouveau groupe. Notre proposition acceptée comme *addendum* [NR97], comporte quelques corrections apportées à GDMO+ et principalement les définitions de formulaires présentés dans les sections suivantes.

7.2.2 Comportement d'une classe de relation

Indépendamment de son appartenance à l'une des catégories de comportements précédentes, le comportement d'une classe de relation peut exprimer des contraintes entre les données suivantes:

- le nombre de participants dans un rôle d'une relation gérée (devant rester dans les cardinalités définies pour le rôle);
- le nombre de relations gérées d'une même classe auxquelles participe simultanément un participant dans le même rôle;
- les valeurs des attributs qualifiants;
- les valeurs des attributs connus dans les participants à un rôle (i.e. ceux dans sa classe compatible);
- la participation simultanée de participants à plusieurs rôles d'une même relation gérée.

Le formulaire qui est défini par la suite permet de spécifier des contraintes par catégories de comportements. Par exemple, si une même contrainte fait partie de la pré-condition d'une opération et de la post-condition d'une autre opération, elle devra figurer deux fois dans le comportement de la classe de relation. La syntaxe d'expression définie dans GDMO+ convient pour les contraintes, des MACRO spécifiques à GRM+ seront présentées au travers d'exemples.

Le formulaire de comportement des classes de relations

La figure 7.3 montre le formulaire de comportement des classes de relations. Ce formulaire est référencé dans leur clause BEHAVIOUR par des formulaires de classes de relations gérées. La clause INVARIANTS ligne 3 permet de définir des conditions invariantes pour les relations gérées. La virgule qui sépare deux expressions condition-description vaut pour un ET logique. Ceci facilite l'écriture en séparant des conditions qui relèvent des points ci-dessus. La clause OPERATIONS ligne 6 marque le début des définitions de comportement liées aux opérations de relation.

```
(1) <relationship-behaviour-label> RELATIONSHIP CLASS BEHAVIOUR
(2)     [COMMENT delimited-string];
(3)     [INVARIANTS condition-description
(4)                 [, condition-description]*;
(5)     ]
(6)     [OPERATIONS relationship-operation
(7)                 [relationship-operation]*
(8)     ]
(9);
```

FIG. 7.3 – *Le formulaire de comportement de classe de relation*

La figure 7.4 détaille le formulaire pour ces opérations. La ligne 2 de cette figure permet d'identifier de manière non ambiguë toute opération d'une relation, par son type générique (lignes 15 et 16) et si besoin, un label spécifique donné dans la classe, le `relationship-operation-label`. Les types référencés aux lignes 3 et 4 sont des définitions en ASN.1 pour les arguments de l'appel et du retour de l'opération. Le comportement de l'opération est exprimé en terme de pré-conditions (lignes 5 à 7), invariants (lignes 8 à 11) et post-conditions (lignes 11 à 13). Les invariants doivent être vrais durant l'exécution de l'opération. Les expressions `precondition-description` et `postcondition-description` ont, comme `condition-description`, un résultat booléen. Lorsqu'une sous-classe redéfinit une opération, les règles d'héritage strict doivent être suivies de la même manière que s'il s'agissait d'une opération de classe d'objet. Le nouveau comportement est le résultat de la disjonction des pré-conditions et de la conjonction, des post-conditions et des invariants, de l'opération dans la sous classe avec celles respectivement définies dans ses super-classes. Le type de l'argument d'appel doit être un super-type de ceux de ses super-classes et celui de l'argument de retour doit en être un sous-type.

```
(1) relationship-operation -->
(2)                 operation-type [<relationship-operation-label>]
(3)                 [WITH-INFO type-reference]
(4)                 [WITH-RETURN type-reference]
(5)                 [FOR precondition-description
(6)                       [, precondition-description]*
(7)                 ]
(8)                 [WITH-INVARIANTS condition-description
(9)                       [, condition-description]*
(10)                 ]
(11)                 [WITH post-condition-description
(12)                       [, post-condition-description]*
(13)                ]
(14)
(15) operation-type -->    ESTABLISH | TERMINATE | QUERY | BIND | UNBIND |
(16)                       NOTIFY | USER-DEFINED
```

FIG. 7.4 – *Comportement des opérations*

Application à un exemple

La classe `traffic` du §4.4 au chapitre 3 réfère le formulaire `trafficBehaviour` se trouvant à la figure 7.5. Ce comportement définit des invariants (ligne 2) de la classe; le premier stipule que les objets gérés participant au rôle `destination` ne peuvent se trouver en même temps dans le rôle `source`. La sémantique de l'expression est explicitée dans les points suivants :

- Le mot clé `FORALL` (ligne 3) signifie que la condition exprimée après le symbole | doit être vraie pour chaque valeur du domaine d'itération de la variable suivant le `FORALL (x)`.
- La ligne 3 initialise également le domaine d'itération de la variable `x`; il est constitué de l'ensemble

des identificateurs d'objets gérés participant au rôle `destination` (donné par `role(destination)`).

- La condition de la ligne 4 est vraie lorsque la valeur de `x` n'est pas dans l'ensemble des participants au rôle `source` (en l'occurence, l'ensemble est un singleton).

Le type de `x` est donné par la construction `role(destination)` qui correspond à l'appel d'une MACRO GRM+ `role` avec un nom de rôle comme paramètre. Ce nom (ou un identificateur, optionnel dans la norme) doit être celui d'un des rôles de la relation. Le type de retour de cette instruction est un ensemble ASN.1 d'identificateurs d'objets gérés (leur *Distinguish Name*). Cette MACRO fournit une abstraction de la notion de rôle qui est largement utilisée dans nos spécifications GRM+.

La deuxième condition (lignes 5 à 12) exprime une équivalence logique (symbole <=>ligne 11) entre les valeurs des attributs `operationalState` contenus dans les participants et dans la relation. Le symbole `->` permet d'accéder à la valeur d'un attribut d'un objet géré à partir de son *DN* et la liste de variables (ligne 8) est une commodité du langage pour exprimer des itérations imbriquées. La MACRO GRM+ `package` (lignes 5 et 6) qui est évaluée à vrai lorsque l'identificateur de PACKAGE GDMO donné en premier paramètre est instancié dans tous les objets gérés qui participent au rôle donné en second paramètre. Cette instruction permet aux spécifications de s'adapter à la présence ou l'absence de modules conditionnels dans les objets gérés. La macro `package` permet d'éviter une multitude de classes de relations dérivées, tout comme le font les modules conditionnels pour les classes d'objets gérés. La suite du formulaire concerne les opérations de relation (lignes 13 à 51):

- le comportement de l'opération `ESTABLISH` (lignes 14 à 28) est constitué d'une spécification de type pour l'argument d'appel et de post-conditions. La première (lignes 18 et 19) vérifie que la relation gérée est instanciée avec les participants donnés en arguments. La seconde (lignes 20 à 28) spécifie les valeurs des éventuels attributs d'état opérationnel des participants et de la relation;
- l'opération `lockTraffic` (lignes 29 à 38) contient des post-conditions qui verifient que le blocage administratif s'est propagé à tous les participants;
- l'opération `unlockTraffic` (lignes 39 à 51) a une pré-condition (lignes 40 et 41) qui évite un déblocage non justifié des points de terminaison. Les post-conditions (lignes 42 à 51) stipulent que la valeur de l'attribut `administrativeState`, des participants qui le possède doivent être à `"unlocked"` si la valeur de leur attribut `operationalState` est à `"enabled"`.

7.2.3 Comportement d'une représentation de relation

Le comportement des représentations rajoute à celui des relations les éléments suivants:

- on peut spécifier des contraintes entre tous les attributs des classes de participants car celles ci sont connues;
- on peut spécifier d'éventuels arguments pour les opérations de relation;
- s'il existe des arguments pour les opérations de relation, alors on peut spécifier le passage de ces arguments à ceux des opérations de système (c.a.d. des opérations sur des objets gérés) qui traduisent une opération de relation.

Concernant le premier point, il est inutile, voir dangereux, de modifier les attributs de relation utilisés dans le cas d'une représentation avec ce moyen. La spécification de représentation avec le GRM associe, dans ce cas, un attribut pour référencer les participants à un rôle; de tels attributs sont modifiés explicitement par les opérations de types `BIND` ou `UNBIND` (également `ESTABLISH` ou `TERMINATE`). En modifiant directement ces attributs dans une spécification de comportement avec GRM+, on risque de toucher à l'intégrité de la relation.

```
(1) trafficBehaviour RELATIONSHIP CLASS BEHAVIOUR
(2)   INVARIANTS
(3)     FORALL x IN role(destination) |
(4)          x NOT IN role(source),
(5)     IF (package(operationalStatePackage, source) AND
(6)         package(operationalStatePackage, destination))
(7)     THEN
(8)       (FORALL x,y IN role(source),role(destination) |
(9)          (x -> operationalState = ''disabled'' AND
(10)          y -> operationalState = ''disabled''))
(11)         <=> operationalState = ''disabled''
(12)    ENDIF;

(13)   OPERATIONS

(14)      ESTABLISH
(15)        WITH-INFO SEQUENCE {source ObjectInstance,
(16)                            dest   ObjectInstance}
(17)        WITH
(18)          establishInfo.source IN role(source) AND
(19)          establishInfo.dest IN role(destination),
(20)          operationalState = ''enabled'',
(21)          IF package(operationalState, source) THEN
(22)                 FORALL x IN role(source) |
(23)                   x -> operationalState = ''enabled''
(24)          ENDIF AND
(25)          IF package(operationalState, destination) THEN
(26)              FORALL x IN role(destination) |
(27)               x -> operationalState = ''enabled''
(28)          ENDIF;

(29)      USER-DEFINED lockTraffic
(30)        WITH
(31)          administrativeState = "locked",
(32)          IF package(administrativeState, source) THEN
(33)            FORALL x IN role(source) |
(34)              x -> administrativeState = "locked"
              ENDIF AND
(35)          IF package(administrativeState, destination) THEN
(36)            FORALL x IN role(destination) |
(37)              y -> administrativeState = "locked"
(38)          ENDIF;

(39)      USER-DEFINED unlockTraffic
(40)        FOR
(41)          administrativeState = "locked";
(42)        WITH
(43)          administrativeState = "unlocked",
(44)          IF package(administrativeState, source) THEN
(45)            FORALL x IN role(source) |
(46)               x -> administrativeState = "unlocked"
(47)          ENDIF AND
(48)          IF package(administrativeState, destination) THEN
(49)            FORALL x IN role(destination) |
(50)                x -> administrativeState = "unlocked"
(51)          ENDIF
(52) ;
```

FIG. 7.5 – *Comportement de la classe de relation de trafic*

Le formulaire de comportement des représentations

Le deuxième formulaire (figure 7.6) permet de décrire le comportement d'une représentation de classe de relation. Le formulaire de représentation, le RELATIONSHIP MAPPING, doit référencer ce second formulaire dans sa clause BEHAVIOUR. La structure de ce formulaire est quasiment identique à celle du formulaire de la figure 7.3, la clause OPERATIONS-MAPPING (lignes 6 et 7) étant dédiée aux comportements des traductions d'opérations de relation en opérations de système. La figure 7.7

```
(1) <relationship-behaviour-label> RELATIONSHIP MAPPING BEHAVIOUR
(2)         [COMMENT delimited-string];
(3)                 [INVARIANTS condition-description
(4)                         [, condition-description]*;
(5)                 ]
(6)                 [OPERATIONS-MAPPING relationship-mapping-operation
(7)                         [relationship-mapping-operation]*
(8)                 ]
(9) ;
```

FIG. 7.6 – *Le formulaire de comportement de représentation de relation*

décrit le comportement associé à la traduction d'une opération de relation. Celle ci est spécifiée de la même manière que pour les opérations dans la classe de relation (ligne 2 fig. 7.4 et fig. 7.7). Le type des arguments d'appel et de retour de l'opération sont à spécifier aux lignes 3 et 4. Le `type-reference` doit être un type ASN.1 (ou en référencer un). Lorsqu'il existe déjà des types définis dans le comportement de la classe de relation, le type de l'argument d'appel dans la représentation doit être un super type de celui de la classe et le type de l'argument de retour doit en être un sous type. Ainsi, comme entre les opérations d'une sous-classe par rapport à celles de ses super-classes, on permet à une représentation de spécifier des informations supplémentaires tout en conservant la définition des types d'appel dans la classe. Pour le retour, on garantit la compatibilité avec la classe. La clause MAPS-TO (ligne 5) permet de spécifier un comportement dans une clause `sys-op-mapping` (définie à la ligne 6) pour chaque opération de système constituant la traduction de l'opération de relation. Une opération de système doit encore être distinguée sans ambiguïté en spécifiant le nom (`system-operation`) et le rôle (ligne 7). Les lignes 8 et 9 spécifient les valeurs des arguments de l'opération de système invoquée, l'expression de type `FieldExpression`, dérive en une séquence ASN.1 dont les champs représentent les arguments d'appel des opérations de système. Pour ce faire nous utiliserons des MACROs définies plus loin. L'expression de type `Expression` (lignes 8 et 9) référence des champs dans le type de l'argument de l'appel de l'opération de relation (ligne 3). Les valeurs contenues dans les réponses des opérations de système sont affectées aux variables constituant la réponse de l'opération de relation d'une manière duale. L'expression de type `FieldExpression` (lignes 20 et 21) référence des champs dans le type défini à la ligne 4 et celle de type `Expression` (lignes 20 et 21) contient des références aux types ASN.1 des réponses d'opérations de système. Les lignes 11 à 19 ont la même sémantique que pour les opérations de relation, à la différence qu'elles s'appliquent à une opération de système. La prise en compte de ces comportements avec ceux de l'opération de relation sera vue au §7.2.5.

Le comportement de la représentation de la classe `connectionTrafficMpCross` (cf. fig 4.16 chap. 3) est donné à la figure 7.8. L'invariant (lignes 2 à 5) décrit une contrainte sur des attributs appartenant aux rôles `source` et `destination`. On peut constater, sur la figure 4.10 du chapitre 4 que le couple d'attributs référencé dans cet invariant représente la relation. Le formulaire de représentation (cf. fig 4.16 chap. 3) ne peut formuler qu'une représentation par rôle, c'est donc aux formulaires GRM+ de spécifier les éventuelles représentations supplémentaires entre les rôles. Le formulaire de comportement de représentation étant élaboré dans un contexte (le formulaire de représentation) où les classes des participants sont connues, est plus sujet à la spécification des représentations supplémentaires que le

```
(1) relationship-mapping-operation -->
(2)        operation-type [<relationship-operation-label>]
(3)                  [WITH-INFO type-reference]
(4)                  [WITH-RETURN type-reference]
(5)                  MAPS-TO sys-op-mapping [, sys-op-mapping]* ;

(6) sys-op-mapping -->
(7)         system-operation OF <role-label>
(8)         [ WITH-SERVICE-INFO FieldExpression := Expression
(9)                           [, FieldExpression := Expression]*
(10)        ]
(11)        [FOR precondition-description
(12)                  [, precondition-description]*
(13)        ]
(14)        [WITH-INVARIANTS condition-description
(15)                  [, condition-description]*
(16)        ]
(17)        [WITH post-condition-description
(18)                  [, post-condition-description]*
(19)        ]
(20)        [WITH-SERVICE-RETURN FieldExpression := Expression
(21)                           [, FieldExpression := Expression]*

(22) system-operation --> CREATE | DELETE | GET [<label>] |
(23)                      REPLACE [<label>] | REMOVE [<label>] |
(24)                      ADD [<label>] | ACTION [<label>] |
(25)                      NOTIFICATION [<label>]
```

FIG. 7.7 – *Comportement des traductions d'opérations*

formulaire de comportement des classes de relations.

La partie suivante concerne les opérations de relation (lignes 6 à 27), seule l'opération ESTABLISH est détaillée, la totalité se trouvant en annexe A. Cette opération, dont le paramètre (lignes 7 à 9) est un sous-type de celui défini dans la classe (fig. 7.5), est traduite par deux opérations `CREATE` (lignes 11 et 20). La macro `mappingCreateInfo` retourne une séquence ASN.1 représentant les paramètres du service `M-CREATE` (ici, `superiorObjectInstance` pour l'objet géré supérieur et `attributeList` pour une liste de valeurs initiales des attributs). Les symboles `[` et `]` sont des constructeurs GDMO+ de valeurs de séquence ASN.1 qui sont utilisées ici pour initialiser les valeurs d'attributs des objets gérés créés (lignes 16 à 19 et 23 à 27). Si des paramètres de service CMIS peuvent être déduits des spécifications en GRM, ils peuvent être omis. Ainsi, dans la création du participant au rôle `leg`, la représentation par une corrélation de nom entre ce rôle et le rôle `mpCross` permet d'omettre que l'instance supérieure du nouveau participant au rôle `leg` est le participant au rôle `mpCross`. Cet exemple met également en valeur le fait que plusieurs opérations de système doivent pouvoir se synchroniser. Dans notre cas, l'opération `CREATE` du rôle `leg` ne peut se réaliser que si celle du rôle `mpCross` est terminée avec succès.

7.2.4 Les `MACRO` définies pour GRM+

Nous avons défini des `MACRO`s réservées à l'usage du comportement des relations gérées. Outre les macros `role` et `package`, des `MACRO`s représentant les arguments des appels d'opérations de relation, nommées *xxx*`Info` (avec *xxx* pour `establish, terminate, ...`) et les réponses de ces appels, nommées *xxx*`Result` sont utilisables. Un autre ensemble de `MACRO`s a été nécessaire pour représenter les paramètres des opérations de système. Elles sont nommées `mapping`*yyy* et `mapping`*yyy*`Result` (*yyy* pour `Get, Set, ...`). Ces `MACRO`s d'arguments ne sont utilisables que dans le comportement des représentations et font référence aux types définis dans les expressions des clauses `WITH-INFO`, `WITH-RETURN` et `WITH-SERVICE-PARAMETERS`. La figure 7.9 montre la définition de la `MACRO role`;

```
(1) connexionTrafficMpCrossBehaviour RELATIONSHIP MAPPING BEHAVIOUR
(2)   INVARIANTS
(3)     FORALL x,y IN role(source), role(destinataire) |
(4)       y -> upstreamConnectivityPointer = x AND
(5)       x -> dowstreamConnectivityPointer = role(destination);

(6)   OPERATIONS-MAPPING

(7)     ESTABLISH WITH-INFO SEQUENCE {supOid ObjectInstance,
(8)                                   source ObjectInstance,
(9)                                   dest   ObjectInstance}
(10)      MAPS-TO
(11)       CREATE OF mpCross
(12)        WITH-SERVICE-INFO
(13)         mappingCreateInfo().superiorObjectInstance :=
(14)               establishInfo().supOid;
(15)         mappingCreateInfo().attributeList :=
(16)               [[fromTermination, establishInfo().source],
(17)                [administrativeState, "unlocked"],
(18)                [operationalState, "enabled"],
(19)                [availabilityStatus, {}]],
(20)       CREATE OF leg
(21)        WITH-SERVICE-INFO
(22)           mappingCreateInfo().attributeList :=
(23)                        [[fromTermination, NULL],
(24)                         [toTermination, establishInfo().dest],
(25)                         [directionality, "unidirectional"],
(26)                         [administrativeState, "unlocked"],
(27)                         [operationalState, "enabled"]];
...
(28);
```

FIG. 7.8 – *Comportement d'une représentation*

une telle définition comporte le type de paramètre d'appel (ligne 4) et celui du retour (ligne 5). Un commentaire informel (ligne 2 et 3) et une définition de la macro (lignes 6 à 9) complètent le formulaire. La définition n'est pas obligatoirement textuelle, mais peut utiliser les expressions de GDMO+.

```
(1) role MACRO
(2) COMMENT "Cette macro ne peut être utilisée que dans des
(3)          spécifications GRM+";
(4) VARIABLE _role_ OBJECT IDENTIFIER
(5) MACRO TYPE SET OF ObjectInstance
(6) DEFINITION TEXTUAL
(7)  "Cette macro prend un identificateur de rôle  dans la relation
(8)   et retourne l'ensemble des identificateurs d'instances qui
(9)   participent à ce rôle";
```

FIG. 7.9 – *Définition d'une* MACRO

7.2.5 Classes et représentations

Les comportements des classes et des représentations diffèrent par le fait que le premier exprime des conditions indépendantes d'une représentation particulière, alors que le deuxième exprime toutes les conditions faisant intervenir la connaissance de la représentation. Chaque relation gérée étant issue d'une classe (avec toutes ses super-classes) et utilisant une seule représentation, elle doit donc se comporter suivant le comportement de sa classe et de sa représentation. D'éventuelles contradictions seront donc à surveiller entre des spécifications de comportement des classes et de leurs représentations. Le

comportement d'une relation gérée doit vérifier les points ci-dessous :

- la conjonction des invariants de la classe et de la représentation ne doit pas être toujours fausse (il doit être possible de l'évaluer à vraie);
- pour chaque opération de relation, la conjonction de ses pré-conditions avec la conjonction des pré-conditions de la première opération de système ne doit pas être toujours fausse;
- les post-conditions de la dernière opération de système et les post-conditions de l'opération de relation ne doivent pas être toujours fausses.

7.3 Conclusions

Ce chapitre a présenté un formalisme pour le comportement des relations. D'une manière générale, la conception des modèles de gestion gagne à spécifier le comportement dans les classes de relations qui serait soit exprimé dans un autre formalisme (ou sans formalisme) séparément des modèles, soit dans les classes d'objets. Or il n'est pas possible de prévoir à l'avance toutes les relations qui vont relier les objets dont les classes sont issues de modèles génériques, surtout si ceux-ci sont normalisés bien avant l'apparition d'applicatifs de gestion. De nouvelles classes peuvent s'ajouter à celles qui sont réutilisées depuis les modèles génériques. Ces nouvelles classes peuvent alors avoir des relations avec les "anciennes" et modifier leur comportement. L'unique solution, avec un modèle purement objet, est alors de dériver une classe du modèle générique et d'adapter son comportement à la relation. Cependant, le comportement propre de l'objet (celui qui concerne ses attributs, à l'exclusion de ses attributs de relation) n'est pas forcément modifié à chaque nouvelle relation avec une nouvelle classe. Cette solution ne facilite pas la compréhension des modèles de gestion. A l'opposé, les relations et leur comportement permettent d'isoler le comportement propre aux objets qui ont alors une sémantique plus claire [FN97].

Le formalisme que nous avons proposé s'insère parfaitement dans le GRM et donc participe activement à l'amélioration des modèles de gestion. L'intégration du GRM+ avec GDMO+ constitue un ensemble de formalismes homogènes sur plusieurs concepts, comme l'utilisation d'une même syntaxe d'expression ou le procédé d'intégration avec les formalismes GRM et GDMO. La figure 7.10 positionne les différents formalismes du modèle de l'information de la gestion OSI. Le GRM est défini comme une extension de GDMO. GDMO+ est lui même une extension de GDMO et le GRM+ une extension du GRM. Nous regroupons les deux formalismes GDMO+ et GRM+ pour constituer une extension au GRM.

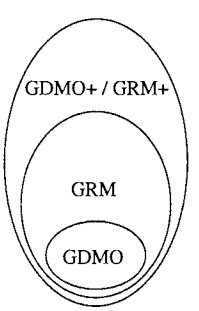


FIG. 7.10 – *Extensions des formalismes*

La formalisation du comportement des relations gérées permet de traiter des cas complexes où les rôles et les interactions sont nombreuses. Comme dans ce chapitre, l'annexe A ou dans [Nat97]. L'utilisation du GRM+ se place dans une approche ascendante, où l'on part de classes d'objets pour en extraire

des configurations types, que l'on transforme en classes de relations et en représentations. L'héritage entre les relations ou leur réutilisation peut alors accompagner la réutilisation de modèles génériques de gestion en allégeant les modèles dérivés de classes d'objets crées pour remplacer les relations. Le comportement des relations permet de dépasser les limites formelles de l'héritage entre les relations du GRM. Un nouvel invariant ou une nouvelle pré-condition, par exemple, donnent lieu à une nouvelle classe de relation, en plus des autre moyens propres au GRM (ajoût d'un rôle, changement de cardinalité, etc.).

Chapitre 8

RelMan : une approche pour l'administration des relations gérées

Ce chapitre présente notre approche de l'intégration des relations gérées dans un environnement standard de gestion OSI. Les deux chapitres précédents ont analysé, puis formalisé le comportement de ces relations. Il en ressort que les spécifications de modèles de gestion avec les formulaires GRM et GRM+ sont entièrement formelles et peuvent être exploitées par des outils logiciels. Ici, nous définissons les fonctionnalités d'un tel outil, dont le but est de permettre l'administration des objets gérés au travers de relations gérées. RelMan[48] est le nom qui englobe la description des concepts dégagés [NFA97].

8.1 Vue générale de RelMan

RelMan doit son nom à sa dépendance envers le Gestionnaire OSI. La figure 8.1 positionne RelMan dans l'environnement de gestion standard [ISO92a]. RelMan est utilisé par des applications de gestion au travers d'interfaces qu'il définit dans certains de ces composants. Il offre ainsi aux applications la possibilité d'exploiter les relations spécifiées à l'aide du GRM dans le modèle de l'information. De plus, comme il est peu probable que lors de la conception d'un modèle, toutes les relations utiles pour toutes les applications soient spécifiées, RelMan permet la prise en compte dynamique de relations nouvelles, évitant d'une part le codage à l'avance de ces relations dans une application et permettant d'autre part d'enrichir sa base de relations et de les partager entre toutes les applications. Les trois composants de RelMan sont les suivants:

RMIB [49] : c'est une base de données qui contient une représentation des relations gérées. Ce sont des instances issues de classes et de représentations de relations. Elles signifient la présence d'objets gérés dans les MIBs ayant des relations entre eux suivant les spécifications GRM et GRM+. Une relation gérée contient une image partielle des objets gérés qui y participent. Les spécifications GRM et GRM+ peuvent être conservées et évaluées dans le `Repository` de la RMIB. La RMIB est accessible uniquement par les deux composants qui suivent.

R-ActiveManager : il propose aux applications de gestion des interfaces pour demander l'exécution d'opérations de relation. Une autre fonctionnalité de plus haut niveau d'abstraction permet de manipuler des ensembles de relations gérées. Le R-ActiveManager est en contact avec le Gestionnaire OSI pour lui communiquer les primitives CMIS à envoyer (ou à recevoir) traduites des opérations de relation.

48. **Rel**ationship **Man**ager
49. **R**elationship **M**anagement **I**nformation **B**ase

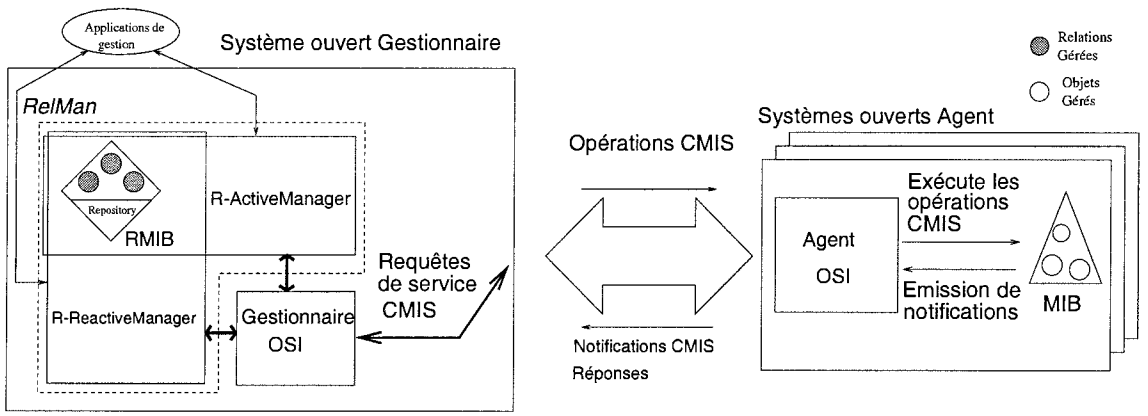


FIG. 8.1 – *RelMan et l'environnement de gestion OSI*

R-ReactiveManager : il contient les fonctionnalités nécessaires pour assurer le maintien d'un état cohérent de la RMIB. L'état d'une relation gérée est dépendant d'informations extraites des MIBs, donc sujettes à des modifications indépendantes du contexte de RelMan. De plus, le R-ReactiveManager offre une interface aux applications de gestion leur permettant de découvrir de nouvelles relations gérées à partir de spécifications GRM et GRM+. L'interface avec le Gestionnaire OSI est utilisée à des fins de scrutation des MIBs.

Il est à noter que les spécifications GRM+ sont évaluées uniquement à la demande des R-Managers (R-Managers désigne les composants R-ActiveManager et R-ReactiveManager). Le R-ActiveManager doit évaluer les conditions spécifiées autour de l'exécution des opérations de relation. Le R-ReactiveManager, en surveillant la cohérence des relations gérées, doit évaluer les conditions des invariants de classe et de représentation.

8.1.1 La base d'information de gestion des relations

La première fonctionnalité permettant la gestion des relations gérées est la possibilité de stocker et de retrouver des informations qui représentent une relation gérée. Cette possibilité est l'équivalent de la MIB des objets gérés, aussi nous l'avons nommée RMIB. La figure 8.2 représente un schéma Entité/Association traduisant la structure conceptuelle des RMIBs. une occurence de l'entité `RMIB` est associée avec un ensemble d'occurences de l'entité `Managed_Relationships`. Les R-Managers qui se partagent la même `RMIB` ne pourront agir que sur cet ensemble de relations gérées.

Les entités du schéma constituent un méta-modèle pour les relations gérées. Toute relation gérée, quelque soit sa classe et sa représentation, peut être décrite par ce modèle. Les occurences de l'entité `Managed_Relationships` contiennent les références du formulaire GRM de classe (`class`) et de représentation (`mapping`) de la relation gérée modélisée. De même, les formulaires de comportement en GRM+ (`classBehaviourList` et `mappingBehaviourList`) y sont référencés. Une occurence de `Managed_Relationships` est associée avec un certain nombre d'occurences de l'entité `Roles`. Ce nombre correspond au nombre de rôles de la classe de relation (comprenant les super classes). Si la représentation de la classe dans une occurence de `Managed_Relationships` utilise un objet de relation, celle ci est associée avec une occurence de l'entité `Relationship_Objects` (vue plus loin) qui modélise un objet géré de relation. La présence d'objet gérés dans un rôle est modélisée par l'association entre les entités `Roles` et `Participants`. Les occurences de cette dernière modélisent des objets gérés existant dans les MIBs. Une occurence de `Roles` est associée avec des occurences de `Par-`

ticipants dans la limite des cardinalités qui y sont définies. La présence redondante des cardinalités (ou des classes d'objets gérés compatibles) permet aux occurences de Roles d'être autonomes pour accepter ou refuser l'ajoût ou le retrait d'associations avec des occurences de Participants. L'entité Participants modélise, comme l'entité Relationship_Objects, un objet géré dans une MIB. Elles contiennent la valeur de l'identificateur de l'objet géré (le *Distinguish Name*).

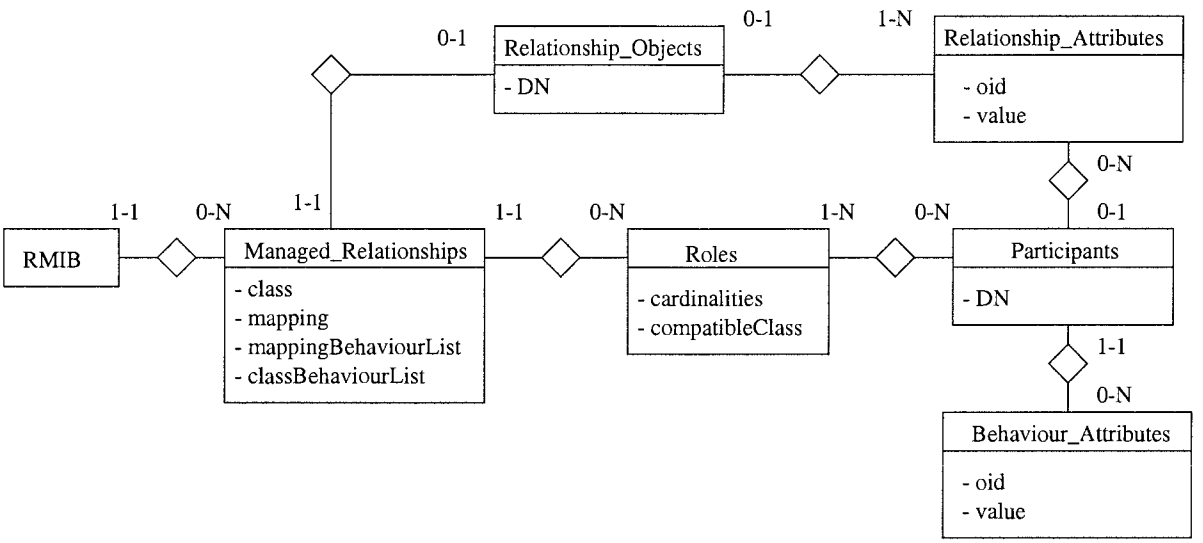


FIG. 8.2 – *Schéma Entité Association de la RMIB*

L'entité Relationship_Attributes modélise les attributs de relation contenus dans des objets gérés. Ces attributs sont eux même modélisés par les occurences ou Participants et de Relationship_Objects. Dans ce dernier cas, une occurence doit nécessairement être associée avec au moins une occurence de Relationship_Attributes. En revanche, si la représentation n'utilise pas d'attribut de relation, les occurences de Participants ne sont pas associées avec des occurences de Relationship_Attributes. Dans tous les cas, une occurence de Relationship_Attributes est associée exclusivement avec un Participant ou un tt Relationship_Object.

L'entité Behaviour_Attributes modélise des attributs d'objets gérés. Ces attributs sont référencés dans des formulaires GRM+ des occurences de Managed_Relationships associées, par l'intermédiaire de l'entité Participant, aux occurences Behaviour_Attributes. Ces attributs sont nécessaires pour évaluer les conditions dans les comportements GRM+.

Une particularité de la structure de la RMIB est qu'un objet géré participant à plusieurs relations gérées est modélisé par une unique occurence de Participants. Les occurences des entités Relationship_Attributes et Behaviour_Attributes associées à l' occurence de Participants correspondent à l'union des attributs GDMO nécessaires pour chaque occurence de Managed_Relationships à laquelle l'occurence de Participants est associée.

Les entités de Relationship_Attributes et de Behaviour_Attributes contiennent l'identificateur (oid) du formulaire GDMO d'attribut et sa valeur courante (value). De ce fait, plusieurs occurences de ces entités peuvent être identiques, mais comme ces attributs appartiennent à un unique objet géré, elles seront distinctes. Toutefois, il peut apparaître un cas particulier, où un objet géré est à la fois un objet de relation pour une relation gérée et un objet participant à une autre relation. Une occurence de Participants et une occurence de Relationship_Objects modélisent alors un même objet géré, risquant en théorie de générer deux occurences distinctes pour modéliser un même attribut. Toutefois, en pratique, si un objet géré sert à représenter une relation, il devrait pour cela ré-

server certains de ses attributs de relation. Si par ailleurs ce même objet participe dans un rôle d'une autre relation gérée, il ne devrait pas permettre une représentation utilisant un des ses attributs réservés au préalable.

La figure 8.3 montre une instance d'une RMIB. Les occurences d'objets modélisent l'instanciation des classes de relations précédentes sur la configuration de la figure 4.10 du chapitre 3. On retrouve par niveau : les relations gérées, leurs rôles, les participants, les attributs de comportement et les attributs de relation.

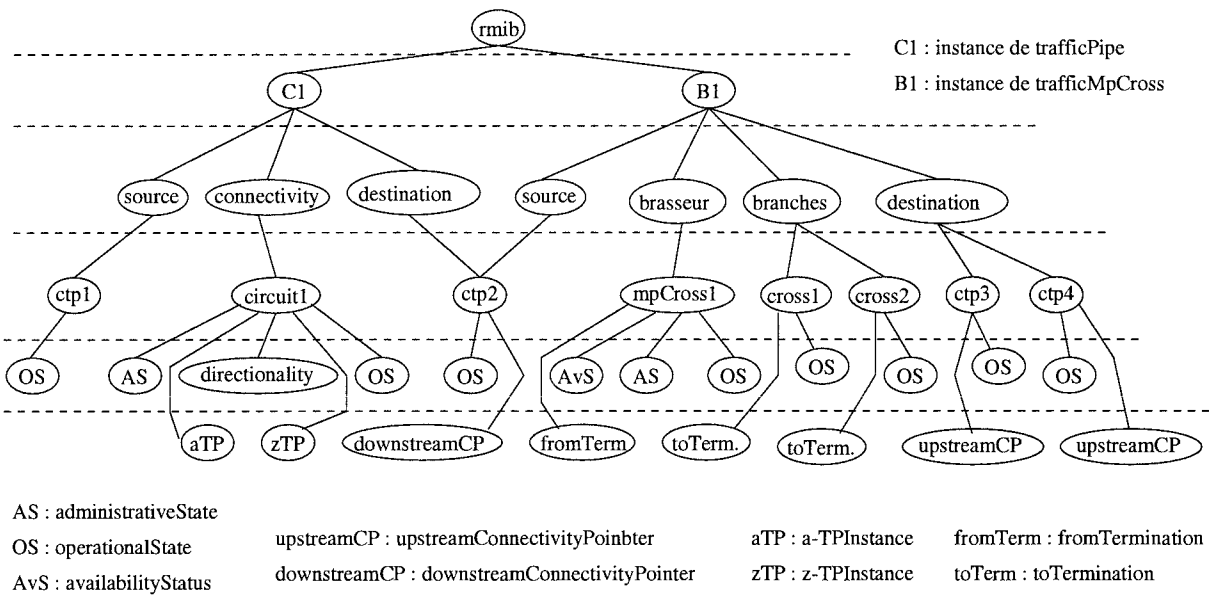


FIG. 8.3 – *Les occurences de la RMIB de l'exemple*

8.2 Le R-ActiveManager

Un des buts de l'intégration du GRM et du GRM+ dans la gestion standard est d'offrir aux applications de gestion la possibilité d'utiliser directement les relations gérées. Les applications que nous considerons dans ce chapitre sont des traitements conçus à partir de spécifications de modèles de gestion. Lorsque de telles applications manipulent les objets gérés, elles en déléguent généralement l'accès à des Gestionnaires OSI. Avec RelMan, les applications délègueront les accès aux relations au R-ActiveManager dont la vue détaillée, extraite de la figure 8.1 est donnée à la figure 8.4. Les applications de gestion ont accès à trois interfaces; l'interface RQ (*Relationship Query*) qui permet de manipuler des ensembles de relations gérées (des instances), l'interface RS (*Relationship Service*) donne la possiblité de faire des requêtes d'opérations sur les relations gérées et l'interface NS (*Notify Subscriber*) qui permet aux applications de s'abonner à des notifications de relation. Ces interfaces acceptent des appels de primitives et les transmettent à des composants du R-ActiveManager qui sont en contact avec la RMIB. Les composants sont le `Query` qui traite les appels de primitives de l'interface RQ, le `Mapper` qui traduit d'une part les opérations de relation en opérations système vers l'interface CMIS et d'autre part reçoit les réponses ainsi que les primitives CMIS de type `M-EVENT-REPORT` suite à des notifications d'objets gérés. Plusieurs réponses système peuvent être regroupées pour former une réponse d'opération de relation qui sera transmise par le `Mapper` à l'interface RS. Plusieurs notifications peuvent être regroupées pour former une notification de relation et le `Mapper` en informe alors le composant `Dispatcher` qui

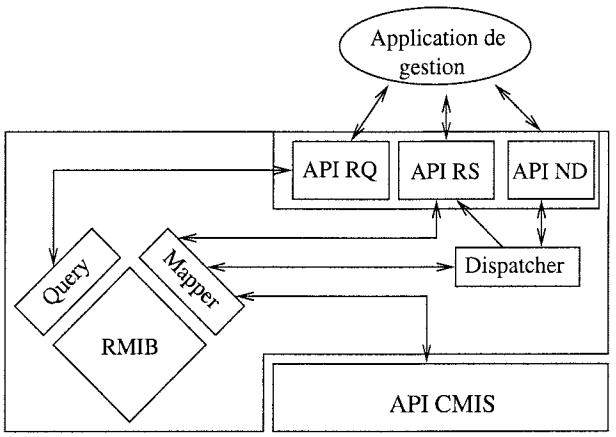


FIG. 8.4 – *Composants du R-ActiveManager Actif*

se charge de prévenir au travers de l'interface RS les applications abonnées au préalable à la notification de relation.

8.2.1 L'interface RQ

Nous avons vu dans la première partie les différences et les similitudes entre le modèle relationnel [Cod70] et le GRM. L'interface RQ (*Relationship Query*) propose des opérations ayant une sémantique comparable à celle des opérateurs de l'algèbre relationnel et qui manipulent des ensembles d'instances de relations gérées.

Pour ce faire, nous considérons une relation gérée comme un ensemble de participants et ses rôles comme des sous-ensembles non disjoints de ses participants. Les participants peuvent appartenir à plusieurs relations gérées.

Les opérateurs définis plus loin utilisent et construisent des ensembles de relations gérées. La caractéristique de ces ensembles est que toutes les relations gérées qu'ils contiennent ont chacune le même nombre de rôles. On appelle ER (Ensemble de Relations) ce type d'ensemble. On note $ER(n)$ pour un sous type de ER dont les relations ont n roles. Il est possible que deux relations gérées de la même classe relient les mêmes objets gérés, aussi un identificateur unique pour chaque relation gérée est nécessaire pour les distinguer dans un ensemble de type ER.

Les opérateurs peuvent contenir des conditions portant sur les relations gérées, leurs rôles et leurs participants. Nous définissons ci dessous l'égalité entre rôles et entre relations gérées:

- deux rôles distincts sont égaux si et seulement s'ils ont les mêmes participants. Par conséquent deux rôles peuvent être comparés si et seulement s'ils ont, d'une part des classes compatibles reliées par une relation d'héritage et d'autre part, une intersection non vide des intervalles de cardinalité des rôles. L'égalité entre rôles est symétrique;
- deux relations sont égales si chaque rôle d'une des relation est égal à exactement un rôle de l'autre et réciproquement.

Les opérateurs que nous avons définis sont inspirés de l'ensemble minimal des opérateurs nécessaires à l'algèbre relationnel (sauf le premier).

RQ-Family : $classe \rightarrow ER(n)$

Avec `classe` une classe de relation

Ensemble des instances de relations gérées de la classe de relation (n est le nombre de rôles de cette classe). Les relations gérées issues de sous classes font également partie de $ER(n)$, mais les

éventuels rôles supplémentaires sont exclus du $ER(n)$ résultat.

RQ-Project : $ER(n) \times \texttt{role} - \texttt{list} \rightarrow ER(l)$

Avec `role - list` une liste de l noms de rôles ($l \leq n$)

Ensemble des relations gérées dans lequel toutes les relations gérées ont uniquement les rôles référencés. Les participants se retrouvant dans aucun rôle référencé sont enlevés. On projette les relations gérées sur des rôles donnés.

RQ-Select : $ER(n) \times condition \rightarrow ER(n)$

Ensemble de relations gérées qui vérifient les prédicats énoncés dans *conditions*. On séléctionne des relations gérées.

RQ-Inter : $ER(n) \times ER(n) \rightarrow ER(n)$

Ensemble de relations gérées communes aux deux ensembles. C'est l'intersection de deux ER.

RQ-Union : $ER(n) \times ER(n) \rightarrow ER(n)$

Ensemble des relations gérées des deux ensembles. C'est l'union de deux ER.

RQ-Diff : $ER(n) \times ER(n) \rightarrow ER(n)$

Ensemble de relations gérées présentes dans le premier ensemble mais pas dans le deuxième. C'est la différence de deux ER.

RQ-Cart : $ER(n) \times ER(m) \rightarrow ER(n+m)$

Ensemble des relations gérées ou chacune est composée d'une relation du premier ensemble et d'une du second. Toutes les relations des deux paramètresER sont composées deux à deux. Si des rôles ont le même nom, ils sont renommés. C'est le produit cartésien de deux ER.

Ces opérateurs peuvent être combinés pour en construire de nouveaux. Comme exemple, une requête RQ sur la RMIB de la figure 8.3 est montrée figure 8.5. Elle vise à retrouver les points de terminaison qui recoivent le traffic de la connexion débutée par le point `ctp1`.

```
(1) RQ-Project(
(2)             RQ-Select(
(3)                       RQ-Cart(RQ-Family(connexionGenerique),
(4)                               RQ-Family(connexionGenerique)
(5)                       ),
(6)                       source1 = {ctp1} ET source2 = destinations1
(7)             ),
(8)             destination2
(9) )
```

FIG. 8.5 – *Une requête RQ*

Par le produit cartésien (lignes 3 et 4), on construit des relations à quatres rôles, renommés `source1`, `destinations1`, `source2`, `destinations2`. Dans cet ensemble, on sélectionne (ligne2) les relations gérées suivant la condition en ligne 6. Dans cette condition, on construit un rôle (`{ctp1}`) pour le comparer au rôle `source1` et on compare les rôles `source2` et `destinations1`. Enfin, on projette l'ensemble obtenu (un singleton en l'occurence) sur le rôle qui nous interesse ici car il contient les points de terminaison finaux de la connexion (i.e. `ctp3` et `ctp4`). On peut constater que la combinaison de ces opérateurs définit une jointure sur les rôles des relations gérées qui ressemble à la jointure dans l'algèbre relationnelle. Dans cette dernière, la jointure est définie de manière analogue en combinant les opérateurs de séléction, de produit cartésien et de projection.

Enfin, il faut noter que l'utilisation de l'interface RQ doit se faire avec une granularité des traitements qui soit supérieure à celle du maintien de la cohérence de la RMIB. L'interface RQ a la charge d'analyser les opérateurs RQ pour en vérifier la syntaxe, puis de les envoyer au composant `Query` qui va construire les réponses.

8.2.2 Le composant `Query`

Les opérateurs RQ reçus par le composant `Query` doivent être interprétés pour générer des requêtes sur la RMIB. Pour ce faire, il est nécessaire d'avoir une structure correspondant au type ER et à tous ses sous types. La structure définie en ASN.1 ci dessous est une possibilité.

```
ERType ::= SET OF RelG
RelG   ::= SET OF Role
Role   ::= SEQUENCE {nom          ObjectIdentifier,
                     participants SET OF DistinguishName}
```

Les types `ObjectIdentifier` et `DistinguishName` sont définis dans CMIP. Le type `Role` est défini par un identificateur de rôle et par la liste des identificateurs des objets gérés qui y participent. Il est nécessaire que les spécifications GRM donnent un identificateur à chaque rôle (ce qui est optionnel dans la norme) qui soit unique afin de construire des valeurs de `ERType` pour toutes les relations GRM. L'égalité entre deux valeurs de type `Role` est vérifiée si et seulement si ils ont les mêmes valeurs dans leur champ `participants`. Deux variables du type `ERType` sont égales si et seulement si chaque `Role` de l'une est égal à exactement un `Role` de l'autre et réciproquement.

Le seul opérateur qui s'adresse directement à la RMIB est l'opérateur RQ-Family. Il construit une variable de type `ERType` en faisant une sélection sur l'entité `Managed_Relationships` dont la condition de sélection est de référencer la classe donnée en paramètre, ou une sous classe. Dans les deux cas, on ne construit que les variables de type `Role` dont les noms sont ceux de la classe de base, les autres rôles des relations gérées sont ignorés. La construction de l'ensemble de participants se fait en suivant les associations entre les entités `Roles` et `Participants` de la RMIB.

8.2.3 L'interface RS

Les classes de relations gérées définissent des opérations qui constituent l'interface de gestion de ces relations. Le but de l'interface RS (*Relationship Service*) est de fournir aux applications de gestion utilisant RelMan la possibilité d'appeler ces opérations et d'en recevoir les réponses. Nous avons défini un ensemble de primitives qui peuvent être utilisées pour demander l'exécution de n'importe quelle opération d'une relation gérée de la RMIB. Les tableaux 8.1 donnent les paramètres de ces primitives.

Les appels de primitives RS sont transmis au composant `Mapper`, qui se charge de transmettre à l'interface RS le retour de ces primitives. Chaque primitive porte le nom de l'opération générique dont elle assure l'exécution. L'opération `M-CANCEL-QUERY` est une primitive que nous avons rajoutée dans le but d'annuler un appel antérieur de la primitive `M-QUERY`. La définition des paramètres des primitives est la suivante:

relationClass, relationMapping doivent référencer les formulaires GRM de classe et de représentation qui décrivent la nouvelle relation gérée à instancier.

operationLabel correspond au label de l'opération, s'il existe, spécifié dans la classe de relation.

relationId doit permettre de désigner une unique occurence de `Managed_Relationships` dans la RMIB. Nous n'avons pas défini de règle pour construire un tel identificateur et c'est à l'implantation de la RMIB d'en fournir une.

infoGrmP doit être du type ASN.1 défini dans les formulaires de comportement des représentations (cf. \$7.2.3 chap. 7). Il est obligatoire dans toutes les primitives, si la spécification GRM+ ne définit aucun type pour une opération, un paramètre vide devra tout de même être fourni à la primitive de l'interface RS invoquée.

destAppli contient un identificateur permettant de faire parvenir la notification de relation vers une application de gestion qui en a fait la demande (voir plus loin l'interface NS).

queryId permet d'identifier un appel de la primitive `M-QUERY`, afin de pouvoir l'annuler par un appel de la primitive `M-CANCEL-QUERY`.

Primitive M-ESTABLISH		
paramètres	appel	retour
relationClass	O	C
relationMapping	O	C
operationLabel	C	C
relationId	-	O
infoGrmP	O	O
Errors	-	C

Primitives M-TERMINATE, M-BIND,M-UNBIND M-QUERY, M-CANCEL-QUERY, M-NOTIFY M-USER-DEFINED		
paramètres	appel	retour[a]
destAppli[b]	O	-
operationLabel	C	C
relationId	O	C
infoGrmP	O	O
queryId	O[c]	O[d]
Errors	-	C

[a] pas de retour pour `M-NOTIFY`
[b] uniquement `M-NOTIFY`
[c] uniquement `M-CANCEL-QUERY`
[d] uniquement `M-QUERY`

O : présence Obligatoire du paramètre
C : présence Conditionnelle du paramètre

TAB. 8.1 – *Primitive et paramètre de l'interface RS*

Errors est retourné en cas de problèmes dans l'exécution des primitives. Les types d'erreurs que l'on peut rencontrer sont :

- `NoError`; la primitive s'est déroulée correctement
- `NoSuchRelationshipClass`; la classe de relation n'existe pas
- `NoSuchMapping`; la représentation n'existe pas
- `NoSuchMappingForRelationshipClass`; la représentation n'est pas une représentation de la classe
- `NoSuchOperation`; l'opération n'existe pas dans la classe
- `NoSuchOperationLabel`; le label de l'opération n'existe pas pour l'opération
- `NoSuchRelationshipInstance`; la relation gérée n'existe pas
- `TypeMismatch`; le type du paramètre GRM+ n'est pas celui du formulaire GRM+
- `PreConditionFailure`; la précondition de l'opération n'est pas validée
- `InvariantOperationFailure`; l'invariant de l'opération n'est pas validé
- `PostConditionFailure`; la postcondition de l'opération n'est pas validée
- `OperationInProgress`; la relation gérée est en cours d'utilisation
- `InvalidRelationship`; la relation gérée est dans un état invalide qui en empêche son utilisation

L'interface NS et le composant `Dispatcher`

L'interface NS (*Notify Subscriber*) est une interface permettant aux applications de gestion de s'abonner auprès du composant `Dispatcher` comme réceptrices de notifications de relations gérées. Cette interface est représentée par les primitives `M-SUBSCRIBE` et `M-UNSUBSCRIBE` dont les paramètres sont donnés dans la table 8.2. Le paramètre **notifyLabel** spécifie un label de `NOTIFY`, les autres paramètres sont semblables à ceux des primitives RS. Les interactions entre interface et composants peuvent être suivies sur la figure 8.6. Lorsqu'un appel de la primitive `M-SUBSCRIBE` est transmis au composant `Dispatcher`, celui ci doit toujours vérifier si une telle requête, sur la même relation gérée et la même notification, n'a pas déjà été demandée par des applications de gestion. Dans ce cas le composant `Dispatcher` rajoute le paramètre **destAppli** dans la liste des identificateurs d'applications déjà enregistrées. Dans le cas de la première application, la requête est transmise au composant `Mapper` (vu plus loin) et la liste d'identificateurs d'applications de gestion abonnées est créée et initialisée dans le composant `Dispatcher`. Si le composant `Mapper` constate une erreur dans les paramètres de la

Primitives M-SUBSCRIBE, M-UNSUBSCRIBE		
paramètres	appel	retour
destAppli	O	C
notifyLabel	C	C
relationId	O	C
Errors	-	C

O : présence Obligatoire du paramètre

C : présence Conditionnelle du paramètre

TAB. 8.2 – *Primitives de l'interface DN*

primitive, il les retourne avec une erreur dans le paramètre Errors. Une relation gérée réunit les condi-

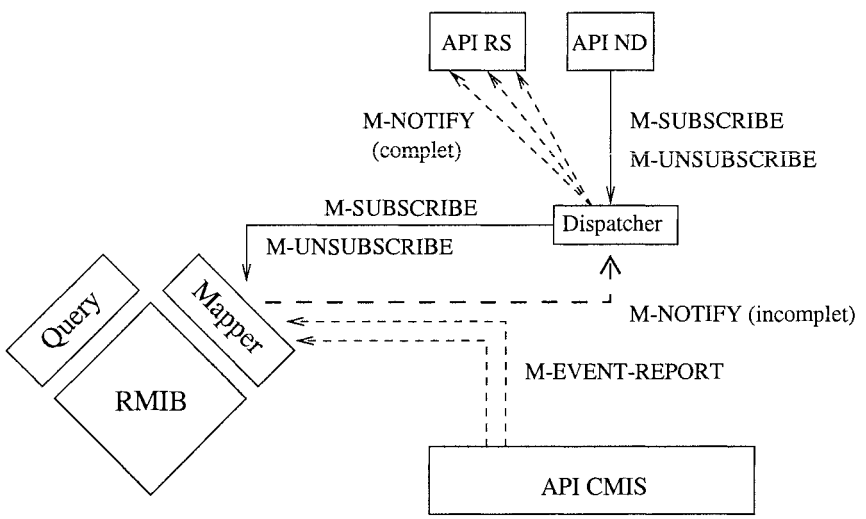


FIG. 8.6 – *Détail des interactions avec le composant* `Dispatcher`

tions pour émettre une notification de relation lorsque les primitives de service `M-EVENT-REPORT` qui lui correspondent sont détéctées par le composant `Mapper`. C'est ce dernier qui envoie alors un appel de la primitive `M-NOTIFY` au composant `Dispatcher`. Celui ci le dupplique, pour chaque application abonnée, en completant le paramètre **destAppli** qui leur correspond. Ces primitives sont ensuite envoyées vers l'interface RS et donc vers les applications.

La primitive `M-UNSUBSCRIBE` est utilisée par une application désirant ne plus recevoir des appels de la primitive `NOTIFY`. Le composant `Dispatcher` doit alors vérifier que l'application est bien enregistrée pour les paramètres donnés dans l'appel de `M-UNSUBSCRIBE`. Si tel est le cas, l'identificateur de l'application est retiré de la liste. Dans le cas contraire, la primitive est retournée avec une erreur. S'il s'agit de la dernière application de gestion enregistrée, la primitive doit être passée au composant `Mapper` et la liste est supprimée.

8.2.4 Le composant `Mapper`

La fonctionnalité du composant `Mapper` est d'exécuter les opérations de relation sur des relations gérées modélisées dans la RMIB. La figure 8.7 détaille la composition du composant `Mapper`. Les appels

de primitives de l'interface RS et du composant `Dispatcher` sont réceptionnés par le composant `ControlPerformers`. Sa fonction est de générer dynamiquement un nouveau composant dans le `Mapper` et de lui transmettre les paramètres de la primitive reçue. Ces composants dynamiques, nommés `Performers`, dialoguent avec la RMIB pour y récupérer ou modifier des informations sur la relation gérée qui doit exécuter l'opération invoquée. Une telle opération implique soit la génération d'opérations systèmes, soit l'attente de notifications d'objets gérés. Les composants `Performers` utilisent pour ce faire, les primitives d'une interface CMIS. Les réponses de cette interface sont ensuite regroupées pour former les réponses aux opérations de relation. Ces réponses sont transmises aux composants qui les ont initiées (l'interface RS ou le composant `Dispatcher`), puis sont généralement détruits. Pour les

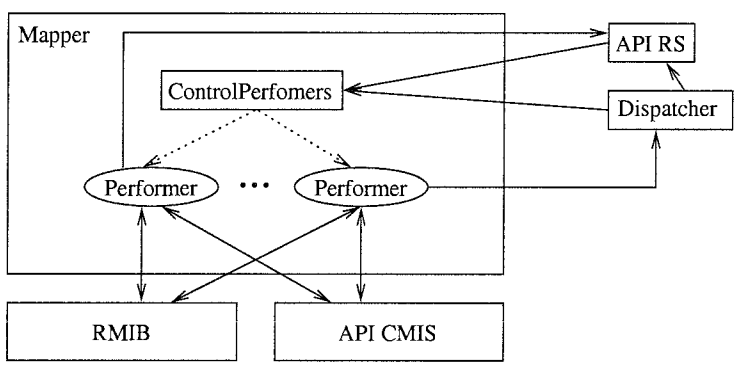


FIG. 8.7 – *Composant* `Mapper`

primivites `M-CANCEL-QUERY` et `M-CANCEL-SUBSCRIBE`, le composant `ControlPerformers` ne génère pas un nouveau `Performer`, mais cherche à en détruire un.

Les tâches successives des composants `Performer` sont les suivantes:

1. Vérification de la sémantique des paramètres (relation existante, label, ...).
2. Vérification de l'état de la relation gérée.
3. Vérification des préconditions de l'opération de relation et des invariants de l'opération.
4. Pour chaque opération système traduisant l'opération de relation, constitution les paramètres des appels de primitives de l'interface CMIS et les émission vers celle-ci.
5. Réception depuis l'interface CMIS des réponses des opérations systèmes émises.
6. Vérification des postconditions et des invariants de l'opération.
7. Constitution et envoi du retour de la primitive de service de relation appellée.

Concernant le point 2, le composant `Performer` ne doit pas interdire l'action de plusieurs autres composants `Performer` sur la même relation gérée. La liste des opérations de relation pouvant être lancées simultanément (plusieurs fois la même opération ou plusieurs opérations différentes) est une information qui permet aux composants `Performer` de décider s'ils peuvent poursuivre leur tâche. Des variables dans les occurences de l'entité `Managed_Relationships`, indiquant la liste des opérations de relation que l'on peut appeler en parallèle et la liste des opérations en cours de traitement, sont alors nécessaires.

Lorsque les conditions aux points 1,2,3 et 6 ne sont pas vérifiées, la succession des traitements est interrompue. Dans le point 6, il faut être de plus en mesure d'annuler les modifications sur la relation et les objets gérés. Si c'est impossible, la relation gérée doit être mise dans un état invalide qui interdit son utilisation par l'interface RS et le composant `Dispatcher`.

Il n'est pas interdit à un composant `Performer` de faire plusieurs appels simultanés à l'interface CMIS. Par exemple, une opération de type `QUERY` peut être traduite en deux opérations `GET` sur des rôles différents. Une liste des opérations systèmes pouvant être lancées en parallèle permettrait une telle optimisation.

8.3 Le R-ReactiveManager

Le R-ReactiveManager maintient l'intégrité de la RMIB et y permet la génération automatique de relations gérées. La figure 8.8 montre ses composants de base. L'interface RI (*Relationship Inference*) est utilisable par les applications de gestion pour découvrir des relations gérées virtuellement présentes dans les MIBs. Le composant `Infers` réalise cette recherche en utilisant l'interface CMIS . Le composant `Coherency` vise à maintenir la cohérence de la RMIB avec les objets gérés et à assurer que les relations conservent leur sémantique.

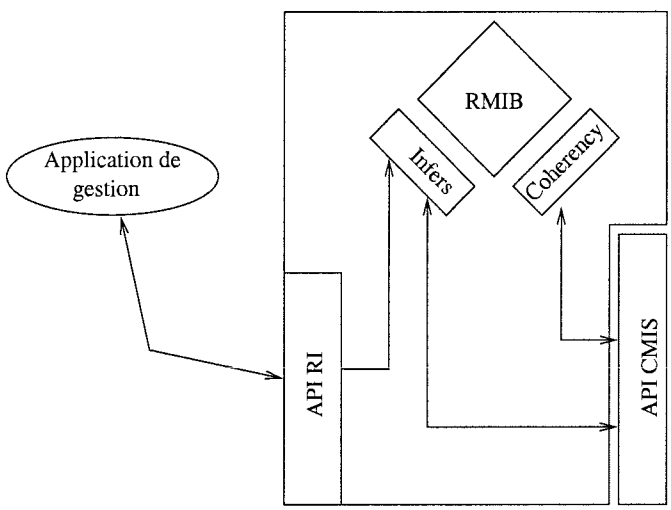


FIG. 8.8 – *Composants du R-ReactiveManager*

8.3.1 Le composant `Coherency`

La présence des participants ou des objets de relation dans une RMIB pose le problème du maintien de leur cohérence avec les objets gérés, localisés dans les MIBs, qu'ils représentent. L'état des objets gérés entraîne des conséquences sur l'état des relations auxquelles ils participent en se propageant sur celui des participants. Par exemple, dans la figure 8.3 une modification de l'état de l'objet `ctp2` peut influencer celui des relations `C1` et `B1`. Les types de modifications d'un objet géré qui interessent le composant `Coherency` sont sa suppression ou des changements de valeur dans ses attributs.

La figure 8.9 détaille les composants qui forment le composant `Coherency`. Toute nouvelle relation gérée de la RMIB est signalée au composant `ControlCoherency`. Celui-ci créé alors un composant `RelationCoherency` qui se charge de l'intégrité de la nouvelle relation gérée. Ce composant créé alors autant de composants `ParticipantCoherency` qu'il y a d'occurences de Participant associés à la relation gérée. Toutefois, si un des objets gérés est déjà dans une relation de la RMIB, il doit déjà exister un composant `ParticipantCoherency` qui se charge de sa cohérence. Le nouveau composant `RelationCoherency` se rajoute alors aux autres, en signalant quels attributs de l'objet géré sous-jacent le concerne. Le composant `ParticipantCoherency` fait l'union de l'ensemble des attributs qui vient de lui être signalé avec son propre ensemble d'attributs à surveiller. La surveillance des attributs est sous la responsabilité des composants `ParticipantCoherency` qui disposent pour cela de l'interface CMIS. Ils peuvent l'utiliser pour scruter (*polling*) périodiquement les objets gérés, en adaptant la période de *polling* suivant leur fréquence de variation d'état [DB97] [DvB97]. La surveillance peut aussi se faire entièrement par notifications. Ceci dans la mesure où les objets gérés peuvent émettre des notifications

indiquant des changements de valeur des attributs référencés dans les formulaires GRM et GRM+ et des notifications de suppression d'objet.

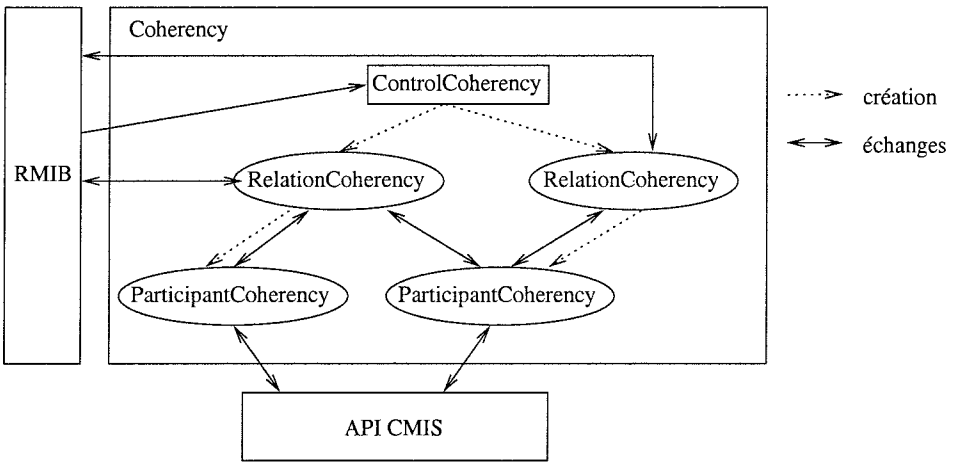


FIG. 8.9 – *Composant* `Coherency`

Lorsqu'une modification d'attribut est repérée par un composant `ParticipantCoherency`, ce dernier la communique aux composants `RelationCoherency` avec lesquels il est relié. Chacun de ces composants reçoit la référence de l'attribut dans la RMIB et la modification perçue (l'ancienne et la nouvelle valeur, l'opération système). Deux cas de figures sont alors possibles:

- c'est un attribut de comportement (occurence de `Behaviour_Attributes`); les invariants de classe et de représentation en GRM+ doivent être évalués avec la nouvelle valeur. Si l'invariant est invalidé, la relation gérée doit être invalidée;
- c'est un attribut de relation qui est utilisé pour représenter la relation gérée:
  - si la modification consiste à rajouter ou remplacer une valeur, il faut vérifier que celle-ci désigne un objet géré qui soit d'une classe correcte pour le rôle que représente l'attribut. Dans ce cas, cette modification correspond à un `BIND`, forcé par les ressources ou par un autre Gestionnaire OSI. Le composant `RelationshipCoherency` recherche alors si une opération `BIND` est définie pour le rôle et si elle est traduite avec la même opération (`ADD` ou `REPLACE`) que celle reçue du composant `ParticipantCoherency`. Si une telle opération est trouvée, les postconditions doivent en être évaluées;
  - si la modification consiste à enlever une valeur, ce sont les postconditions des opérations `UNBIND` ou `TERMINATE` qu'il faut vérifier (celles qui se traduisent par un `REMOVE`).

  Dans les deux cas, les intervalles de cardinalité sont également à surveiller, de même que l'évaluation de toutes les conditions qui font intervenir la cardinalité du rôle. Si aucune opération n'est trouvée, si les cardinalités sont violées ou qu'une condition est invalide, la relation est invalidée.

Si une notification de suppression d'objet est reçue par un composant `ParticipantCoherency`, le même raisonnement qu'avec l'opération `UNBIND` ou `TERMINATE` s'applique, mais en cherchant une traduction avec un `DELETE`.

8.3.2 L'interface RI et le composant Infer

Une plateforme de gestion implantant RelMan n'est pas obligée d'être lancée au démarrage du système de gestion. Les objets gérés peuvent avoir été mis en relation sans son intervention, par des Ges-

tionnaires OSI. Le but de l'interface RI (*Relationship Inference*) et du composant Infer est de permettre une initialisation de la RMIB à partir de la configuration courante des objets gérés.

L'interface RI

Une application voulant initialiser RelMan utilise les primitives de l'interface RI données dans la table 8.3. Les paramètres **relationClass** et **relationMapping** sont des références à des formulaires GRM de la classe et de la représentation des relations gérées recherchées. La réponse consiste principalement en une liste d'identificateurs de relations dans la RMIB (le paramètre **relationIdList**). Ces relations viennent d'y être enregistrées suite à l'appel de la primitive `M-INFERS`. La primitive `M-CANCEL-INFERS` sert

Primitives M-INFERS, M-CANCEL-INFERS		
paramètres	appel	retour
requestId	O[a]	O[b]
relationClass	C	C
relationMapping	O	C
relationIdList	-	C
Errors	-	C

[a] uniquement `M-CANCEL-INFERS`
[b] uniquement `M-INFERS`

O : présence Obligatoire du paramètre
C : présence Conditionnelle du paramètre

TAB. 8.3 – *Primitives de l'interface RI*

à arrêter l'exécution d'une primitive `M-INFERS` précédente, identifiée par le paramètre **requestId**. La réponse à cette primitive contient les relations gérées découvertes et enregistrées dans la RMIB à l'instant de réception de l'appel `M-CANCEL-INFERS`.

Le composant `Infer`

Le composant `Infer` contrôle la découverte des relations gérées. Celle ci peut s'avérer très coûteuse puisqu'il faut recueillir et analyser des informations sur toute l'étendue du domaine de gestion de RelMan (i.e. les objets gérés accessibles par le Gestionnaire OSI de RelMan). A partir d'une spécification de représentation, la démarche doit être la suivante:

- pour chaque rôle, rechercher dans le domaine de RelMan, tous les objets gérés des classes rattachées au rôle;
- pour chaque objet géré d'un rôle, chercher ceux qui lui sont reliés dans les autres rôles, en accord avec la représentation (attributs, corrélations de noms, ...) et construire ainsi un graphe reliant les objets;
- une relation gérée est découverte lorsque le graphe ne peut plus être étendu par de nouveaux objets, les contraintes de cardinalité et les invariants GRM+ sont alors à vérifier pour intégrer la relation gérée dans la RMIB.

Une optimisation de ce processus peut être faite en analysant la spécification de représentation de la classe de relation recherchée. L'idée est de n'avoir à rechercher que certaines classes d'objets gérés, à partir des instances desquelles on puisse retrouver d'autres objets qui forment une instance de relation gérée. Une représentation qui utilise un objet de relation donne directement ces objets (du moins leur

classe) puisqu'un objet de relation contient des attributs de relation qui réfèrencent tous les participants à la relation. Dans les autres cas de représentation, les rôles sont associés deux à deux dans les cas suivants:

- une corrélation de noms (`NAME BINDING`) associe un rôle supérieur avec un rôle subordonné. La recherche de tous les participants subordonnés peut se faire par une lecture (service `M-GET`) où l'objet de base est le participant supérieur, la profondeur de champ (*Scope*) n'est que de niveau un et un filtre séléctionne les objets gérés qui sont de la classe subordonnée;
- un attribut de relation dans un objet géré participant à un rôle, référence des objets participants dans un autre rôle. La recherche des participants se fait en deux étapes : (1) lecture de la valeur de l'attribut et (2) lecture des participants référencés.

Le problème consistant à trouver quelles classes d'objets sont celles qui permettent de retrouver les autres est alors ramené à un problème de recherche de noeuds dans un graphe orienté où les noeuds sont les rôles et où les arrêtes modélisent la possibilité de découvrir les participants d'un rôle à partir d'un autre rôle. Le graphe ainsi obtenu est un 1-graphe orienté dont l'ordre est égal au nombre de rôle [Ber85]. La figure 8.10 montre le graphe de la représentation `connectionTrafficMpCross` (fig. 4.16 chap. 3). Le rôle `mpCross` est celui qui permet de retrouver la totalité de la relation. Dans un cas général, il faut d'une part décomposer le graphe initial en sous-graphes dans lesquels chaque noeud est la destination d'au moins un autre noeud dans le sous-graphe, d'autre part pour chaque sous-graphe, il faut déterminer le ou les noeuds qui permettent de parcourir tous les autres noeuds du sous-graphe.

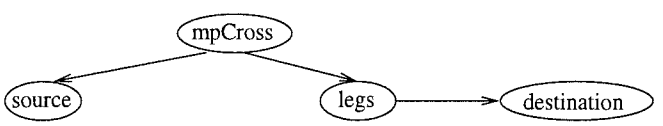


FIG. 8.10 – *Graphe de l'accessibilité des rôles*

8.4 Conclusions

Nous avons décrit une architecture de composants divers permettant la gestion autonome, ainsi que la gestion via des applications, des objets gérés au travers des spécifications de relations gérées.

RelMan est conçu comme une extension du Gestionnaire OSI. Les raisons de ce choix sont d'une part que des relations gérées peuvent lier des objets dans différentes MIBs et d'autre part qu'il n'existe pas de service de communication pour demander l'exécution des opérations de relations. Ce choix a en outre l'avantage de pouvoir proposer RelMan comme un composant des applications de gestion, au même niveau qu'un Gestionnaire OSI. Ces applications peuvent utiliser directement les deux Gestionnaires (OSI et RelMan).

Les concepts de RelMan : une base de données des relations gérées et des services pour les manipuler ouvrent une voie dans le développement des applications de gestion. Toutes celles qui ont besoin de parcourir le graphe d'objets gérés peuvent le faire avec les relations gérées, sans faire intervenir les détails de la modélisation logique des relations (i.e. les attributs de relation ou les corrélations de noms). Les relations gérées cachent aux applications ces détails et leur conception s'en trouve allégée. Les relations gérées sont également des interfaces qui contrôlent les manipulations des objets qu'elles relient. Les opérations de relations cachent aux applications les opérations sur les objets. Les risques d'erreurs de manipulation sur ces derniers en sont donc diminués. RelMan permet une gestion automatisée des relations gérées. Celles-ci influencent généralement le comportement des objets reliés et les effets qu'ont ces influences sur ces derniers peuvent être vérifiés.

Les concepts de RelMan sont indépendants des implantations qui peuvent en être faites. La RMIB peut être une base de donnée distribuée, où les relations gérées sont enregistrées dans la RMIB la plus proche des objets gérés qu'elle relie. La RMIB pourrait également être implantée par des objets de gestion GDMO crées dans ce but. Les opérations de relations seraient alors invoquées par l'intermédiaire d'une action sur les objets modélisant les relations. Le maintien de la cohérence et la découverte de relations gérées pourraient être confiés à des agents mobiles parcourant le réseaux pour receuillir les états des objets gérés dans une relation, ou pour découvrir les relations gérées d'une classe et d'une représentation donnée dont la définition serait dans l'agent mobile.

Troisième partie

Implantations

Cette partie présente les implantations que nous avons réalisées dans le domaine de la gestion OSI. Ces réalisations sont directement ou indirectement liées aux contributions de cette thèse. Un interface de programmation qui émule le service de gestion de l'OSI à été réalisée pour servir de support à une implantation du concept de gestionnaire de relation RelMan.

Chapitre 9

Une interface CMIS en Java et une implantation RMI

Ce chapitre présente l'interface de programmation (API : *Application Programming Interface*) `cmisOverJava`. Cette interface offre à des applications écrites en Java une émulation du service CMIS en associant un mécanisme d'événements à la simulation de ce service. L'API `cmisOverJava` a été développée pour fournir une interface cliente du service CMIS aux composants de type R-Manager de RelMan. Nous avons choisi de séparer `cmisOverJava` et RelMan afin de disposer d'une interface générique, pouvant être utilisée par d'autres applications de gestion que RelMan. Pour être opérationnelle, l'API `cmisOverJava` doit être implantée au dessus d'un service de communication. Différents services peuvent être utilisés. Dans ce chapitre, nous présentons une implantation de `cmisOverJava` au dessus du système des RMI (*Remote Method Invocation*).

9.1 L'API `cmisOverJava`

9.1.1 Environnement de l'API

La figure 9.1 positionne `cmisOverJava` et son implantation entre RelMan et des agents OSI. RelMan utilise `cmisOverJava` pour faire des requêtes CMIS et pour en recevoir les confirmations. Entre les agents et l'implantation de `cmisOverJava`, il doit exister un service de communication. Ce dernier peut être une implantation du service CMIS, donnant une indépendance totale entre l'implantation de `cmisOverJava` et les implantations d'agents. Un service de communication simulant CMIS peut aussi être développé dans l'implantation de `cmisOverJava` et dans celle des agents, dans ce cas les deux implantations sont indissociables, c'est le cas de l'implantation décrite dans la deuxième section de ce chapitre.

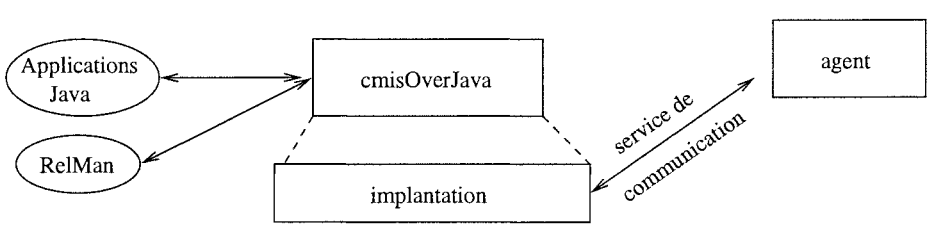


FIG. 9.1 – *Contexte d'utilisation de* `cmisOverJava`

9.1.2 Vue générale de l'API

Deux fonctionnalités composent `cmisOverJava`; la simulation du service CMIS et un mécanisme permettant d'associer des événements à cette simulation. Comme le montre la figure 9.2, seule la partie concernant la simulation nécéssite une implantation qui rend opérationnelle la communication avec les agents. Le mécanisme des événements est entièrement implanté dans l'API. Une partie de ce mécanisme peut être adaptée aux besoins d'une application comme RelMan par héritage.

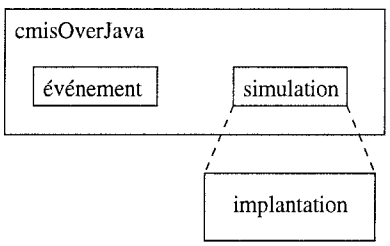


FIG. 9.2 – *Vue interne de* `cmisOverJava`

9.1.3 L'API de simulation

La figure 9.3 détaille la composition de l'API de simulation. Elle est constituée de deux modules, des *packages* dans la terminologie Java. Ces modules contiennent des classes d'objets et des interfaces Java. Une interface Java est une définition de prototypes de méthodes, il n'y a aucun corps de méthode, ni d'attribut dans une interface. Les définitions d'interfaces peuvent utiliser un mécanisme d'héritage multiple. Pour être utilisable, une interface Java doit être implantée par une classe d'objet Java. Les instances d'une telle classe pourront être vues comme des instances de l'interface.

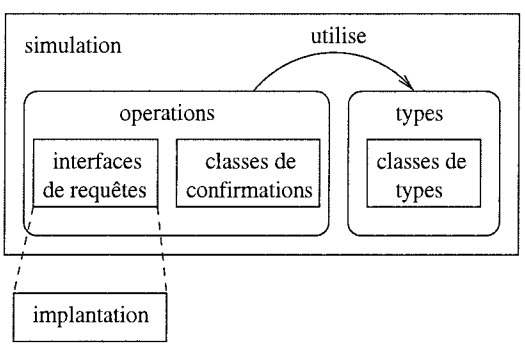


FIG. 9.3 – *Vue interne de la simulation*

Le module `operations` est constitué d'interfaces modélisant les primitives d'appels du service CMIS, la table 9.1 explicite ces interfaces. Pour chaque primitive, une classe d'objet modélise la confirmation de cette primitive. On note que tous les services de CMIS sont définis dans l'API. Elle est prévue pour être utilisée aussi bien dans le rôle gestionnaire que dans celui de l'agent. La figure 9.4 montre la position de `cmisOverJava` par rapport au utilisateurs du service CMIS (des MIS-User[50] au sens de l'OSI [ISO92a]). L'API d'écrit un fournisseur de service CMIS. L'implantation de l'API `cmisOver-`

50. Management Information Service-User

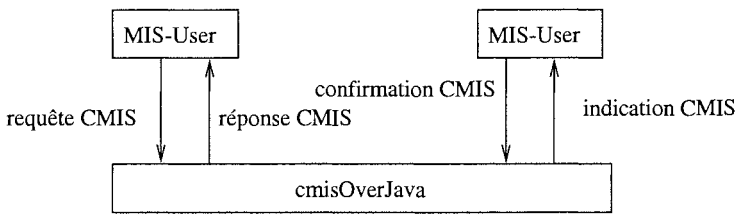


FIG. 9.4 – cmisOverJava *fournisseur de service CMIS*

Java se fait en implantant les interfaces de requêtes. Nous verrons plus loin la raison de ce choix par rapport à la définition de classes d'objets pour les confirmations.

primitive CMIS	Requête : interfaces cmisOverJava	Confirmation : classes cmisOverJava
M-CREATE	CreateCall	CreateReply
M-DELETE	DeleteCall	DeleteReply
M-GET	GetCall	GetReply
M-CANCEL-GET	CancelGetCall	CancelGetReply
M-SET	SetCall	SetReply
M-ACTION	ActionCall	ActionReply
M-EVENT-REPORT	EventReportCall	EventReportReply

TAB. 9.1 – *Primitives CMIS et* cmisOverJava

Le module types est composé de classes d'objets qui modélisent les différents types de paramètres des primitives CMIS. La table 9.2 explicite quelques unes des classes utilisées dans la primitive M-GET. Il faut noter que dans le cas des interfaces de requêtes, ces classes sont utilisées comme paramètres des méthodes des interfaces, alors que pour les classes de confirmation, les classes de types sont aussi des attributs de ces classes.

Module des opérations

Les interfaces et les classes définissent toutes les méthodes nécessaires à la mise à jour des paramètres de service et à leur lecture. Certaines caractéristiques communes nous ont amené à concevoir une hiérarchie d'héritage pour les interfaces et les confirmations. La figure 9.5 montre celle des interfaces de requêtes. L'interface CMISCall est la super-interface de toutes les interfaces de service. Les caractéristiques communes qui donnent lieu à l'héritage sont liées à la présence de mêmes types de paramètres entre les primitives. Pour les interfaces modélisant les primitives d'appels, cela se traduit par des méthodes de mise à jour et d'accès à des paramètres. Ainsi par exemple, une méthode setInvokeId qui initialise le paramètre de service InvokeId est dans la définition de l'interface CMISCall car ce paramètre est présent dans toutes les primitives d'appels de CMIS. L'interface FilteredAndScopedCall regroupe les requêtes qui utilisent la portée et le filtrage comme paramètre de service.

La figure 9.6 montre la hiérarchie entre les classes de confirmation. La classe CMISLinkedReply est une classe abstraite qui regroupe les caractéristiques propres à des confirmations pouvant être multiples. Les réponses multiples étant possibles lorsque la requête utilise le mécanisme de portée et de filtrage, les classes de confirmation correspondent aux interfaces de requêtes qui héritent de l'interface FilteredAndScopedCall. La classe CMISReply est une classe abstraite. Les caractéristiques

Nom du paramètre	Requête : classes `cmisOverJava`	Confirmation : classes `cmisOverJava`
Invoke identifier	InvokeId	InvokeId
Linked identifier	InvokeId	InvokeId
Base object class	ObjectClassID	-
Base object instance	ObjectInstanceID	-
Scope	Scope	-
Filter	Filter	-
AccessControl	AccessControl	-
Synchronization	Synchronization	-
Attribute identifier list	AttributeIdList	-
Managed object class	-	ObjectClassID
Managed object instance	-	ObjectInstanceID
Current time	-	GeneralizedTime
Attribute list	-	AttributeList
Errors	-	Error

TAB. 9.2 – *Paramètres et classes de paramètres du service* `M-GET`

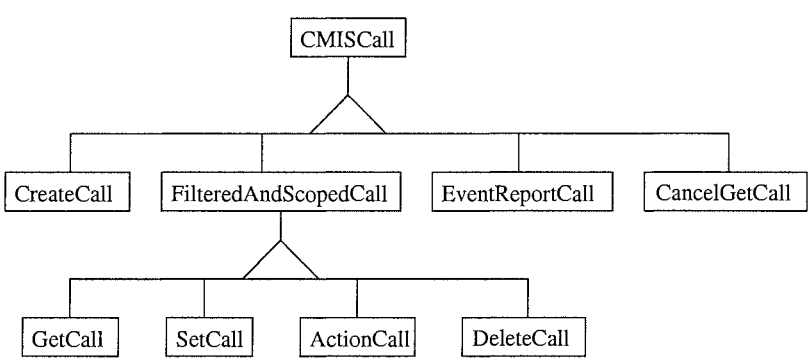


FIG. 9.5 – *Hiérarchie d'héritage entre les interfaces de requêtes*

communes qui justifient l'héritage entre ces classes sont du même genre que celles des interfaces, avec en plus la présence d'attributs dans les classes d'objets.

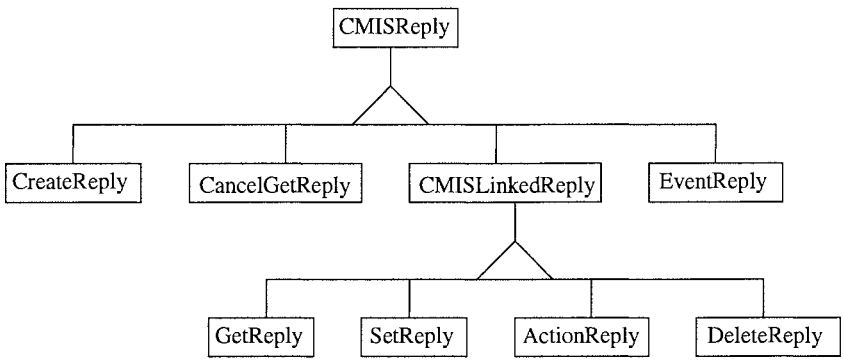


FIG. 9.6 – *Hiérarchie d'héritage entre les classes de réponses*

Module de types

La figure 9.7 représente une partie de l'arbre d'héritage entre les classes modélisant les types de paramètres de service CMIS. Les relations d'héritage permettent de caractériser la valeur de certains types de paramètres. Par exemple, la classe `Scope` est utilisée par les interfaces et les classes du module `operations` pour utiliser le mécanisme de portée. Les sous-classes désignent un des types de portée (cf. chap.2 §2.4.3) : l'ensemble d'un sous arbre (`WholeSubTreeScope`), de la racine à un niveau donné (`BaseToNthLevelScope`) et tout un niveau donné (`IndividualLevelScope`).

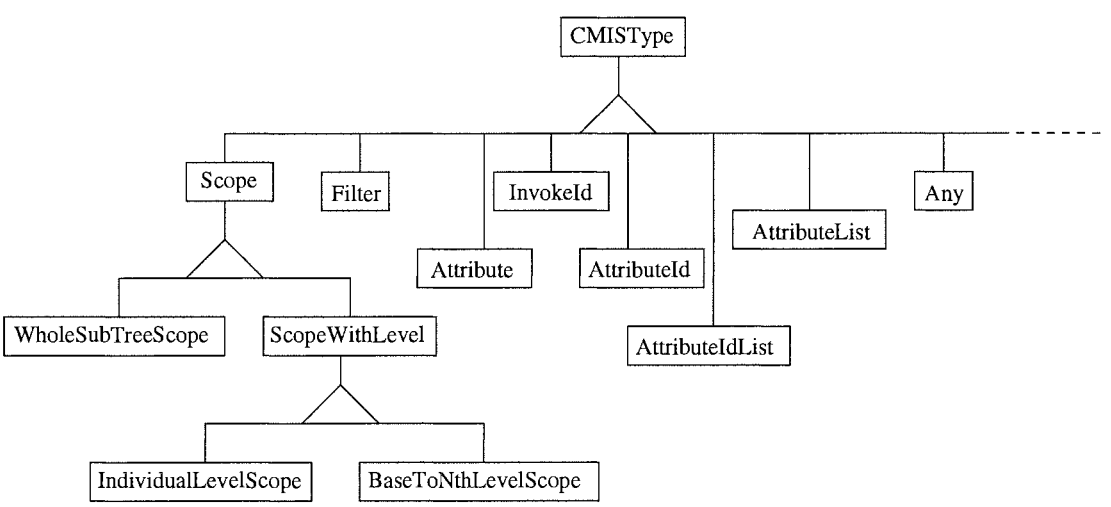


FIG. 9.7 – *Hiérarchie d'héritage entre les classes de types*

Certains types ont des relations entre eux, par exemple, figure 9.8, une relation `L` associe une instance dans le rôle `list` avec plusieurs instances dans le rôle `element`. La relation `A` correspond à la composition d'un attribut avec un identificateur (`AttributeId`) et une valeur quelconque (`CMISType`).

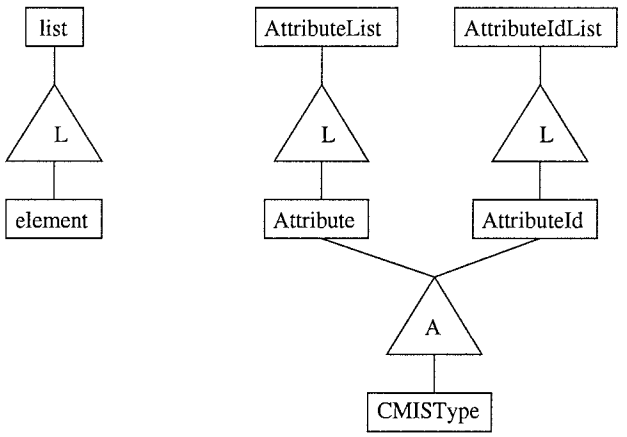


FIG. 9.8 – *Relations entre classes de types*

Lancement d'une requête

Pour qu'une application utilise `cmisOverJava`, elle doit disposer d'une instance de classe implémentant une des interfaces de requête CMIS : l'**instance de requête**. Seules des classes implémentant les interfaces aux feuilles de l'arbre d'héritage de la figure 9.5 sont à définir. L'application utilisatrice doit initialiser les paramètres du service désiré avec des instances des classes du module `types`. L'instance de requête (figure 9.9) doit être en contact avec un système de communication, dépendant de l'implantation et qui réalise le service CMIS. Toutes les classes de primitives de requête doivent implanter une méthode appellée `doIt()`, qui est spécifiée dans l'interface `CMISCall`. L'invocation de cette méthode entraîne l'émission d'une requête CMIS (ou de son équivalent) vers un agent OSI (ou équivalent). C'est à l'implantation de décider si l'appel de la méthode `doIt()` est bloquant ou non. Suite à une invocation de la méthode `doIt()` sur une instance de requête, le système de communication qui réalise la demande de service doit recevoir les informations nécessaires pour constituer les confirmations sous la forme d'instances de classes de confirmations. Ces **instances de confirmation** sont alors en paramètre de l'une des méthodes définie dans la classe de l'instance de requête :

- `reply` est invoquée lorsqu'une confirmation à une requête est reçue et que cette requête attend encore d'autres confirmations (i.e. les requêtes avec *Scope* et *Filter*);
- `complete` est invoquée lorsque la dernière confirmation qu'attendait la requête est reçue.

L'API `cmisOverJava` fournit un mécanisme d'événement qui permet de traiter les confirmations suivant cette distinction.

La méthode `doIt()` est dépendante de l'implantation car c'est elle qui va s'adresser au service de communication choisi. Pour cette raison, nous avons proposé des interfaces de requêtes plutôt que des classes. En revanche, les objets de confirmation sont des objets contenant des informations et des méthodes pour les mettre à jour ou les lire, mais sans méthode liée à la réception de confirmations par le système de communication. Ils ne sont pas dépendants de l'implantation et `cmisOverJava` les fournit comme des composants directement utilisables.

9.1.4 Les événements de `cmisOverJava`

Le modèle des événements de `cmisOverJava` est dérivé de celui défini dans l'API standard de Java. Il est notamment utilisé dans les modules liés aux conceptions d'interfaces graphique (AWT *Advanced Window Toolkit*). Il est illustré par la figure 9.10 qui montre comment ce modèle est appliqué dans `cmisOverJava`. Le principe est de réagir à des événements prédéfinis qui surviennent dans une instance

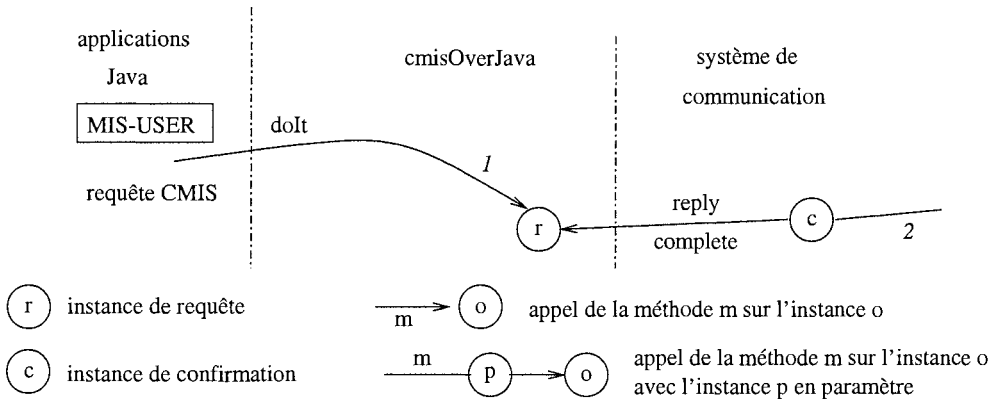


FIG. 9.9 – *Le lancement d'une requête CMIS et la réception de confirmations*

d'objet. Dans notre cas, il s'agit d'une instance d'une classe de requête et les événements à surveiller sont les arrivées des deux types de confirmations précédents. La réaction à un événement donné se concrétise par l'appel d'une méthode donnée (`onReply` ou `onComplete`) sur des objets à l'écoute de l'instance de requête : les **instances d'écoute** (*Listener*). Ces instances doivent avoir été placées dans une liste de *Listener* maintenue par l'instance de requête. Les méthodes ont en paramètre une instance d'une **classe d'événement** qui dépend de l'événement survenu dans l'instance de requête.

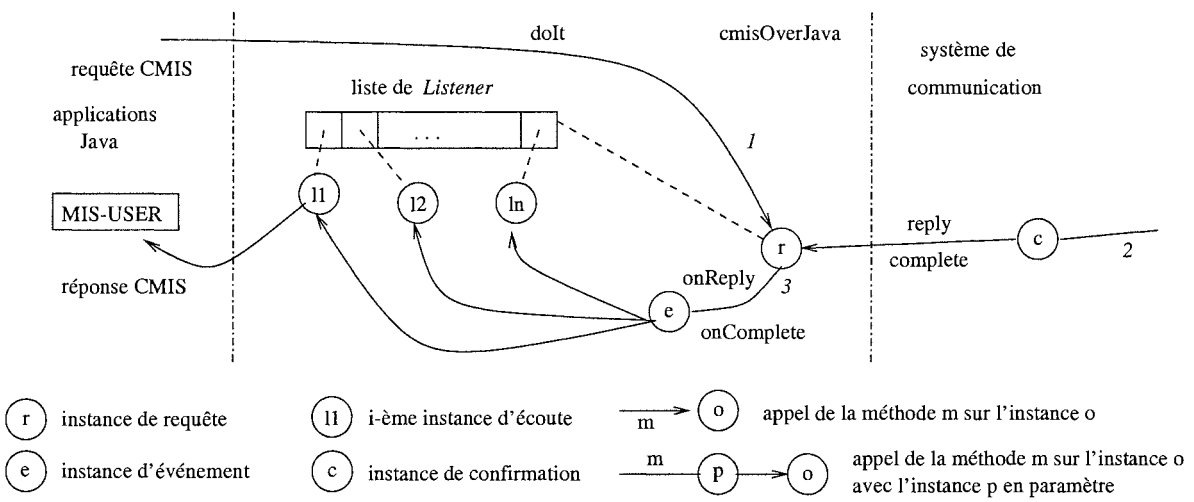


FIG. 9.10 – *Les événements et les Listener*

Les interfaces *Listener*

Le modèle de base des événements défini dans Java est représenté par l'interface `EventListener` qui doit être implantée par les objets d'écoute. Cette implantation est triviale car l'interface ne contient aucune méthode. La figure 9.11 montre la hiérarchie d'héritage des interfaces *Listener* de `cmisOverJava`. L'interface `CMISListener` définit les deux méthodes `onReply` et `onComplete`. Les autres interfaces ne définissent aucune méthode. L'interêt visé de ces interfaces est d'assurer qu'une instance d'écoute ne puisse "écouter" que des instances de requêtes d'une classe donnée. Une méthode `add-`

`Listener` est définie dans chaque interface de requête implantable et son paramètre est du type de l'interface qui lui correspond (par ex. la méthode `addListener` de l'interface `CreateCall` prend une instance implantant l'interface `CreateListener` en paramètre).

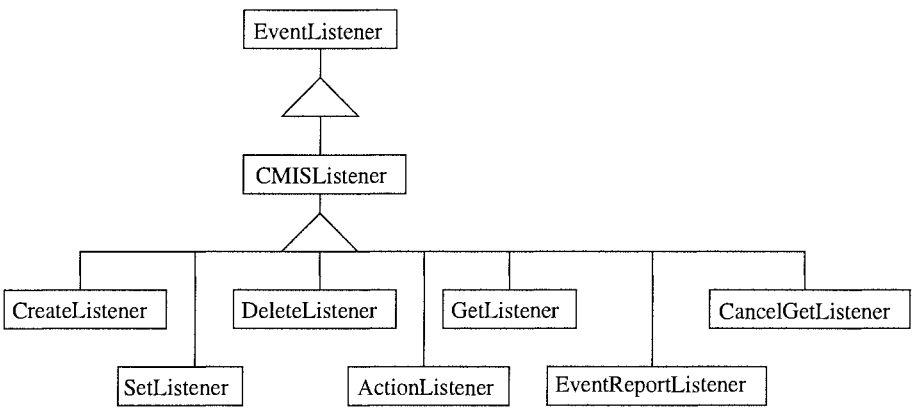


FIG. 9.11 – *Hiérarchie d'héritage des classes de Listener*

Les classes d'événements

En plus de l'interface `EventListener`, Le modèle de base des événements défini dans Java est représenté par la classe `EventObject`. Cette classe contient un attribut `source` de la classe `Object` qu est la classe de base de toutes classes Java. Le contructeur `EventObject` prend une instance de la classe `Object` en paramètre et la méthode `getSource` permet de la récupérer. La figure 9.12 montre la hiérarchie d'héritage de l'API `cmisOverJava` pour les classes d'événements. Seules les classes aux feuilles de cet arbre sont instanciables. Les classes intermédiaires avec la classe `CMISEvent` sont présentes pour permettre aux instances d'écoute de faire l'abstraction du type d'événement et de ne voir qu'un événement de confirmation. L'objet en paramètre des constructeurs des classes d'événements doit être une instance de la classe de confirmation qui à été reçue par l'instance de requête. Ainsi, par exemple, une instance de `CreateReplyEvent` est créée avec une instance de `CreateReply` en paramètre.

9.2 Une implantation avec les RMI

Cette section détaille une implantation de `cmisOverJava` qui a été réalisée au dessus du système des RMI de Java. Ce système permet d'invoquer des méthodes **objets distants** qui ne sont pas dans la même machine virtuelle Java que les **objets locaux** qui invoquent ces méthodes. Les objets distants doivent s'enregistrer auprès d'un serveur auquel s'adressent (au travers du réseau) les objets locaux. Ceux-ci récupèrent des interfaces leur permettant de manipuler les objets distants comme s'ils étaient des objets locaux.

9.2.1 Vue générale

La figure 9.13 montre le contenu de l'implantation de `cmisOverJava` et l'agent avec lequel l'API dialogue. La simulation est implantée par des **classes de requêtes** correspondantes aux interfaces de requêtes. La classe `Associations` s'intègre à l'implantation, elle permet de maintenir une communication par RMI avec un agent distant. Cette classe modélise une notion d'association similaire à celle que

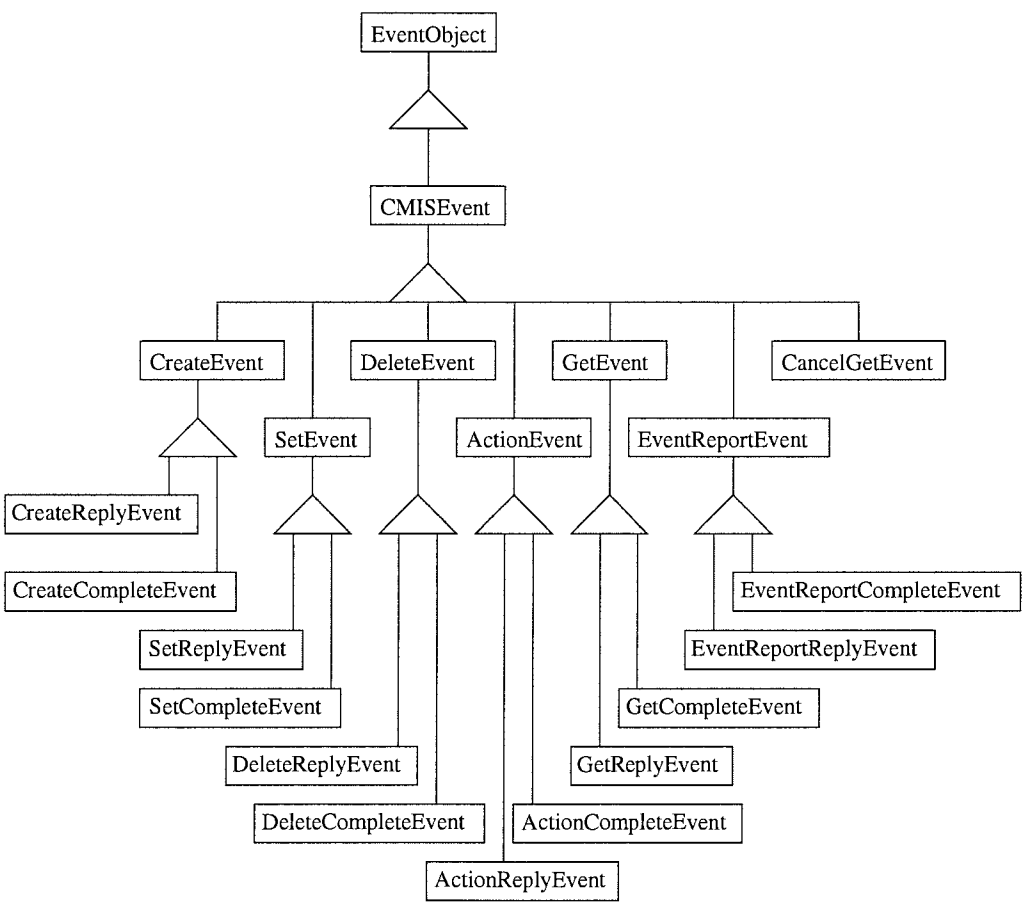


FIG. 9.12 – *Hiérarchie des événements*

définit l'élément ACSE de la couche application (cf. chap.2 §2.4 fig.2.6). Une association est établie entre deux utilisateurs de service CMIS, l'un invoquant des services et l'autre les réalisant (respectivement le gestionnaire et l'agent). Dans notre implantation, cela se traduit par le fait que toutes les requêtes avec un agent donné se font par l'intermédiaire d'une instance donnée de la classe `Association`. Les services que rend l'élément ACSE, comme la version de CMIS utilisée ou la négociation des unités fonctionnelles (i.e. les fonctions de gestion de l'OSI), ne sont pas pris en compte par la classe `Association`.

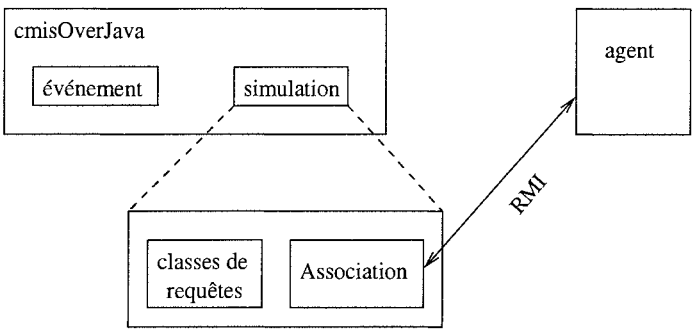


FIG. 9.13 – *Vue générale de l'implantation RMI*

9.2.2 Les classes de requêtes

La table 9.3 montre les classes de requêtes et les interfaces de requêtes de `cmisOverJava` qu'elles implantent. La figure 9.14 montre l'arbre d'héritage de ces classes. On peut noter la ressemblance entre

Interface de requête	Classe de requête
CreateCall	CreateCallImpl
DeleteCall	DeleteCallImpl
GetCall	GetCallImpl
CancelGetCall	CancelGetCallImpl
ActionCall	ActionCallImpl
SetCall	SetCallImpl
EvenReportCall	EventReportCallImpl

TAB. 9.3 – *Classes de requêtes implémentant les interfaces de requêtes*

celui des interfaces (fig. 9.5), il faut toutefois spécifier l'héritage dans les classes. La classe `UnicastRemoteObject` est nécessaire pour les objects qui implantent des interfaces et dont les instances seront accédées à distance par les RMI.

9.2.3 La classe `Association`

Les rôles d'une instance de la classe `Association` (ou **association**) sont les suivants :
- établir un contact avec l'agent;
- instancier à la demande des classes de requêtes;
- transmettre les références de ces requêtes à l'agent;
- recevoir de l'agent les instances de confirmations.

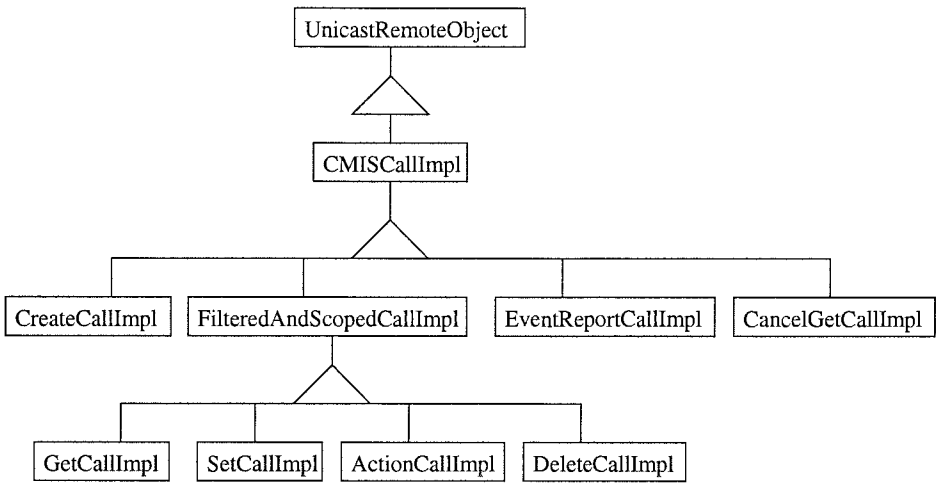


FIG. 9.14 – *Hiérarchie d'héritage des classes de requêtes*

Pour le premier point, l'association doit s'adresser au serveur auprès duquel l'agent a dû préalablement s'enregistrer. Elle reçoit en retour une référence RMI, qui est vue comme une instance d'objet implantant une interface Java `Agent`, indissociable de l'implantation. Le deuxième point est réalisé à la suite d'une invocation de méthode de l'association. Par exemple, figure 9.15(a), la méthode `newCreateOperation()` retourne une référence à une instance de la classe `CreateCallImpl`, vue comme une interface `CreateCall`. Parallèlement à ce retour, l'instance créée est placée dans une liste d'instances de requêtes dans l'association. Le deuxième point (figure 9.15(b)), suite à une invocation de la méthode `doIt()`, entraîne l'envoi d'une référence RMI de l'instance de requête vers l'agent par l'invocation d'une de ses méthodes (`createCallIndication` dans l'exemple). Cette invocation se fait sur la référence RMI de l'agent contenue dans l'association. Le dernier point est traité dans la section suivante.

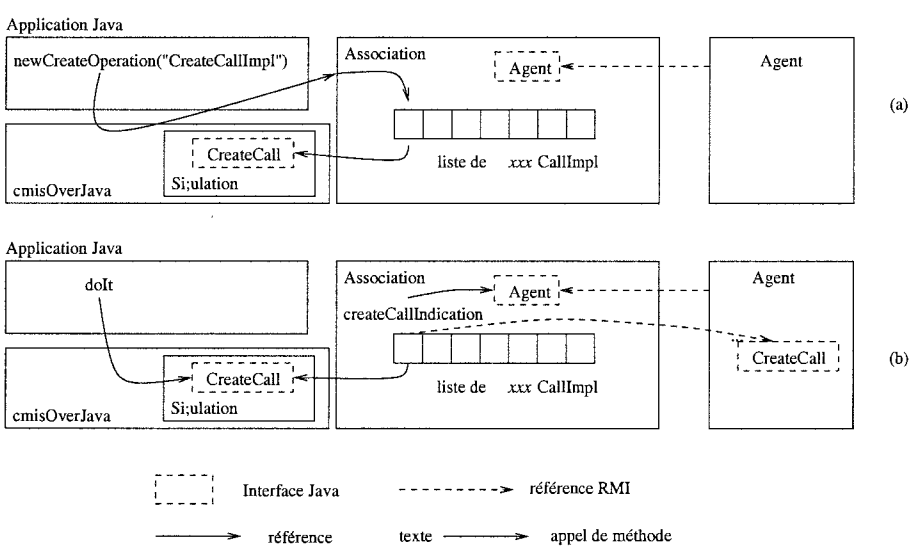


FIG. 9.15 – *Fonctionnement de l'Association*

9.2.4 L'agent

Parallèlement à l'implantation RMI de `cmisOverJava`, nous avons défini une interface `Agent` dont les instances peuvent envoyer une référence RMI permettant de les manipuler à distance, par exemple avec la méthode `createCallIndication` de la section précédente. Associé à cet agent, nous avons dû définir des interfaces et des classes Java pour modéliser une MIB d'instances d'objets gérés. Cette MIB est générique car elle peut créer des intances de toute classe GDMO. Les objets gérés qui y sont instanciés n'ont pas de contact avec de quelconques ressources, les valeurs d'attributs peuvent être modifiées librement mais il n'y a pas d'implantation pour les traitements liés à un action d'objet géré.

Nous considérons donc dans la suite qu'une instance d'agent est en contact avec une MIB qui exécute les services modélisés par les interfaces de requêtes et qui fournit à l'agent des instances des classes de confirmations correspondant aux réponses. A l'invocation de ses méthodes d'indication de requête (`createCallIndication` de la figure 9.15), l'agent créé, avec la méthode `newOperationThread`, une instance d'un objet dont le code va s'exécuter de manière autonome et concurente (objets *Threads*). La figure 9.16(a) montre cette instanciation, l'objet *Thread* va poursuivre le traitement de la requête, libérant l'agent pour recevoir d'autre interfaces de requêtes. La figure 9.16(b) décrit la fin de l'interaction de gestion. L'objet *Thread* est détruit dès que la MIB a récupéré la référence RMI de la requête. La MIB procède alors à son traitement et suivant les réponses à fournir, invoque les méthodes `complete` ou `reply` avec les instances de confirmations issues du traitement de la requête. Les instances des objets de confirmations sont sérialisées, c'est à dire que des copies en sont véhiculées vers l'association. Les instances de confirmations sont alors accessibles par l'API `cmisOverJava` qui peut enclancher son mécanisme des événements.

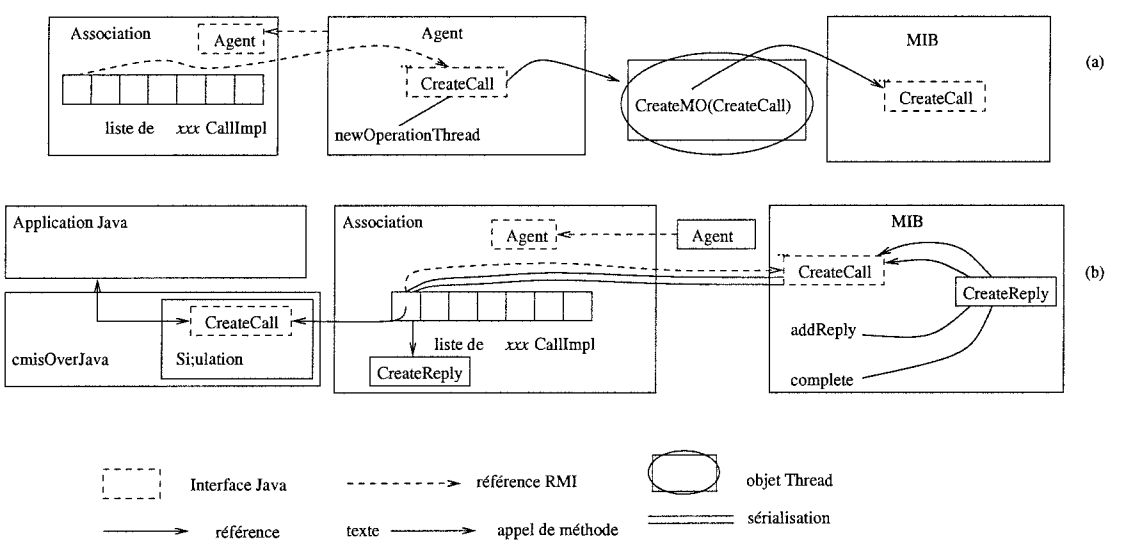


FIG. 9.16 – *Fonctionnement de l'Agent et de la MIB*

9.2.5 Un exemple d'utilisation

La figure 9.17 montre un exemple d'utilisation de `cmisOverJava` pour la création d'un objet géré dans une MIB. Le programme principal (fig.9.17(a)) établit une association avec un agent localisé sur une machine, `jenna`, dont le serveur de référence RMI attend les requêtes sur le port 6543. Le serveur est distingué d'éventuels autres serveurs par un identificateur (ici `CMISAgent`). C'est par l'association

que se créée une requête de création (interface `CreateCall`, implantée par la classe `CreateCallImpl`). Cette requête est initialisée avec des paramètres tels que la classe de l'objet à gérer et une liste de valeurs initiales d'attributs. Un objet *Listener* est ensuite instancié, puis ajouté à la liste de *Listener* de la requête. Finalement l'invocation de la méthode `doIt()` envoie la requête, laissant à l'objet *Listener*, la responsabilité de traiter la réponse.

La figure 9.17 (b) montre la classe du *Listener* pour la création précédente. Comme une création n'entraîne qu'une réponse, seul le code de la méthode `onComplete` est nécessaire. Elle est paramétrée par une instance de l'événement qui correspond à la réponse d'une création (`CreateReplyEvent`). L'instance de confirmation de classe `CreateReply` est récupérée par la méthode `getSource()` de l'événement. Elle est ensuite affichée à l'écran (`System.out.println`).

(a)

```
public class CmisOverJavaExample
{
 public static void main()
  throws java.rmi.RemoteException,
  ...
 {

  // Create a new Association
  // with jenna's Agent
  AssociationImpl ass = new
   AssociationImpl(
    "rmi://jenna:6543/CMISAgent");

  // set a CreateCall
  CreateCall createCall =
   ass.newCreateOperation(
    "CreateCallImpl");

  // Initializing parameters
  createCall.setInvokeId(new
   InvokeId(1));
  createCall.
   setManagedObjectClass(
    new ObjectClassID("system"));

  // Create and initializing
  // an attribute list
  AttributeList lAttributeList =
   new AttributeList();
  lAttributeList.add("systemId",
                  new Any("1"));
     ...

  // set the attribute list
  // in the call
  createCall.setAttributeList(
   lAttributeList);

  // Create a listener object
  CreateListenerExample listener =
   new CreateListenerExample();

  // add the listener
  createCall.
   addCreateListener(listener);

  // send the create call
  createCall.doIt();
 }
}
```

(b)

```
public class CreateListenerExample
 implements CreateListener
{
 public void onComplete(
  CreateReplyEvent pCr)
 {
  try{
   // retrieve the CreateCall
   CreateCall lCreateCall =
    (CreateCall)pCr.getSource();

   // retrieve the CreateReply
   CreateReply lCreateReply =
    pCr.getSource();

   // shows the Create
   // reply parameters
   System.out.println(
    "Managed object instance : "
     + lCreateReply.
     getManagedObjectInstance().
                    getID());
  ...
  }
   catch(Exception e){
    e.printStackTrace();
   }
 }
}
```

FIG. 9.17 – *Appel du service M-CREATE avec* `cmisOverJava` *sur RMI*

9.3 Conclusions

L'API `cmisOverJava` permet à des applications, comme celles fondées sur RelMan, de faire de la gestion de réseaux dans le rôle d'un gestionnaire. La conception de ces applications est simplifiée d'une part avec l'abstraction de requêtes par des objets qui cache la compléxité de leur émission et d'autre part avec le mécanisme des événements qui induit une programmation de ces applications où sont clairement séparés les traitements avant l'émission d'une requête et les traitements suivant la réception de confirmations. L'API `cmisOverJava` est également la première expérience d'une API CMIS en Java. Elle est inspirée des travaux du NMF [Cha97] et de X/Open [SH97] sur une API en C++. `cmisOverJava` se distingue des propositions existantes notamment par son mécanisme d'événements.

L'implantation de `cmisOverJava` sur le système des RMI permet de réaliser des opérations de gestion sans avoir à supporter la manipulation d'une pile complète de communication. La transparence à la distribution qu'apportent le système RMI a grandement facilité l'implantation. La séparation claire entre les interfaces et les implantations nous a permis de développer l'API sur différentes plates-formes de gestion, démontrant l'interêt de nos choix. L'implantation au dessus d'une pile est à ce jour plus complète que celle avec les RMI. Elle offre notamment l'ensemble des types ASN.1 ainsi que l'API côté agent. L'API `cmisOverJava` manque encore d'un moyen pour que les utilisateurs de service puissent s'inscrirent à des notifications d'objes gérés. Seul l'utilisation du service comme intiateur de requête est possible avec l'API décrite dans ce chapitre. Ce point est une extension de première importance à apporter à l'API sur Java. Elle est implantée dans une version au dessus d'une API commerciale.

Chapitre 10

Une implantation de RelMan en Java

Ce chapitre présente une implantation en Java de concepts qui ont été définis dans RelMan. Cette réalisation utilise l'API `cmisOverJava` du chapitre précédent. Elle est sensée s'exécuter sur une machine virtuelle Java, offrant à des applications, également écrites en Java, la possibilité d'intégrer dans leur code des opérations sur les relations gérées. Ces dernières étant maintenues par une partie non accessible à ces applications, contenant des composants comme la RMIB, l'inférence et contrôle de l'intégrité. L'implantation que nous proposons est également dotée d'une interface graphique pour gérer les relations.

L'implantation présentée ici est un prototype dont le but est de valider les concepts de RelMan et d'en produire une architecture logicielle de base. Des développements à plus grande échelle pourront s'en inspirer ou se servir des interfaces qui y sont définies.

10.1 Vue générale

La figure 10.1 met en place les différents modules logiciels qui entrent en jeu dans l'implantation de RelMan. Le module `services` permet de constituer des requêtes d'opérations de relations et de les envoyer vers des relations gérées. Le module `rmib` maintient une représentation logique des relations gérées. Le module `repository` contient une représentation des modèles de gestion (GDMO, GRM et GRM+) qui sont instanciés dans le réseau vu par RelMan. Ce dictionnaire des données est initialisé par le module `frontEnd` qui analyse les spécifications normalisées. Les deux modules précedents font partie de l'environnement MODERES [FNA96a]. Le module `mapper` prend en charge l'exécution des opérations de relations émanant du module `services` et les traduits en requêtes de service CMIS qui sont construites et émises avec l'API `cmisOverJava`. Le module `dispatcher` est utilisé pour les notifications de relations. Le module `infers` effectue la fonctionnalité du composant `Infers` du même nom dans le R-ReactiveManager. Enfin, une interface graphique à été développée (module `GUI`) pour permettre à un utilisateur humain de manipuler RelMan. Certains modules de l'implantation de RelMan peuvent également être utilisés par des applications autonomes.

10.2 Le module `services`

Le module `services` prend en charge les services offerts par les APIs `RS` et `NS` de RelMan. Il est divisé en trois modules (figure 10.2), `operations` et `subscriptions` pour ces APIs et `types` pour les types des paramètres de services définis dans les deux derniers modules. Comme le montre la figure, les modules `operations` et `subscriptions` utilisent les modules `mapper` et `dispatcher`, à la manière des composants de RelMan.

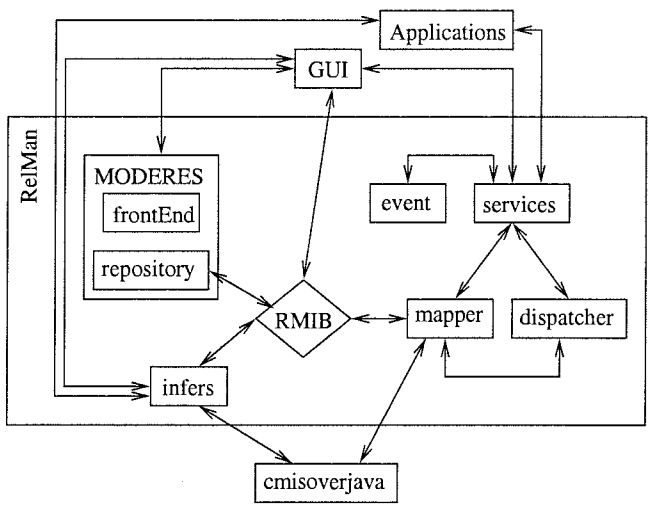


FIG. 10.1 – *Vue générale de RelMan*

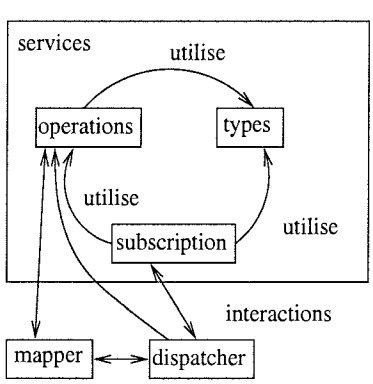


FIG. 10.2 – *Le module* `services`

10.2.1 Les interfaces de requête

La figure 10.3 montre la hiérarchie d'héritage des interfaces permettant de constituer des requêtes d'opérations de relation ainsi que les réponses. On retrouve une interface Java par primitive de l'interface `RS` de RelMan. L'interface `RSRoleList` regroupe les interfaces de requêtes qui peuvent avoir des DNs (*Distinguish Name*) d'objets gérés comme paramètres. La super-interface `RSCall` définit une méthode `doIt()` qui entraîne un traitement devant amener à l'exécution de l'opération.

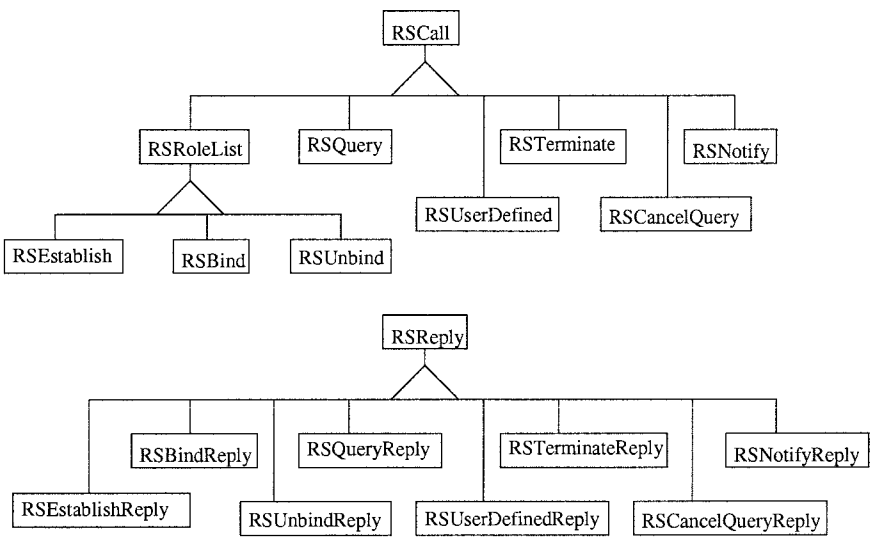


FIG. 10.3 – *Hiérarchie d'héritage des interfaces de requêtes et de réponses*

10.2.2 Les interfaces d'enregistrements

La figure 10.4 montre les deux interfaces Java définies pour ce module. Elle correpondent à celles de l'API `NS` de RelMan. L'interface `NSCall` définit une méthode `doIt()` pour l'activation des mécanismes permettant à une application de prendre en compte des notifications de relation.

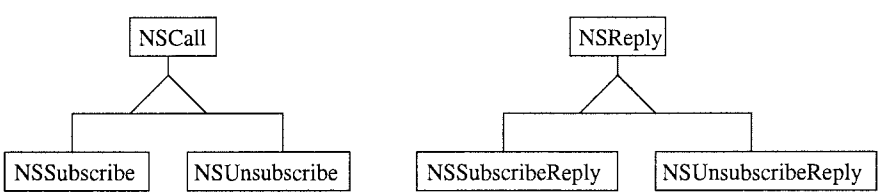


FIG. 10.4 – *Hiérarchie d'héritage des interfaces d'enregistrements et de leur réponses*

10.2.3 Les types

La figure 10.5 montre la hiérarchie d'héritage entre les interfaces modélisant les types des paramètres des services des modules `services` et `subscription`. L'interface `RoleItem` est utilisée par l'interface `RSRoleList` (fig. 10.3). Les instances implantant l'interface `RoleParticipant` contiennent un identificateur de rôle de relation gérée et un DN d'objet géré. Elles sont utilisées lorsqu'une opération

de relation vise à ajouter ou enlever des participants dans des rôles donnés à une relation. L'interface `RelObjectType` permet de désigner un objet géré existant utilisé comme objet de relation dans une représentation de relation. Lorsqu'une opération de relation doit créer un objet géré dans un rôle donné, le DN de celui qui doit lui être supérieur est nécessaire. L'interface `SupObject` modélise ce fait sans rajouter de caractéristiques à l'interface `RoleParticipant`.

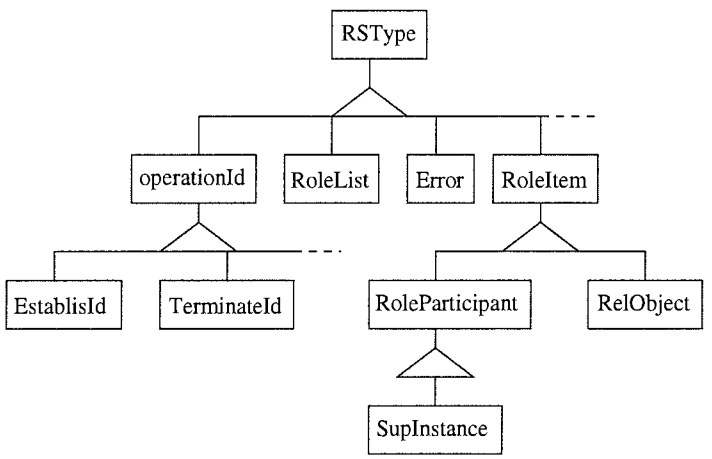


FIG. 10.5 – *Hiérarchie d'héritage des interfaces de types*

Dans les spécifications en GRM+, le type des arguments des opérations de relation n'est pas contraint. Dans notre implantation, il est nécessaire que ces spécifications puissent correspondrent aux interfaces Java de cette section. Les valeurs du paramètre **infoGrmP** (cf. chap.7 table 8.1) doivent être instanciées dans des classes Java qui implémentent ces interfaces.

A la différence de l'API `cmisOverJava`, il n'y a pas de type défini pour les instances de relations. L'ISO n'a pas établi de convention pour nommer les relations gérées, ormis le cas de représentation avec un objet de relation dont le DN remplit cette fonction. L'implantation décrite dans ce chapitre considère que les instances de requêtes, d'opérations et d'enregistrements à des notifications, de relations sont dans la même machine virtuelle Java que les instances de relations présentes dans la RMIB (vue plus loin). Ce sont les références (éventuellement RMI) des ces instances qui servent de paramètres aux requêtes d'opérations et d'enregistrement.

10.2.4 Les *Listener*

Le module d'événement remplit la même fonctionnalité que pour `cmisOverJava` (cf. chap.9 §9.1.4), avec le module `mapper` jouant le rôle du service de communication. La figure 10.6 montre les interfaces de *Listener* et les classes qui ont été définies pour contrôler les traitements suite aux réponses des opérations de relation. La notion de réponses multiples n'étant pas présente dans les opérations de relations, les objets d'événement ne modélisent que des réponses uniques aux opérations.

10.2.5 Les notifications

Le cas des notifications de relation est particulier (figure 10.7). Une instance d'une classe implantant l'interface `RSNotify` (objet `n`) doit être créée dans le code de la méthode `doIt()` définie dans l'interface `NSSubscribe` (flèche *1* vers l'objet `ns`). Une référence à cette instance créée est retournée, par l'intermédiaire de l'interface `NSSubscribeReply` (non représentée), à l'application qui avait invoqué la méthode et à un objet du module `dispatcher` (flèches en pointillé partant de l'objet `n`). C'est

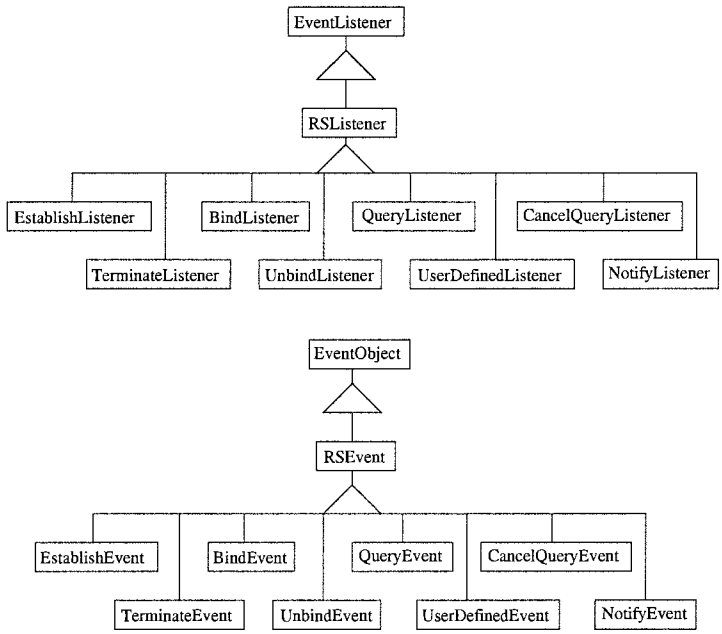


FIG. 10.6 – *Hiérarchie d'héritage des interfaces Listener et des classes d'événements*

à l'application d'adjoindre à cette nouvelle instance des objets *Listener* (objet `l` ajouté par la méthode `addListener`, flèche *2*), puis de lui invoquer la méthode `doIt()` (flèche *3*). Ce mécanisme remplace le paramètre **destAppli** (cf. chap.8 tables 8.1 et 8.2). A la suite de cette invocation, le `dispatcher` déclanche un mécanisme d'attente de notification (vu plus loin). La détection d'une notification dans ce composant entraîne l'invocation d'une méthode `complete` avec un objet de confirmation (objet `r`, flèche 4) en paramètre. L'objet `n` de requête propage alors cette confirmation aux objets *Listener* (`l`) par l'intermédiaire d'un objet d'événement (`e`) et la méthode `onComplete` (flèche 5). Les traitements propres aux applications de gestion peuvent alors commencer.

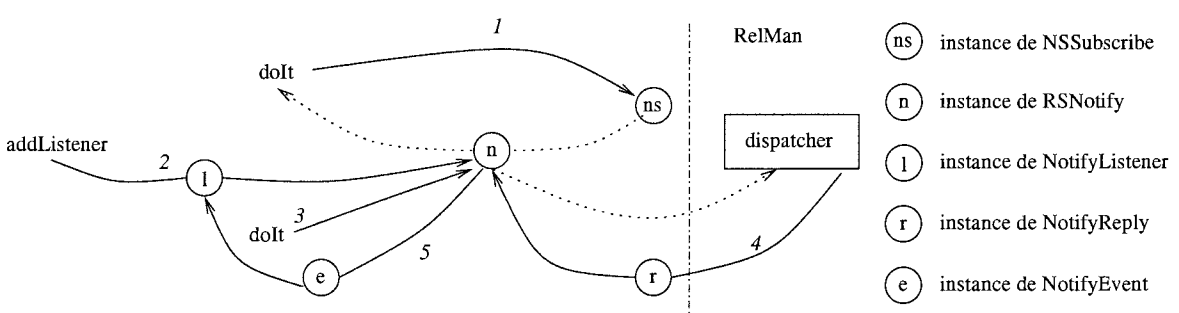


FIG. 10.7 – *Les notifications de relations*

10.3 La RMIB

La figure 10.8 montre l'ensemble des interfaces qui constituent le module `rmib`. La relation `R1` spécifie qu'une `RMIB` (instance d'une classe implantant l'interface `RMIB`) contient un ensemble de `MR`

(*Managed Relationship*) et un ensemble de `Participant` avec une égalité entre ce dernier et l'ensemble des `Participant` qui sont dans les `MR` de la `RMIB`. L'ensemble des `Participant` s'obtient par composition des relations `R2` et `R3`. Une `MR` contient exclusivement un certain nombre de `Role` (`R2`) et un `Role` contient (non exclusivement) un certain nombre de `Participant` (`R3`).

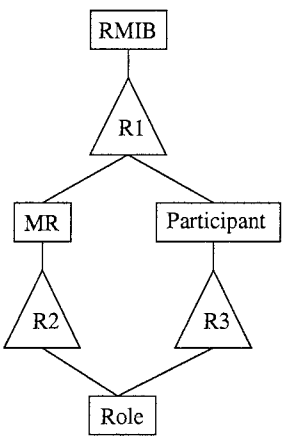


FIG. 10.8 – *Relations entre les interfaces de la RMIB*

Le maintien des cardinalité de rôles des relations est effectué dans les `Role` lors des ajoûts ou retraits de `Participant`. L'interface `RMIB` définit des méthodes qui permettent le maintien des cardinalités de relation liées aux `Role`. Il est en effet nécessaire d'avoir une vue de toutes les relations gérées pour effectuer ce contrôle.

10.3.1 Les interfaces de `cmisOverJava`

L'API `cmisOverJava` est utilisée pour lancer les requêtes d'opérations de système traduites des opérations de relation. La figure 10.9 montre les interfaces et classes dérivées spécialement pour l'utilisation de `cmisOverJava` dans RelMan. On note que l'interface associée au service M-CANCEL-GET n'est pas utilisée car il ne peut y avoir de traduction d'opération de relation avec ce service. Comme pour le cas des *Listener* de RelMan, seuls les événements correspondant à une dernière confirmation sont utilisés, les traductions agissant sur plusieurs objets en une opération de système ne sont pas prévues dans cette première implantation de RelMan.

10.4 Le `mapper`

Lorsqu'une requête de service de relation est prête, l'invocation de sa méthode `doIt()` envoie une référence de cette requête au module `mapper`, dont nous décrivons les fonctionnalités au lieu des interfaces. Le `mapper` doit alors procéder à la traduction d'une opération de relation en une ou plusieurs opérations CMIS. Au cours de cette traduction, plusieurs contrôles sur la sémantique des paramètres (instances implantant les interfaces de types de la section 10.2.3) sont effectués. Pour chaque opération de relation, un algorithme spécifique est nécessaire. Par exemple, nous détaillons ci-dessous les traitements liés à une opération ESTABLISH:

vérifications de la présence de la classe de relation et de la représentation dans le `repository`,
vérification de la présence de l'opération ESTABLISH dans la classe de relation,

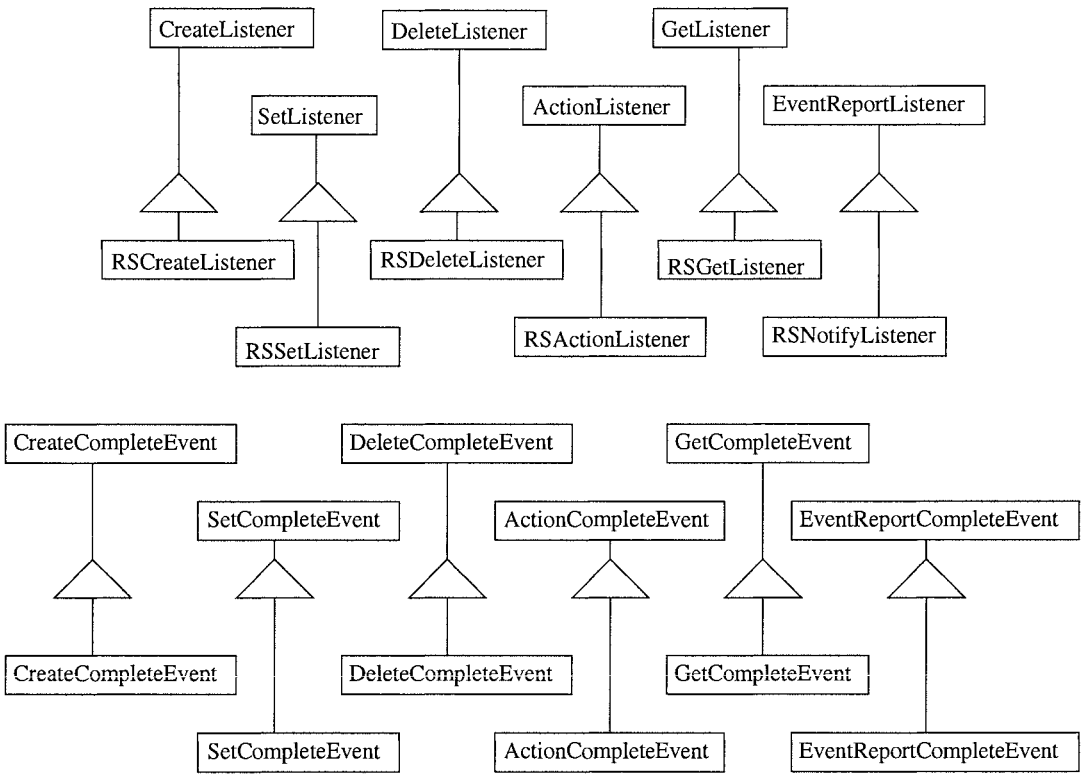


FIG. 10.9 – *Réutilisation des éléments de* `cmisOverJava`

pour chaque opération système:
si la traduction est de la forme : `CREATE c OF ROLE r:`
si le rôle `r` est représenté dans la relation par le rôle subordonné d'une corrélation de noms et que le rôle supérieur de cette corrélation représente un autre rôle `s` de la relation :
vérifier qu'il existe déjà au moins un participant dans le rôle `s`,
sinon
rechercher dans **infoGrmP** de la requête, une instance de SupInstance avec le rôle `r`,
vérifier par un service M-GET que l'instance supérieure existe dans une MIB (utilisation de `cmisOverJava`),
vérifier dans le `repository` qu'une corrélation de nom existe entre la classe de l'instance supérieure et la classe c,
finsi
vérifier que la cardinalité permise de rôle du rôle `r` n'est pas dépassée si on ajoute un participant,
créer la requête `CreateCall`,
ajouter une instance de `RSCreateListener`.
Invocation de la méthode `doIt()` sur le `CreateCall`,
à la réception d'un `RSCreateEvent`, ajouter le DN de l'objet géré créé dans la relation de la RMIB (qui est créée si elle n'existe pas),
sinon[51]
si la traduction est de la forme : `ADD att OF ROLE r:`
rechercher dans **infoGrmP** de la requête, une instance de `RoleParticipant`, dont la valeur de DN qu'elle contient doit être ajoutée à l'attribut `att`,

51. dans ce cas, au moins une autre opération CREATE a déjà été effectuée pour mettre un objet dans le role `r`

vérifier que l'instance d'objet géré existe bien et est de la bonne classe (i.e. d'une des classes du rôle désigné par l'attribut `att`), la vérification se fait dans la RMIB en premier et dans une MIB (avec `cmisOverJava`) s'il n'est pas déjà un participant à une relation gérée,
vérifier que la cardinalité permise de rôle du rôle désigné par att n'est pas dépassée si on lui ajoute un participant,
créer la requête `SetCall`,
ajouter un `RSSetListener`,
à la réception d'un `RSSetEvent`, ajouter le DN de l'objet géré désigné dans la relation de la RMIB (qui est créée si elle n'existe pas). Si le DN est déjà présent dans la RMIB, le `Participant` est simplement rajouté à la relation.

finSi
finSi
finPour

Lorsqu'une vérification échoue, les traitements sont arrêtés et une exception spécifique est envoyée en réponse de l'invocation de la méthode `doIt()` sur la requête d'opération de relation.

Lorsque le module `mapper` reçoit une requête, il instancie un objet *Thread* qui va prendre en charge la traduction de l'opération de relation (figure 10.10). Cet objet va instancier au cours de son existance, plusieurs requêtes de service issues de l'API `cmisOverJava`.

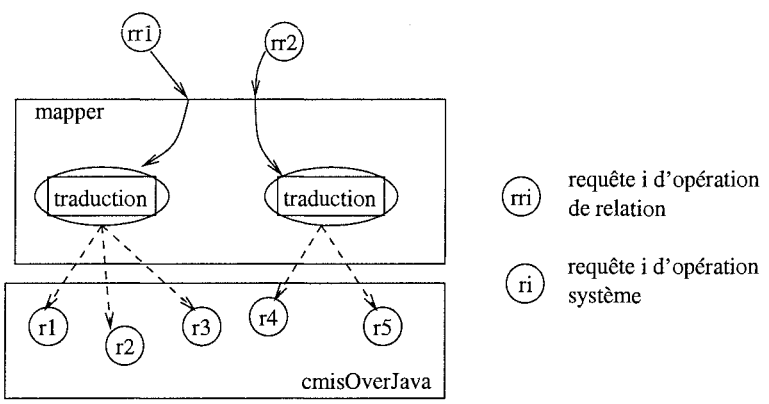


FIG. 10.10 – `mapper` *et* `cmisOverJava`

10.5 Le `dispatcher`

Le `dispatcher` reçoit les requêtes de notification de relation (`RSNotify`). Il maintient un ensemble des requêtes de notification déjà demandées, classées par relation gérée et par notifications définies dans la relation. La figure 10.11 montre que deux relations gérées `r1` et `r2` ont des requêtes de notification en attente sur leurs notifications, respectivement `n1`, `n3` et `n2`. Pour chaque élément de l'ensemble, une unique requête de notification de relation est transmise au module `mapper`.

10.6 Le module `infers`

Le module `infers` permet d'inférer de nouvelles relations gérées à partir des objets gérés présents dans le domaine de gestion de RelMan (i.e. l'ensemble des MIBs auxquelles il a accès). La figure 10.12 montre les interfaces définies pour ce module. Elles correspondent aux primitives de services définies dans l'architecture de RelMan (cf. chap.8 §8.3.2). Comme le montre la figure, une requête d'inférence

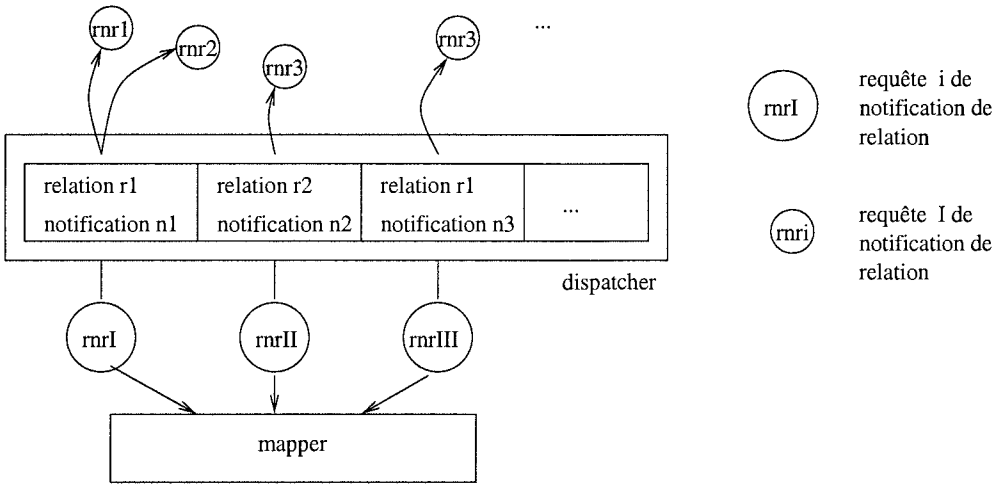


FIG. 10.11 – *Le module* `dispatcher`

est traduite par plusieurs requêtes M-GET, offertes par `cmisOverJava`. La décomposition du graphe des rôles en composantes connexes et la détermination des caractéristiques à rechercher dans les objets gérés pour qu'ils soient des participants à une relation de la classe donnée dans la requête (paramètres **relationClass** et **relationMapping** chap.8 table 8.3) sont calculés par un objet *thread* du module `infers`.

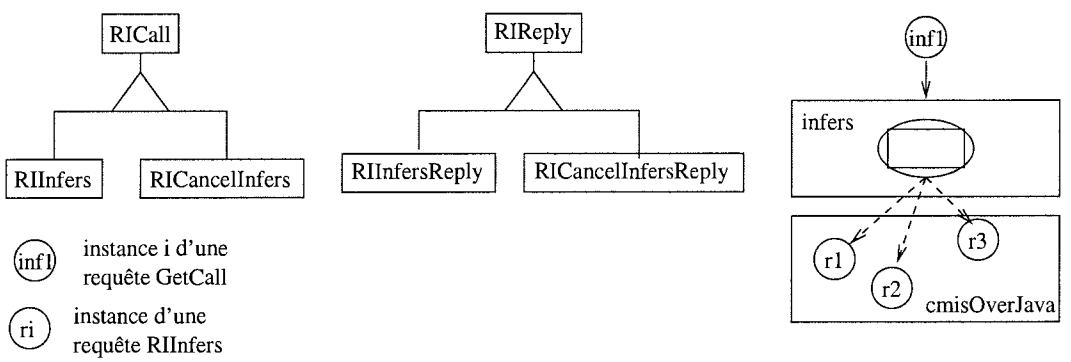


FIG. 10.12 – *Le module* `infers`

10.7 L'interface graphique

Cette section montre quelques copies d'écrans de l'interface graphique développée au dessus de RelMan (module `GUI` fig.10.1). Dans la suite, un utilisateur de RelMan va manipuler des objets gérés situés dans une MIB, au travers d'opérations de relation. La classe de relation et sa représentation qui sont utilisées dans cet exemple sont dans l'annexe D.

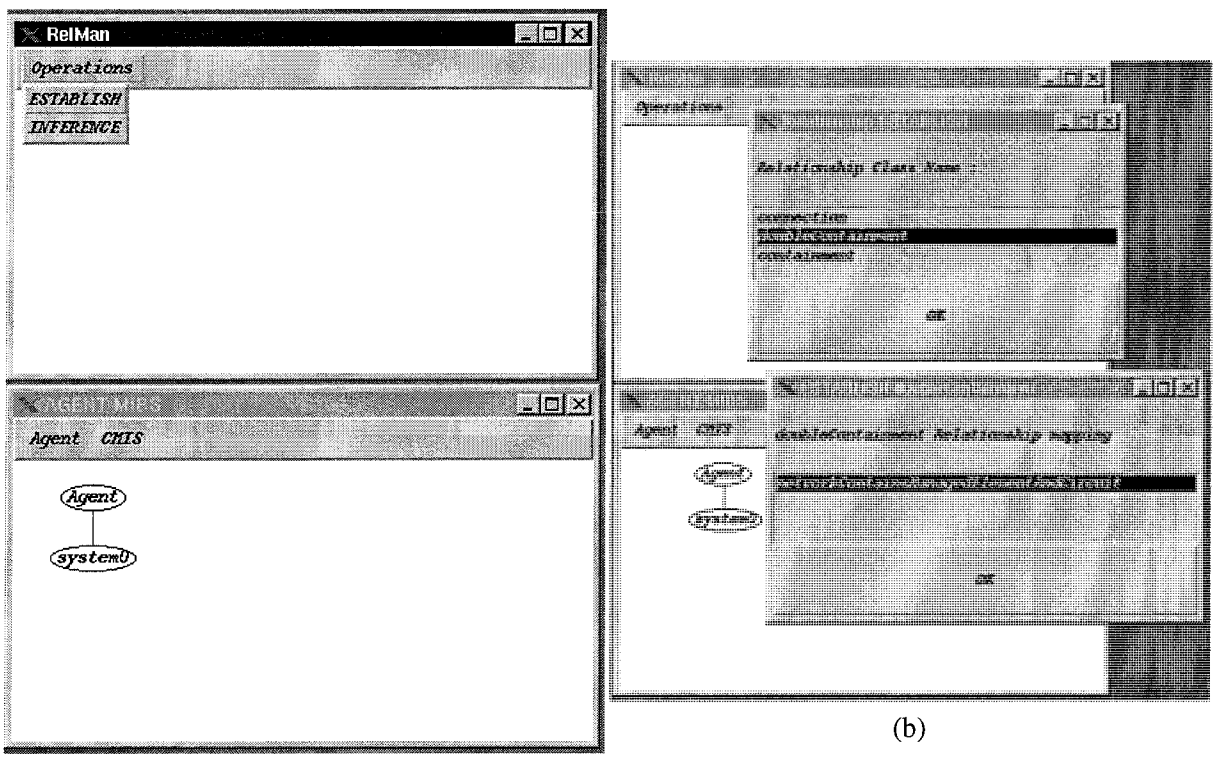
10.7.1 Démarrage

Le premier écran (figure 10.13.a) est obtenu après le démarrage de l'application, le chargement de modèles de gestion (GDMO, GRM) et la connexion à un agent. La fenêtre supérieure est l'interface

graphique de RelMan et l'inférieure est une simulation d'un agent (surtout de sa MIB) avec laquelle RelMan vient de se connecter. À l'état initial de cet agent, seul l'objet `system0`, de la classe `system` [ISO92h] est présent. Le menu `Operation` de RelMan propose deux types d'opérations, l'établissement d'une nouvelle relation gérée, où l'inférence de nouvelles relations.

10.7.2 Etablissement d'une nouvelle relation

Après avoir choisi `ESTABLISH` dans le menu `Operation`, une première fenêtre (figure 10.13.b) permet de sélectionner une classe de relation gérée parmi la liste des spécifications chargées préalablement. Dans l'exemple, la classe `doubleContainment` est choisie. Lorsque ce choix est fait, une deuxième liste permet de décider d'une représentation de relation de la classe choisie. Ici, seule la représentation `networkContainsManagedElementAndCircuit` est définie.



(a) (b)

FIG. 10.13 – *Initialisation d'une classe de relation*

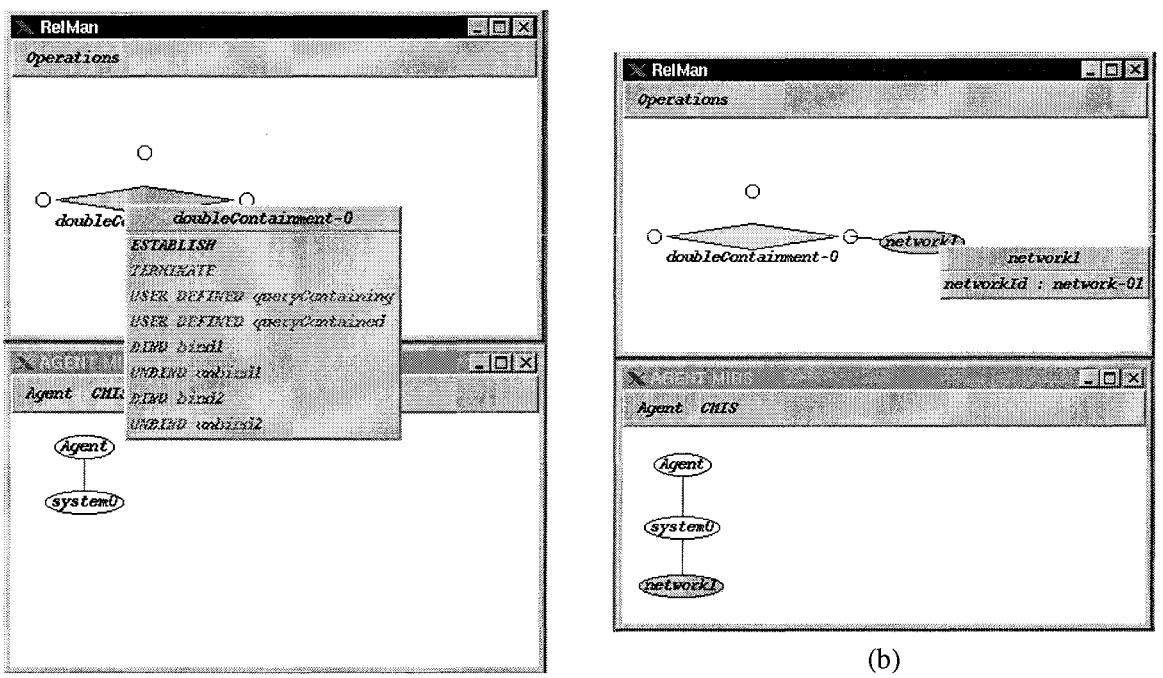
10.7.3 Première opération

Lorsque la classe et la représentation sont choisies, il apparaît sur l'écran de RelMan (figure 10.14.a) une schématisation d'une relation gérée (losange et cercles signifiant les rôles). Si on active cette relation, un menu déroulant fait apparaître les opérations de relation qui sont possibles sur cette instance potentielle. On note que parmi l'ensemble de ces opérations, seule l'opération de type `ESTABLISH` est sélectionnable car la relation n'existe pas encore dans la RMIB. Sa sélection entraîne la création d'un objet de la classe `network` ([M3195] et annexe D). Nous passons les étapes où l'interface graphique demande à l'utilisateur de choisir une instance supérieure dans la MIB de l'agent. L'instance `systeme0` est choisie et l'opération peut se poursuivre par le choix de valeurs initiales d'attributs de l'objet à

créer par une requête de service `M-CREATE`. Ces valeurs pouvant être demandées à l'utilisateur ou être définies dans le comportement de la représentation de la relation en GRM+.

10.7.4 État après établissement

Quand l'opération `ESTABLISH` est terminée, l'écran de RelMan contient les participants de la relation. Dans la figure 10.14.b, l'opération `ESTABLISH` est traduite par une création de l'objet `network1` dans la MIB de l'agent et dans la RMIB de RelMan. Une erreur dans le traitement de cette opération nous aurait ramenés au cas de la figure 10.14.a. La figure montre une fonctionnalité de l'interface qui permet de visionner les attributs des objets gérés de la RMIB (l'attribut `networkId` de l'objet `network1` vaut `"network-01"`).



(a) (b)

FIG. 10.14 – *ESTABLISH de la relation*

10.7.5 Autres opérations

Une fois établie, toutes les opérations de la relation sont potentiellement utilisables (figure 10.15.a), sauf `ESTABLISH`. Sur cette même figure, nous avons fait apparaître le nouveau menu des opérations de relation, on constate que l'opération `ESTABLISH` n'est plus activable.

La figure 10.15.b montre l'apparence de l'interface graphique après avoir effectué deux opérations `BIND`. L'opération `bind1` rajoute un participant de la classe `managedElement` et l'opération `bind2` fait de même dans un autre rôle pour la classe `interExangeCircuit`. Ces opérations sont toutes traduites par des opérations de création d'objet, à la différence de l'opération précédente, l'objet supérieur existe déjà (`network1`). La création des nouveau participant se fera donc quasi-automatiquement, seules les valeurs initiales de ces nouveaux participants peuvent être à définir par l'utilisateur (ou dans le formulaire de comportement de relation GRM+).

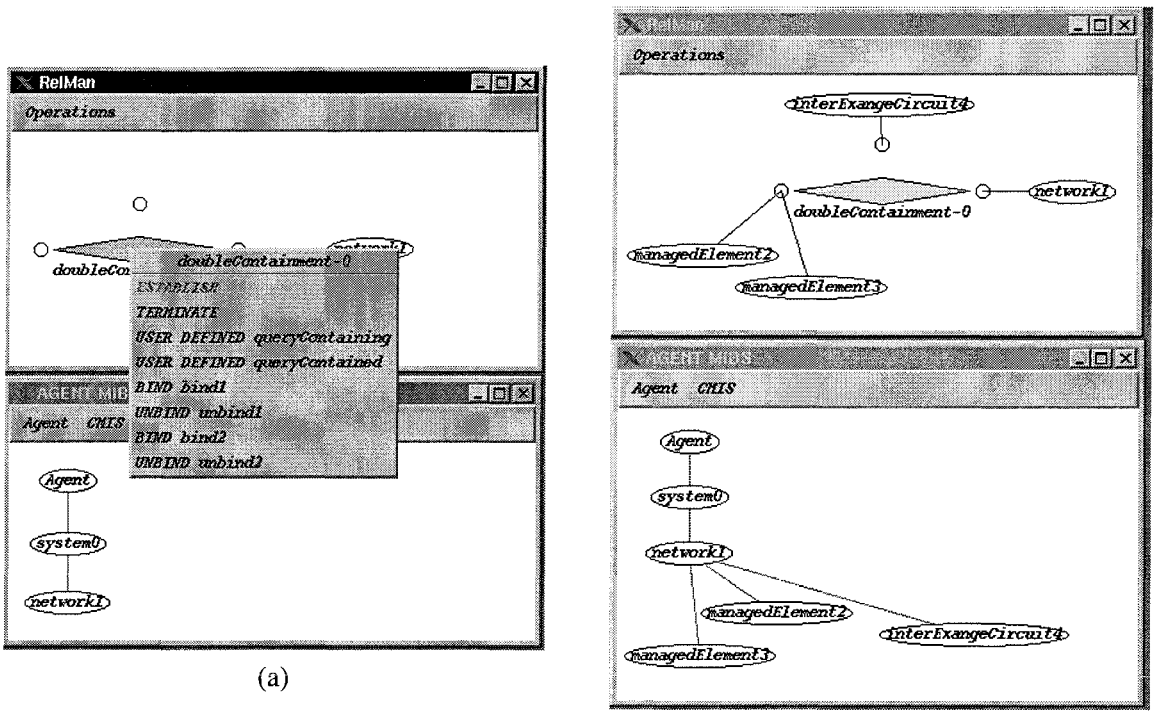


FIG. 10.15 – *Opérations*

10.8 Conclusions

Ce chapitre a présenté un prototype implantant des concepts définis dans le chapitre 8 pour l'architecture d'un gestionnaire de relation appelé RelMan. La représentation des relations avec corrélation de noms est implantée. Elles a été la première car ces corrélations sont un modèle de relation déjà existant dans le contexte des objets gérés et il était facile d'utiliser leurs fonctionnalités pour représenter une relation. Une fonctionnalité importante de ces corrélations est de maintenir automatiquement la relation. Les représentations avec attributs de relations nécessitent une gestion plus fine, où il faut maintenir des valeurs de DNs dans des objets pour conserver le sens de la relation. Elle est décrite dans [Jma98] et est en voie d'intégration avec le prototype existant. Les représentations de relation avec un objet de relation sont du même niveau de difficulté et les représentation avec des actions dépendent de leur implantation dans les objets gérés. Toutes les fonctionnalités décrites pour RelMan ne sont pas encore implantées. Notamment les fonctionnalités concernant la gestion des notifications de relation sont dépendantes de celles offertes par l'API `cmisOverJava` qui manque d'une vision côté gestionnaire pour les réceptions de notifications d'objets gérés.

Un intérêt de l'utilisation de Java pour RelMan est dans sa grande portabilité, vu l'hérogénéité des systèmes d'exploitation rencontrée dans une interconnexion de réseaux. Différentes implantations de RelMan peuvent être amenées à cohabiter, chacune sur un sous-réseau dont elles gèrent les relations internes. Des implantations de RelMan peuvent aussi être dédiées pour les relations entre sous-réseaux. Une application de gestion, écrite en Java et portant sur le réseau global, pourra distribuer ses traitements sur les gestionnaires de relations avec la transparence des caractéristiques de leurs implantations. Les développements autours de Java et de la gestion de réseaux, notamment dans la mobilité du code, vont dans de telles directions.

Quatrième partie

Conclusion

Chapitre 11

Conclusion Générale

Les relations entre les objets définis dans l'administration de réseaux ou les systèmes distribués posent des problèmes qui nécessitent d'être résolus. Différentes approches de tels systèmes d'informations ont intégré la notion de relation dans leurs caractéristiques. Dans cette thèse, nous avons contribué à améliorer l'utilisation d'une méthode de spécification issue du domaine de l'administration de réseaux. Les améliorations que nous avons apportées ont toujours été faites avec le soucis de les intégrer dans ce domaine. Le GRM offre une solution spécifique à la gestion OSI en permettant de spécifier des relations entre les objets de cette approche. Par rapport à des modèles de relations utilisés usuellement dans les bases de données, le GRM est plus adapté pour distinguer des ensembles d'objets ayant des dépendances liées à des buts communs, que pour établir des liens individuels entre couples d'objets. Sa puissance d'expression peut lui permettre de modéliser plusieurs autres modèles de relations, rendant ainsi possible une cohabitation de différents modèles de relations avec des mécanismes de passages entre les modèles.

Cependant, le GRM a certaines limites qui rendent son utilisation difficilement intégrable dans le domaine pour lequel il a été défini. Les relations contiennent une part importante de la sémantique des modèles de gestion et la validation de propriétés de ces systèmes doit les prendre en compte. De plus, le GRM souffre, comme les objets gérés, d'une absence de formalisme pour exprimer le comportement des relations. Enfin, la gestion OSI ne prévoit pas l'exploitation des relations du GRM.

11.1 Résumé des contributions

11.1.1 Valider le comportement global

Au vu des modèles de gestion existants, la tendance est vers une simplification du comportement des objets individuels et vers une plus grande complexité des dépendances entre ces objets. L'utilisation d'outils de validation permet d'acquerir des certitudes sur le comportement global des modèles de gestion, dont les objets sont contenus dans des MIBs.

Dans l'objectif de valider des propriétés liées aux relations entre les objets gérés d'un modèle complexe, nous avons proposé l'utilisation de plusieurs FDTs. L'utilisation de plusieurs méthodes permet de voir un même modèle de gestion sous des aspects différents, apportant chacun des éléments nouveaux par rapport aux autres. Il reste toujours un ensemble de ces éléments qui soit commun entre les approches. Le but de ces différentes vues est alors de traduire chaque élément apporté par une méthode formelle en des termes appartenant au modèle de gestion. Il ne s'agit pas de faire une nouvelle spécification du modèle de base (sauf s'il contient des erreurs trop importantes), mais de rajouter des contraintes qui n'avaient pas été spécifiées et qui ont été révélées au cours des spécifications formelles ou de leur validation.

Dans ces spécifications, nous avons proposé une séparation entre la spécification du comportement

des objets et celle de celui des relations. Ceci offre plusieurs avantages dans les spécifications formelles nécessaires à la validation. Tout d'abord nous avons observé sur le modèle de gestion étudié que cette sission permet, au niveau de l'analyse du comportement, de ne se concentrer que sur un type de comportement (objet ou relation) à la fois au lieu de vouloir embrasser la globalité de la sémantique des modèles de gestion. Il est apparu ensuite que cette séparation se traduit par des concepts généralement différents dans les méthodes formelles que nous avons utilisées. Les deux types de comportements peuvent aussi être fusionnés dans une méthode formelle. Il faut, dans de tels cas, bien identifier quel type de comportement l'on modélise.

11.1.2 Spécifier le comportement dans les relations

De par la nature informelle du comportement des objets et des relations de la gestion OSI, le travail précédent de spécification formelle dans un but de validation, dépend grandement des personnes qui lisent ces comportements. Une même explication en langage naturel peut être traduite différemment avec une même méthode formelle, entrainant des risques d'incompatibilité lorsque plusieurs personnes travaillent séparemment. La définition d'un langage de spécification formelle pour le comportement des modèles de gestion limite des problèmes tels que les contre-sens ou les ambiguïtés des spécifications informelles. Les travaux qui ont donné naissance à GDMO+ ont cet objectif. La syntaxe de ce langage est proche de GDMO. La spécification en formulaires ou l'utilisation directe des caractéristiques des objets GDMO donnent au spécifieur et au lecteur une vision homogène, ce qui facilite la compréhension des spécifications GDMO+. La sémantique de ce langage est définie informellement dans le document standardisé et il existe des travaux pour l'exprimer formellement (en Z).

Nous avons proposé GRM+ comme équivalent de GDMO+ pour les comportements de relations gérées. De ce fait, la syntaxe du GRM+ est proche de celle du GRM, notamment avec l'utilisation de formulaires pour les relations et leur représentation, mais aussi via l'expression de caractéristiques propres aux relations, comme les rôles. Le GRM utilise des concepts du modèle des objets gérés. Cela se traduit dans les spécifications de classes de relation par des références à des classes d'objets GDMO. Le comportement des relations influe généralement celui des objets reliés et il est normal d'intégrer dans GRM+ un certain nombre de concepts issus de GDMO+. La syntaxe des expressions de GDMO+, qui est concentrée sur les manipulations de types ASN.1 est entièrement reprise dans GRM+.

Le GRM+ est donc une extension de GDMO+ et fait l'objet d'un amendement à ce dernier. Nous avons aussi, au cours de ce travail, apporté quelques améliorations à GDMO+.

11.1.3 Exploiter les relations

Après les travaux d'analyses qui viennent d'être résumés et sur lesquels portent deux contributions de cette thèse, il nous a paru important de sortir l'utilisation du GRM et de son comportement du cadre de la spécification pour le porter au niveau de l'exploitation des relations. Les relations se sont avérées être utiles pour mieux spécifier les modèles de gestion, aussi devraient elles être utiles pour améliorer l'implantation de ces modèles et leur utilisation.

Pour cela, nous proposons RelMan dont l'architecture concrétise la notion d'instance de relation gérée. Nous avons explicité le besoin de maintenir la cohérence de ces instances et celui de les découvrir de manière dynamique, en parcourant les objets gérés existants. De plus, l'implantation de RelMan offre un ensemble de services pour manipuler les relations avec le même niveau de complexité que celui des manipulations d'objets. C'est à dire que ces manipulations (objets ou relations) se font par l'intermédiaire de services génériques qui sont disponibles pour des applications de gestion ou pour des utilisateurs humains. Deux types de services sont utilisables : des invocations d'opérations sur les relations gérées et

l'enregistrement à des notifications émises par ces relations. A la différence des services dédiés aux objets gérés, les services de relation ne sont pas des services de communication, il n'y a pas de protocole associé aux définitions des primitives de services. Toutefois la ressemblance entre les primitives du service de relation et des services de communication est voulue, tant pour permettre une éventuelle extension aux concepts de la gestion OSI (vue plus loin dans les perspectives) par un protocole véhiculant les requêtes de services, que pour homogénéiser les traitements dans des applications de gestion qui utilisent des services d'objet et des services de relation. Dans les deux catégories d'utilisation de RelMan (instance et service), le comportement des relations est pris en compte. Le maintien des relations fait intervenir des invariants de classes de relation et les invocations de services de relation mettent en jeu l'évaluation des pré- et des post-conditions.

Les concepts que nous avons proposés pour réaliser des gestionnaires de relations sont divisés en plusieurs parties qui traitent chacune un aspect spécifique de la gestion des relations. Les interactions qui soudent ces parties constituent une architecture de plate-forme de gestion. Chaque composant de RelMan a été défini de façon abstraite, par ses fonctionnalités, afin de donner la possibilité d'implanter RelMan avec des composants logiciels existants (base de données, pile de communication, ...).

Une implantation de RelMan, ainsi que celle d'un service CMIS en Java ont été réalisées afin de valider notre proposition.

11.1.4 Conclusions

Les contributions de cette thèse s'intègrent dans le domaine de la gestion de réseaux. Elles peuvent être utilisées dans différentes étapes de la conception de systèmes d'information distribués.

Lors des premières étapes de leur conception, où les fonctionnalités principales des systèmes sont définies, la validation de propriétés ou la simulation des systèmes sont d'une grande importance. Toute la suite de l'élaboration des systèmes est fondée sur ces premières étapes et une erreur peut se révéler d'autant plus coûteuse qu'elle est découverte tardivement.

Dans le cadre de la normalisation des spécifications d'objets et de relations, il est nécessaire de diffuser ces documents pour que des fabricants de ressources puissent s'y référer. Le choix d'un formalisme indépendant de toute méthode formelle autre que celle utilisée dans le domaine est important. C'est le but du GRM+. En effet, il offre une vision homogène des spécifications et laisse une totale liberté pour l'implantation de ces dépendances.

La dernière étape dans la conception du système d'information est son exploitation. D'une part l'utilisation des approches objets et relations dans les modèles de gestion permet de spécifier des traitements visant à assurer dynamiquement l'intégrité des données. Les problèmes de l'intégrité et du comportement des relations sont ceux d'une base de données distribuée. D'autre part, les applications de gestion reposent sur ces modèles de l'information et les relations leur offrent un moyen puissant pour faire de la gestion avec les objets. La navigation d'objets en objets ou la cohérence des manipulations sur plusieurs objets en sont des exemples. RelMan est en mesure d'offrir de tels contrôles et services spécialisés pour les relations.

11.2 Perspectives

11.2.1 Compléments de recherche

Une critique envers nos contributions est qu'elles ne sont pas reliées par des règles de transformation. Le GRM+ devrait pouvoir se traduire en spécifications d'une ou plusieurs méthodes formelles en vue

de la validation de propriétés du comportement. Le GRM+ devrait également pouvoir être compilé ou interpreté pour son exploitation. Nous avons proposé des ébauches d'une telle traduction. Celles-ci sont fournies dans [Nat98] mais ne sont pas finalisées à ce jour.

Une autre critique concerne le fait que nous ne nous sommes pas préoccupés des aspects relatifs au temps. Le besoin de spécification de contraintes temporelles dans les modèles de gestion n'est pas présent dans les modèles génériques étudiés mais peut apparaître dans plusieurs cas de figures. La réactivité du système de gestion se mesure depuis la ressource jusqu'à l'utilisateur, en passant par les objets gérés, l'agent, le service et le gestionnaire. Cette réactivité doit être connue quand des événements, survenus au niveau des ressources, doivent être signalés aux utilisateurs dans des délais spécifiés sous peine de ne pouvoir agir en conséquence. La réactivité est également importante dans l'autre sens, une action de gestion devant être appliquée aux ressources en un temps déterminé après son invocation. Au niveau des relations, leurs opérations étant traduites en opérations sur les objets, des contraintes à respecter sur la durée et l'ordonnancement de ces opérations peuvent être nécessaires pour garantir un état cohérent du système avec une certaine granularité du temps. Ces contraintes devraient être intégrées dans les spécifications. Elles ne le sont pas à ce jour.

Nous n'avons pas comparé la modélisation de relation du GRM avec celles qui existent dans les langages de programmation. Un état de l'art de ces solutions est proposé dans [NG95]. Notre interêt s'est porté vers des environnements distribués qui sont plus comparables au domaine de la gestion de réseaux que des langages indépendants de la distribution. L'évolution des applications distribuées va dans le sens d'une intégration de modèles de programmation centralisés avec la transparence de la distribution.

Au niveau des travaux d'intégration en gestion de réseaux, une contribution proposant un mapping GRM vers CORBA/IDL, en lien avec la proposition du JIDM[52], devrait être réalisée.

Le prototype RelMan et son environnement logiciel est en cours d'évolution. L'utilisation avec la plate-forme Open Master de Bull est partiellement réalisée et certains contrôles doivent encore être implantés.

11.2.2 Utilisations des contributions

De nombreux modèles de gestion sont encore à étudier pour établir des librairies de relations en GRM qui pourraient être proposées afin de compléter les standards existants. De plus en plus, différents modèles de gestion OSI vont être dépendants l'un de l'autre (PDH, SDH, ATM, IP, ...) et leurs interactions seront nombreuses et complexes. Les relations seront alors confrontées à des problèmes de modélisation et de performance qui jugeront de la valeur du modèle. Parallèlement à ces travaux, des applications de gestion utilisant les relations du GRM sont à dériver de celles se servant des relations d'une manière *ad hoc*. La spécification de librairies de relations peut aussi se faire suivant les aires fonctionnelles de la gestion et donc proposer des relations spécifiques aux applications relevant de ces aires. En résumé, un important travail de modélisation à l'aide du GRM est nécessaire pour sa promotion.

L'évolution des environnements de gestion vont vers une homogénéisation des concepts de gestionnaire et d'agent. Notamment lorsque des modèles de gestion cohabitent et que les gestionnaires d'un modèle sont aussi les agents pour d'autres gestionnaires d'autres modèles. Il nous semble important d'adapter le gestionnaire de relation pour qu'il devienne un agent de relation. L'apparition d'un premier amendement à la fonction de gestion OSI de partage de l'information [10198] s'intègre bien à cette perspective. Cet amendement consiste en une définition de classes d'objets GDMO qui modélisent les spécifications du GRM. Il se rajoute à la définition de classes GDMO modélisant les spécifications GDMO. Ce méta-modèle est une brique de base importante de RelMan. La structure de la RMIB peut être

52. Join Inter Domains Management [SH97]

traduite en objets d'une MIB et le service de relation être converti en action sur une classe particulière aux instances de laquelle les applications gérant les relations peuvent s'adresser. Des règles formelles de ces transformations sont encore à établir.

Une dernière perspective concerne la mobilité dans la gestion de réseaux. Un gestionnaire de relation pourrait bénéficier de la mobilité du code afin que des applications puissent parcourir le réseaux à la recherche de nouvelles instances de relations ou veiller au maintien de l'intégrité des relations. On peut aussi imaginer que des objets gérés soient mobiles et que par conséquent les instances de relations le soient de même. De nouveaux services et mécanismes sont à prévoir dans RelMan pour accepter dynamiquement de nouvelles instances de relation, dont la classe et la représentation ne sont pas forcément connues.

ANNEXES

Annexe A

Classes et comportements des relation de l'exemple

A.1 Diagramme des classes génériques

La figure A.1 montre le schéma d'héritage entre les classes de relations du GRM qui ont été traitées dans l'exemple du chapitre 4. La signification des symboles est la même que pour la figure 5.1 du chapitre 5.

A.2 Définition de la classe `traffic`

A.2.1 Classe GRM `traffic`

```
traffic RELATIONSHIP CLASS
  BEHAVIOUR trafficBehaviour;
  SUPPORTS ESTABLISH, TERMINATE,
           QUERY querySource, QUERY queryDestination
           QUERY queryOpState, QUERY queryAdState,
           USER-DEFINED lockTraffic,
           USER-DEFINED unlockTraffic;
      QUALIFIED BY "ITU-T X.721: 1992":administrativeState,
                   "ITU-T X.721: 1992":operationalState;
  ROLE source
    COMPATIBLE-WITH "ITU-T M.3100 1995" terminationPoint
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1
   PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 0..1
 ROLE destination
   COMPATIBLE-WITH "ITU-T M.3100 1995" terminationPoint
   PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..N
   PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 0..1;
REGISTERED AS {1};
```

A.2.2 Comportement GRM+ de la classe `traffic`

```
trafficBehaviour RELATIONSHIP CLASS BEHAVIOUR

  COMMENT

  ``La classe traffic modélise une relation entre des points de
  terminaison. Les participants ne peuvent être que dans un rôle à la
  fois.    L'état opérationnel de    la relation dépend des états
  opérationnel des participants de chaque rôle.

  L'établissement de la relation se fait avec un participant dans
```

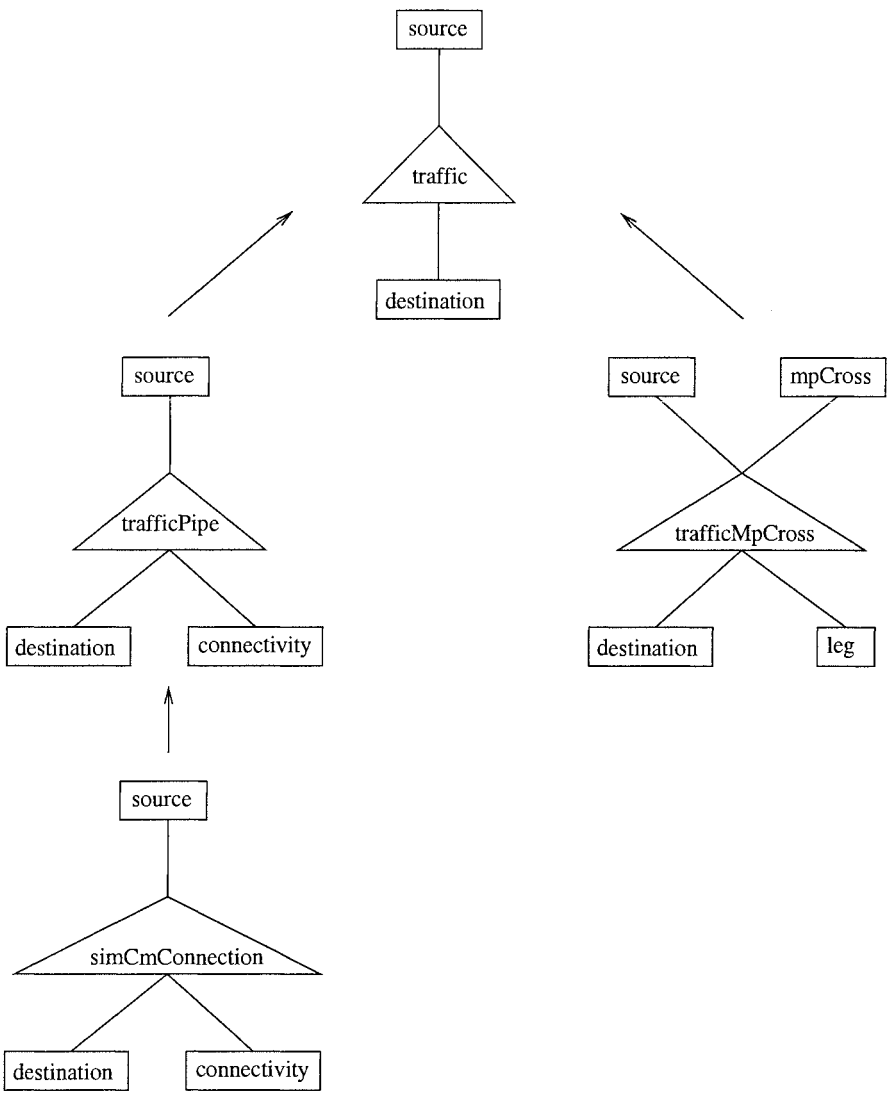


FIG. A.1 – *Diagramme des classes de relations*

```
chaque rôle et  la relation est établie dans  un état opérationnel.
Les participants doivent  être  opérationnels à  la  suite de cette
opération.

L'opération  lockTraffic  bloque administrativement la  relation et
ses participant.

L'opération unlockTraffic débloque  la relation et les participants
si la relation était au préalable bloquée administrativement.''

INVARIANTS
  FORALL x IN role(destination) |
       x NOT IN role(source),
  IF (package(operationalStatePackage, source) AND
      package(operationalStatePackage, destination))
  THEN
    (FORALL x,y IN role(source),role(destination) |
       (x -> operationalState = ``disabled'' AND
       y -> operationalState = ``disabled''))
      <=> operationalState = ``disabled''
 ENDIF;

OPERATIONS

   ESTABLISH
     WITH-INFO SEQUENCE {source ObjectInstance,
                         dest   ObjectInstance}
     WITH
       establishInfo.source IN role(source) AND
       establishInfo.dest IN role(destination),
       operationalState = ``enabled'',
       IF package(operationalState, source) THEN
              FORALL x IN role(source) |
                x -> operationalState = ``enabled''
       ENDIF AND
       IF package(operationalState, destination) THEN
           FORALL x IN role(destination) |
            x -> operationalState = ``enabled''
       ENDIF;

   USER-DEFINED lockTraffic
     WITH
       administrativeState = "locked",
       IF package(administrativeState, source) THEN
         FORALL x IN role(source) |
           x -> administrativeState = "locked"
           ENDIF AND
       IF package(administrativeState, destination) THEN
         FORALL x IN role(destination) |
           y -> administrativeState = "locked"
       ENDIF;

   USER-DEFINED unlockTraffic
     FOR
       administrativeState = "locked";
     WITH
       administrativeState = "unlocked",
       IF package(administrativeState, source) THEN
         FORALL x IN role(source) |
            x -> administrativeState = "unlocked"
       ENDIF AND
       IF package(administrativeState, destination) THEN
         FORALL x IN role(destination) |
             x -> administrativeState = "unlocked"
       ENDIF
;
```

A.3 Définition de la classe `trafficPipe`

A.3.1 Classe GRM `trafficPipe`

```
trafficPipe RELATIONSHIP CLASS
   BEHAVIOUR trafficPipeBehaviour;
   DERIVED FROM traffic;
   SUPPORTS QUERY queryDirectionality;
   QUALIFIED BY "ITU-T M.3100: 1995":directionality;
   ROLE connectivity
     COMPATIBLE-WITH "ITU-T M.3100: 1995":pipe
     PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1
     PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 1..1
  ROLE destination
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1;
REGISTERED AS {2};
```

A.3.2 Comportement GRM+ de la classe `trafficPipe`

```
trafficPipeBehaviour RELATIONSHIP CLASS BEHAVIOUR
  COMMENT

  ''La relation  trafficPipe relie  deux  participants qui  sont des
  points de  terminaison  d'un canal  de communication.  L'information
  caractéristique doit être la même dans chaque participant.

  Les attributs fromTermination et  toTermination  de l'objet dans  le
  rôle  connectivity  réfèrent  aux objets  dans  les  rôles source et
  destination.

  L'état opérationnel de la relation dépend de celui des participants
  aux rôles source et destination.

  Par rapport à  la  spécification de  la  M3100,  il est rajouté  le
  comportement suivant :

  Si l'état administratif de la  relation est locked, alors cela  doit
  être  de même  pour celui  des  participant aux  rôles source  et
  destination.''

  INVARIANTS
   IF package(characteristicInformation, source) AND
      package(characteristicInformation, destination) THEN
         FORALL x,y IN role(source), role(destination) |
               x -> characteristicInformation =
                           y -> characteristicInformation
   ENDIF,
   FORALL x IN role(connectivity) |
     x -> fromTermination IN role(source) AND
     x -> toTermination IN role(destination),
   IF package(operationalState, source) THEN
        FORALL x IN role(source) |
          IF x -> operationalState = disabled THEN
             operationalState = disabled
          ENDIF
    ENDIF,
    IF package(operationalState, destination) THEN
        FORALL x IN role(destination) THEN
          IF x -> operationalState = disabled THEN
             operationalState = disabled
          ENDIF
    ENDIF,
    IF package(administrativeState, source) AND
       package(administrativeState, destination) THEN
        IF administrativeState = locked THEN
           FORALL x, y IN role(source), role(destination) |
             x -> administrativeState = unlocked AND
             y -> administrativeState = unlocked
        ENDIF
    ENDIF;
```

A.4 Définition de la classe `trafficMpCross`

A.4.1 Classe GRM `trafficMpCross`

```
trafficMpCross RELATIONSHIP CLASS
  BEHAVIOUR trafficMpCrossBehaviour;
  DERIVED FROM traffic;
  SUPPORT QUERY queryAvState;
  QUALIFIED BY  "ITU-T X.721: 1992":availabilityStatus;
  ROLE mpCross
    COMPATIBLE-WITH "ITU-T M.3100: 1995":mpCrossConnection
    PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..1
    PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 1..1
 ROLE leg
   COMPATIBLE-WITH "ITU-T M.3100: 1995":crossConnection
   PERMITTED-ROLE-CARDINALITY-CONSTRAINT 1..N
   PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT 0..1
 ROLE destination
   BIND-SUPPORT
   UNBIND-SUPPORT;
REGISTERED AS {3};
```

A.4.2 Comportement GRM+ de la classe `trafficMpCross`

```
trafficMpCrossBehaviour RELATIONSHIP CLASS BEHAVIOUR

  COMMENT

  ''La relation  trafficMpCross  relie des participants  à un brassage
  multipoint au sein d'un élément  géré.  Il  doit  y avoir autant  de
  participants  dans les rôles leg  et destination.  Si un participant
  dans le rôle leg n'est pas  opérationnel, alors la relation est dans
  un  état   dégradé.  Si  tous   les  participants au  rôle  leg sont
  inopérationnel,   alors  la   relation  est   inopérationnelle.  Le
  participant au  rôle mpCross doit  référencer le participant au rôle
  source avec son attribut fromTermination.    Le participant au  rôle
  source doit  référencer   le participant au  rôle  mpCross  avec son
  attribut crossConnectionObjectPointer.  Les participants au rôle leg
  doivent être dans un état unidirectionnel et ne pas désigner d'objet
  avec leur attribut fromTermination.  Chaque participant dans le rôle
  leg doit  référencer un unique participant  dans le rôle destination
  avec  son attribut    toTermination et chaque  participant  au  rôle
  destination doit référencer un  unique participant au rôle leg  avec
  son  attribut  crossConnectionObjectPointer.   Les participant   aux
  rôles leg et destination  doivent  pouvoir être associés par  paires
  suivant les valeurs de ces attributs

  Après l'établissement de la  relation, les participants au  rôle leg
  sont opérationnels.

  L'ajoût d'un participant au rôle destination ne peut se faire que si
  la relation est débloquée et opérationnelle et que le participant au
  rôle source soit opérationnel.

  La suppression d'un participant au rôle source  ne peut se faire que
  si la relation est débloquée et opérationnelle.''

  INVARIANTS
   #role(leg) = #role(destination),

   (FORALL x IN role(leg) | x -> operationalState = ''enabled'')
   OR ''degraded'' IN availabilityStatus,

   (THEREIS x IN role(leg) | x -> operationalState = ''enabled'')
   OR operationalState = ''disabled'',

   (FORALL x IN role(mpCross) |
       x -> fromTermination IN role(source))
   AND
   FORALL x IN role(source) |
```

```
      if package(crossConnectionPointerPackage, source) THEN
        x ->  crossConnectionObjectPointer IN role(mpCross),
   FORALL x IN role(leg) |
       x -> fromTermination = NULL AND
       x -> directionality = "unidirectional",
    (FORALL x IN role(leg) |
      THEREIS y IN role(destination) |
       x -> toTermination = y)
    AND
    FORALL x IN role(destination) |
     THEREIS y IN role(leg) |
      (y -> toTermination = x AND
       IF  package(crossConnectionPointerPackage, destination) THEN
        x ->  crossConnectionObjectPointer = y
       ENDIF AND
       FORALL z IN role(leg) - {y} |
        z -> toTermination <> x),
   IF package(crossConnectionPointerPackage, source) THEN
      FORALL x IN role(source) |
        FORALL y IN role(mpCross) |
         x -> crossConnectionObjectPointer = y
    ENDIF;

   OPERATIONS

     ESTABLISH
       WITH
         FORALL x IN role(leg) `|
           x -> operationalState = ``enabled'';
     BIND
       WITH-INFO SEQUENCE{dest ObjectInstance}
       FOR
         bindInfo().dest NOT IN role(destination) AND
         administrativeState = ``unlocked'' AND
         operationalState = ``enabled'' AND
         IF package(operationalState, source) THEN
           FORALL x IN role(source) |
                  x -> operationalState = ``enabled''
         ENDIF;
       WITH
         bindInfo().dest IN role(destination);
     UNBIND
       WITH-INFO SEQUENCE{dest ObjectInstance}
       FOR
         bindInfo().dest IN role(destination) AND
         administrativeState = ``unlocked'' AND
         operationalState = ``enabled'';
       WITH
        bindInfo().dest NOT IN role(destination);
;
```

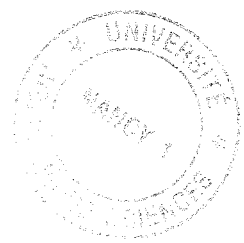
A.5 Définition de la représentation de la classe `trafficPipe`

A.5.1 Représentation GRM `connectionTrafficPipe`

```
connectionTrafficPipe RELATIONSHIP MAPPING
  BEHAVIOUR connectionTrafficPipeBehaviour;
  RELATIONSHIP CLASS trafficPipe;

  ROLE source RELATED-CLASSES
        "ITU-T M.3100: 1995":connectionTerminationPointSource
    REPRESENTED-BY ATTRIBUTE "ITU-T M.3100: 1995":a-TPInstance;

  ROLE connectivity RELATED-CLASSES
       "ITU-T M.3100: 1995":connection
    QUALIFY "ITU-T X.721: 1992":administrativeState
            "ITU-T X.721: 1992":operationalState
            "ITU-T M.3100: 1995":directionality;
```



```
   ROLE destination RELATED-CLASSES
        "ITU-T M.3100: 1995":connectionTerminationPointSink
     REPRESENTED-BY ATTRIBUTE "ITU-T M.3100: 1995":z-TPInstance;

  OPERATIONS MAPPING

    ESTABLISH MAPS-TO-OPERATION CREATE OF connectivity,
    TERMINATE MAPS-TO-OPERATION DELETE OF connectivity;
    QUERY querySource MAPS-TO-OPERATION
       GET "ITU-T M.3100: 1995":a-TPInstance OF connectivity;
    QUERY queryDestination MAPS-TO-OPERATION
       GET "ITU-T M.3100: 1995":z-TPInstance OF connectivity;
    QUERY queryAdState MAPS-TO-OPERATION
       GET "ITU-T X.721: 1992":administrativeState OF connectivity;
    QUERY queryOpState MAPS-TO-OPERATION
       GET "ITU-T X.721: 1992":operationalState OF connectivity;
    QUERY queryDirectionality MAPS-TO-OPERATION
       GET "ITU-T M.3100: 1995":directionality OF connectivity;
    USER-DEFINED lockTraffic MAPS-TO-OPERATION
       REPLACE "ITU-T X.721: 1992":administrativeState OF connectivity;
    USER-DEFINED unlockTraffic MAPS-TO-OPERATION
       REPLACE "ITU-T X.721: 1992":administrativeState OF connectivity;
REGISTERED AS {2.1};
```

A.5.2 Comportement GRM+ de la représentation `connectionTrafficPipe`

```
connectionTrafficPipeBehaviour RELATIONSHIP MAPPING BEHAVIOUR

  COMMENT

  ''Cette représentation de la  classe  trafficPipe relie un point  de
  terminaison de connexion collecteur (sink) dans le rôle source et un
  point de terminaison de   connexion source dans le  rôle destination
  (les termes sink et source sont relatifs à  un élément géré et non à
  la connexion).

   L'attribut directionality      de  la   relation     doit    être
   unidirectional car le traffic va de l'objet dans le rôle source à
   l'objet dans le rôle destination.'';

  INVARIANTS

     directionality = unidirectional;

  OPERATIONS-MAPPING

    ESTABLISH WITH-INFO SEQUENCE {supOid ObjectInstance,
                                  source ObjectInstance,
                                  dest   ObjectInstance}
      MAPS-TO
       CREATE OF connectivity
        WITH-SERVICE-INFO
         mappingCreateInfo().superiorObjectInstance :=
               establishInfo().supOid;
         mappingCreateInfo().attributeList :=
               [[fromTermination, establishInfo().source],
                [toTermination, establishInfo().dest],
                [administrativeState, "unlocked"],
                [operationalState, "enabled"],
                [availabilityStatus, {}]];

    USER-DEFINED lockTraffic
       MAPS-TO
         REPLACE OF connectivity
          WITH-SERVICE-INFO
              mappingSet().scope := baseLevelOnly,
              mappingSet().modificationList :=
                 [[modifyOperator, REPLACE],
                  [administrativeState, locked]];
```

```
    USER-DEFINED unlockTraffic
       MAPS-TO
         REPLACE OF connectivity
          WITH-SERVICE-INFO
              mappingSet().scope := "baseLevelOnly",
              mappingSet().modificationList :=
                 [[modifyOperator, "REPLACE"],
                  [administrativeState, "unlocked"]];

  ;
```

A.6 Définition de la représentation de la classe `trafficMpCross`

A.6.1 Représentation GRM `connectionTrafficMpCross`

```
connectionTrafficMpCross RELATIONSHIP MAPPING
  BEHAVIOUR connectionTrafficMpCrossBehaviour;
  RELATIONSHIP CLASS trafficMpCross;

   ROLE source RELATED-CLASSES
            "ITU-T M.3100: 1995":connectionTerminationPointSink
     REPRESENTED-BY ATTRIBUTE
              "ITU-T M.3100: 1995":fromTermination;

   ROLE mpCross RELATED CLASSES
            "ITU-T M.3100: 1995":mpCrossConnection
    REPRESENTED-BY NAMING
              "ITU-T M.3100: 1995":crossConnection-mpCrossConnection
             USING SUPERIOR
    QUALIFY "ITU-T X.721: 1992":administrativeState
            "ITU-T X.721: 1992":operationalState
            "ITU-T X.721: 1992":availabilityStatus;

  ROLE leg RELATED CLASSES "ITU-T M.3100: 1995":crossConnection
    REPRESENTED-BY NAMING
              "ITU-T M.3100: 1995":crossConnection-mpCrossConnection
              USING SUBORDINATE;

  ROLE destination RELATED-CLASSES
           "ITU-T M.3100: 1995":connectionTerminationPointSource
    REPRESENTED-BY ATTRIBUTE
              "ITU-T M.3100: 1995":toTermination;

  OPERATIONS MAPPING

    ESTABLISH MAPS-TO-OPERATION CREATE OF mpCross,
                                CREATE OF leg;
    TERMINATE MAPS-TO-OPERATION DELETE OF mpCross;
    QUERY querySource MAPS-TO-OPERATION
          GET "ITU-T M.3100: 1995":fromTermination OF mpCross;
    QUERY queryDestination MAPS-TO-OPERATION
          GET "ITU-T M.3100: 1995":toTermination OF leg;
    QUERY queryAdState MAPS-TO-OPERATION
          GET "ITU-T X.721: 1992":administrativeState of mpCross;
    QUERY queryOpState MAPS-TO-OPERATION
          GET "ITU-T X.721: 1992":operationalState of mpCross;
    QUERY queryAvState MAPS-TO-OPERATION
          GET "ITU-T X.721: 1992":availabilityStatus of mpCross;
    USER-DEFINED lockTraffic MAPS-TO-OPERATION
          REPLACE "ITU-T X.721: 1992":administrativeState of mpCross;
    USER-DEFINED unlockTraffic MAPS-TO-OPERATION
          REPLACE "ITU-T X.721: 1992":administrativeState of mpCross;
    BIND MAPS-TO-OPERATION CREATE OF leg;
    UNBIND MAPS-TO-OPERATION DELETE OF leg;

REGISTERED AS {3.1}
```

A.6.2 Comportement GRM+ de la représentation `connectionTrafficMpCross`

```
connexionTrafficMpCrossBehaviour RELATIONSHIP MAPPING BEHAVIOUR

  COMMENT

  ''Cette représentation de la relation trafficMpCross instancie un
  brassage multipoint entre deux extrémités de connexions. Les
  participants aux rôles source et destination doivent se référencer
  mutuellement avec leur attribut, respectivement,
  dowstreamConnectivityPointer et upstreamConnectivityPointer"

  INVARIANTS
    FORALL x,y IN role(source), role(destination) |
      y -> upstreamConnectivityPointer = x AND
      x -> dowstreamConnectivityPointer = role(destination);

  OPERATIONS-MAPPING

    ESTABLISH WITH-INFO SEQUENCE {supOid ObjectInstance,
                                  source ObjectInstance,
                                  dest   ObjectInstance}
     MAPS-TO
      CREATE OF mpCross
       WITH-SERVICE-INFO
        mappingCreateInfo().superiorObjectInstance :=
              establishInfo().supOid;
        mappingCreateInfo().attributeList :=
              [[fromTermination, establishInfo().source],
               [administrativeState, "unlocked"],
               [operationalState, "enabled"],
               [availabilityStatus, {}]],
      CREATE OF leg
       WITH-SERVICE-INFO
          mappingCreateInfo().attributeList :=
                      [[fromTermination, "NULL"],
                       [toTermination, establishInfo().dest],
                       [directionality, "unidirectional"],
                       [administrativeState, "unlocked"],
                       [operationalState, "enabled"]];

   BIND WITH-INFO SEQUENCE {dest   ObjectInstance}
     MAPS-TO
      CREATE OF leg
       WITH-SERVICE-INFO
          mappingCreateInfo().attributeList :=
                      [[fromTermination, "NULL"],
                       [toTermination, establishInfo().dest],
                       [directionality, "unidirectional"],
                       [administrativeState, "unlocked"],
                       [operationalState, "enabled"]];

   UNBIND WITH-INFO SEQUENCE {dest ObjectInstance}
     MAPS-TO
       DELETE OF leg
        WITH-SERVICE-INFO
          mappingDeleteInfo().baseObjectInstance := x;
        FOR
          x IN role(leg) AND
          x -> toTermination = mappingDeleteInfo().dest;

   USER-DEFINED lockTraffic
      MAPS-TO
        REPLACE OF mpCross
         WITH-SERVICE-INFO
            mappingSet().scope := "baseLevelOnly",
            mappingSet().modificationList :=
               [[modifyOperator, "REPLACE"],
                [administrativeState, "locked"]];

   USER-DEFINED unlockTraffic
      MAPS-TO
```

```
          REPLACE OF mpCross
           WITH-SERVICE-INFO
               mappingSet().scope := "baseLevelOnly",
               mappingSet().modificationList :=
                  [[modifyOperator, "REPLACE"],
                   [administrativeState, "unlocked"]];
;
```

Annexe B

Relation et comportement du SIM-CM

B.1 Définition de la classe simcmCircuitTrafficPipe

B.1.1 Classe GRM simcmCircuitTrafficPipe

```
simcmCircuitTrafficPipe RELATIONSHIP CLASS

  BEHAVIOUR simcmCircuitTrafficPipeBehaviour;
  DERIVED FROM trafficPipe;
  SUPPORTS USER-DEFINED lockSource,
           USER-DEFINED unlockSource;
  QUALIFIED BY "ITU-T X.721: 1992": availabilityStatus;

  ROLE source
     COMPATIBLE-WITH
   "ITU-T M.3100: 1995":connectionTerminationPointBidirectional
  ROLE connectivity
     COMPATIBLE-WITH
      "SIM-CM 1994":interExchangeCircuit
  ROLE destination
     COMPATIBLE-WITH
   "ITU-T M.3100: 1995":connectionTerminationPointBidirectional;
REGISTERED AS {4}
```

B.1.2 Comportement GRM+ de la classe simcmCircuitTrafficPipe

```
simcmCircuitTrafficPipeBehaviour RELATIONSHIP CLASS BEHAVIOUR

 OPERATIONS
  USER-DEFINED lockSource
    FOR
      package(administrativePackage, source) AND
      FORALL x IN role(source) |
              x -> administrativeState = ''unlocked'';
    WITH
        (FORALL x IN role(destination) |
                x -> operationalState = ''disabled'' AND
                x -> availabilityStatus =
                  OLD(y -> availabilityStatus) + { ''dependency''})
        AND operationalState = ''disabled''
        AND availabilityStatus =
                  OLD(availabilityStatus) + { ''dependency''};
   USER-DEFINED unlockSource
     FOR
      package(administrativePackage, source) AND
      FORALL x IN role(source) |
              x -> administrativeState = ''locked'';
    WITH
        (FORALL x IN role(destination) |
             x -> availabilityStatus =
```

```
            OLD(y -> availabilityStatus) - { ``dependency''})
        AND availabilityStatus =
                OLD(availabilityStatus) - { ``dependency''};
;
```

Annexe C

MACRO définies pour GRM+

C.1 MACRO pour obtenir les participants d'un rôle

```
role MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  VARIABLES _role_ OBJECT IDENTIFIER
  MACRO-TYPE GRMplusModule.Role;
  DEFINITION TEXTUAL
    "Cette macro prend l'identificateur d'un rôle dans la relation
    courante et retourne l'ensemble des instances d'objets gérés qui
    participent au rôle _role_ de la relation.";
```

C.2 MACRO pour la présence de modules conditionnels

```
package MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  VARIABLES _object_ CMIP-1.ObjectInstance,
            _package_ OBJECT IDENTIFIER;
  MACRO-TYPE BOOLEAN;
  DEFINITION TEXTUAL
    "Cette macro prend un DN et une référence de package comme
    argument et retourne vrai si _package_ est instancié dans l'objet
    géré _object_.";
```

C.3 MACROs pour les opérations de relation

C.3.1 MACRO pour l'opération ESTABLISH

```
establishInfo MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.estasblishInfo;
  DEFINITION TEXTUAL
    "Cette macro référe la structure qui est définie comme l'argument
    d'appel (WITH-INFO) de l'opération ESTABLISH.";

establishResult MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.estasblishResult;
  DEFINITION TEXTUAL
    "Cette macro référe la structure qui est définie comme l'argument de
    retour (WITH-REPLY) de l'opération ESTABLISH.";
```

C.3.2 MACRO pour l'opération TERMINATE

```
terminateInfo MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.terminateInfo;
  DEFINITION TEXTUAL
    "Cette  macro référe la structure qui   est définie comme l'argument
    d'appel (WITH-INFO) de l'opération TERMINATE.";

terminateResult MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.terminateResult;
  DEFINITION TEXTUAL
    "Cette macro référe la structure qui  est définie comme l'argument de
    retour (WITH-REPLY) de l'opération TERMINATE.";
```

C.3.3 MACRO pour l'opération NOTIFY

```
notifyInfo MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.notifyInfo;
  DEFINITION TEXTUAL
    "Cette macro référe la structure qui est définie comme l'argument
    d'appel (WITH-INFO) de l'opération NOTIFY.";

notifyResult MACRO
 COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.notifyResult;
  DEFINITION TEXTUAL
    "Cette macro référe la structure qui est définie comme l'argument
    de retour (WITH-REPLY) de l'opération NOTIFY.";
```

C.3.4 MACRO pour l'opération USER-DEFINED

```
user-definedInfo MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.userd-definedInfo;
  DEFINITION TEXTUAL
    "Cette macro référe la structure qui est définie comme l'argument
    d'appel (WITH-INFO) de l'opération USER-DEFINED.";

user-definedResult MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.userd-definedResult;
  DEFINITION TEXTUAL
    "Cette macro référe la structure qui est définie comme l'argument
    de retour (WITH-REPLY) de l'opération USER-DEFINED.";
```

C.3.5 MACRO pour l'opération BIND

```
bindInfo MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.bindInfo;
  DEFINITION TEXTUAL
    "Cette macro référe la structure qui est définie comme l'argument
    d'appel (WITH-INFO) de l'opération BIND.";

bindResult MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.bindResult;
```

```
  DEFINITION TEXTUAL
    "Cette macro réfère la structure qui est définie comme l'argument
    de retour (WITH-REPLY) de l'opération BIND.";
```

C.3.6 MACRO pour l'opération UNBIND

```
unbindInfo MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.unbindInfo;
  DEFINITION TEXTUAL
    "Cette macro réfère la structure qui est définie comme l'argument
    d'appel (WITH-INFO) de l'opération UNBIND.";

unbindResult MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.unbindResult;
  DEFINITION TEXTUAL
    "Cette macro réfère la structure qui est définie comme l'argument
    de retour (WITH-REPLY) de l'opération BIND.";
```

C.3.7 MACRO pour l'opération QUERY

```
queryInfo MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.queryInfo;
 DEFINTION TEXTUAL
    "Cette macro réfère la structure qui est définie comme l'argument
    d'appel (WITH-INFO) de l'opération QUERY.";

queryResult MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les spécifications GRM+.";
  MACRO-TYPE GRMplusModule.queryResult;
  DEFINITION TEXTUAL
    "Cette macro réfère la structure qui est définie comme l'argument
    de retour (WITH-REPLY) de l'opération QUERY.";
```

C.4 MACROs pour les opérations système

C.4.1 MACRO pour l'opération ACTION

```
mappingAction MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les formulaires de
    comportement de représentation de relation, dans la clause
    WITH-SERVICE-PARAMETER, quand l'opération de relation est traduite
    avec une ACTION.";
  MACRO-TYPE CMIP-1.ActionArgument;
  DEFINITION TEXTUAL
    "Cette macro réfère la structure qui contient les informations
    transportées par le service CMIS M-ACTION.";
```

C.4.2 MACRO pour l'opération CREATE

```
mappingCreate MACRO
  COMMENT
    "Cette macro ne peut être utilisée que dans les formulaires de
    comportement de représentation de relation, dans la clause
    WITH-SERVICE-PARAMETER, quand l'opération de relation est traduite
    avec un CREATE.";
  MACRO-TYPE CMIP-1.CreateArgument;
  DEFINITION TEXTUAL
```

```
    "Cette   macro  référe la  structure   qui contient les informations
    transportées par le service CMIS M-CREATE.";
```

C.4.3 MACRO pour l'opération DELETE

```
mappingDelete MACRO
  COMMENT
    "Cette   macro ne  peut  être  utilisée que dans   les formulaires de
    comportement  de   représentation   de   relation,   dans  la  clause
    WITH-SERVICE-PARAMETER,   quand l'opération de  relation est traduite
    avec un DELETE.";
  MACRO-TYPE CMIP-1.DeleteArgument;
  DEFINITION TEXTUAL
    "Cette macro   référe  la structure   qui  contient les  informations
    transportées par le service CMIS M-DELETE.";
```

C.4.4 MACRO pour l'opération GET

```
mappingGet MACRO
  COMMENT
    "Cette macro  ne peut  être   utilisée que  dans les  formulaires  de
    comportement  de   représentation  de   relation,   dans   la  clause
    WITH-SERVICE-PARAMETER, quand   l'opération de relation  est traduite
    avec un GET.";
  MACRO-TYPE CMIP-1.GetArgument;
  DEFINITION TEXTUAL
    "Cette  macro référe  la   structure qui  contient les   informations
    transportées par le service CMIS M-GET.";
```

C.4.5 MACRO pour l'opération SET

```
mappingSet MACRO
  COMMENT
    "Cette macro    ne peut être utilisée  que   dans les  formulaires de
    comportement    de représentation   de  relation,    dans la clause
    WITH-SERVICE-PARAMETER,  quand  l'opération de  relation est traduite
    avec un SET (i.e. ADD, REPLACE ou REMOVE).";
  MACRO-TYPE CMIP-1.SetArgument;
  DEFINITION TEXTUAL
    "Cette macro référe   la   structure qui contient  les   informations
    transportées par le service CMIS M-SET.";
```

C.4.6 MACRO pour l'opération NOTIFICATION

```
mappingNotification MACRO
  COMMENT
    "Cette macro   ne  peut être   utilisée que dans   les formulaires de
    comportement  de    représentation de   relation,    dans  la   clause
    WITH-SERVICE-PARAMETER, quand  l'opération  de relation est  traduite
    avec une NOTIFICATION.";
  MACRO-TYPE CMIP-1.EventReportArgument;
  DEFINITION TEXTUAL
    "Cette macro   référe   la structure  qui  contient  les  informations
    transportées par le service CMIS M-EVENT-REPORT.";
```

C.5 Module ASN.1

```
GRMplusModule
DEFINITIONS ::= BEGIN
IMPORTS
ObjectInstance
FROM CMIP-1
Role ::= SET OF ObjectInstance
establishInfo ::= ANY       -- Le contexte définit le type
```

```
establishResult ::= ANY     -- Le contexte définit le type
terminateInfo ::= ANY       -- Le contexte définit le type
terminateResult ::= ANY     -- Le contexte définit le type
bindInfo ::= ANY            -- Le contexte définit le type
bindResult ::= ANY          -- Le contexte définit le type
unbindInfo ::= ANY          -- Le contexte définit le type
unbindResult ::= ANY        -- Le contexte définit le type
queryInfo ::= ANY           -- Le contexte définit le type
queryResult ::= ANY         -- Le contexte définit le type
notifyInfo ::= ANY          -- Le contexte définit le type
notifyResult ::= ANY        -- Le contexte définit le type
user-definedInfo ::= ANY    -- Le contexte définit le type
user-definedResult ::= ANY -- Le contexte définit le type
END
```

Annexe D

La classe et la représentation de relation de double contenance

D.1 Classe GRM doubleContainment

```
doubleContainment                RELATIONSHIP CLASS
        BEHAVIOUR containmentBehaviour DEFINED AS
            BEHAVIOUR
               ``L'objet dans le rôle containing contient les objets
               dans les rôles contained1 et contained2.'';;

        SUPPORTS
                USER DEFINED queryContaining,
                USER DEFINED queryContained,
                TERMINATE ,
                ESTABLISH ;
        ROLE contained1
                PERMITTED-ROLE-CARDINALITY-CONSTRAINT Grm.card0-N
                REQUIRED-ROLE-CARDINALITY-CONSTRAINT Grm.card0-1
                BIND-SUPPORT bind1
                UNBIND-SUPPORT unbind1
                PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT Grm.card1-1

        ROLE contained2
                PERMITTED-ROLE-CARDINALITY-CONSTRAINT Grm.card0-N
                REQUIRED-ROLE-CARDINALITY-CONSTRAINT Grm.card0-1
                BIND-SUPPORT bind2
                UNBIND-SUPPORT unbind2
                PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT Grm.card1-1

        ROLE containing
                PERMITTED-ROLE-CARDINALITY-CONSTRAINT Grm.card1-1
                REQUIRED-ROLE-CARDINALITY-CONSTRAINT Grm.card1-1
                PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT Grm.card0-N;

REGISTERED AS {RelMan.relationshipClass.1};
```

D.2 Représentation GRM networkContainsManagedElementAnd-Circuit

```
networkContainsManagedElementAndCircuit RELATIONSHIP MAPPING
        RELATIONSHIP CLASS doubleContainment;
        BEHAVIOUR networkContainsManagedElementAndCircuitBehaviour
           DEFINED AS BEHAVIOUR
              `` Un réseau (containing) contient des éléments géré
              (contained1) et des circuits (contained2) établis entre
              ces éléments.'';;
```

```
        ROLE contained1 RELATED-CLASSES managedElement
        REPRESENTED-BY NAMING managedElement-network
                USING SUBORDINATE,
        ROLE contained2 RELATED-CLASSES interExangeCircuit
        REPRESENTED-BY NAMING interExangeCircuit-network
                USING SUBORDINATE,
        ROLE containing RELATED-CLASSES network
        REPRESENTED-BY NAMING managedElement-network
                USING SUPERIOR;

        OPERATIONS MAPPING
                ESTABLISH
                        MAPS-TO-OPERATION
                        CREATE OF containing,
                TERMINATE
                        MAPS-TO-OPERATION
                        DELETE OF containing,
                USER DEFINED queryContaining
                        MAPS-TO-OPERATION
                        GET networkId OF containing,
                USER DEFINED queryContained
                        MAPS-TO-OPERATION
                        GET managedElementId OF contained,
                BIND bind1
                        MAPS-TO-OPERATION
                        CREATE OF contained1,
                UNBIND unbind1
                        MAPS-TO-OPERATION
                        DELETE OF contained1,
                BIND bind2
                        MAPS-TO-OPERATION
                        CREATE OF contained2,
                UNBIND unbind2
                        MAPS-TO-OPERATION
                        DELETE OF contained2;
REGISTERED AS {RelMan.representation.1 };
```

D.3 module ASN.1

```
GRM
DEFINITIONS ::= BEGIN

card0-N ::= INTEGER
card0-1 ::= INTEGER (0..1)
card1-1 ::= INTEGER (1..1)

END
```

Bibliographie

[10198] iso, *Management Knowledge Management Function, Amendement 1 : Extension for General Relationship Model*, 10164-16, April 1998.

[ABC94] Arnold (A.), Bégay (D.) et Crubillé (P.). – *Construction and Analysis of Transition Systems with MEC.* – World Scientific, 1994, *AMAST Series in Computing*, volume 3.

[AFN+97] Andrey (L.), Festor (O.), Nataf (E.), Schaff (A.) et Tata (S.). – Expérience multi-FDT sur un modèle de gestion de configuration d'interconnexion de commutateurs. *Technique et science informatiques*, vol. 16, n6, 1997, pp. 753–777.

[Agu89] Aguilar (L.). – NCMA: A Management Architecture that Integrates Entreprise Network Assets. *In : [MW89]*, pp. 27–39. – 1989.

[Arn92] Arnold (A.). – *Systèmes de transitions finis et sémantique des processus communicants.* – Masson, 1992.

[AY95] Arai (K.) et Yemini (Y.). – MIB view Language (MVL) for SNMP. *In : [SRFV95]*, pp. 454–465. – 1995.

[BA93] Brlek (S.) et Arnold (A.). – Conception d'un système de gestion des appels téléphoniques. Un exemple d'utilisation des méthodes formelles. *In : CFIP*, éd. par Dssouli (R.) et Bochmann (G.v.). pp. 149–164. – HERMES, 1993.

[Bap93] Bapat (S.). – Richer Modeling Semantics for Management Information. *In : [HY93]*, pp. 15–28. – 1993.

[Bap94] Bapat (S.). – *Object-Oriented Networks : Models for architecture, operations and management.* – Prentice Hall, 1994.

[Bar95] Bars (G.). – Principes de la hiérarchie numérique synchrone. *L'écho des RECHERCHES*, vol. 161, 3e trimestre 1995, pp. 3–14.

[BDBR96] Bennani (A.), Dssouli (R.), Benkiran (A.) et Rafiq (O.) (édité par). – *Colloque Francophone sur l'Ingénierie des Protocoles.* – ENSIAS, Octobre 1996.

[Ber85] Berge (C.). – *Graphs.* – North-Holland, 1985.

[BL93] Benz (C.) et Leischner (M.). – A High Level Specification Technique for Modeling Network and their Environnment including Semantic Aspects. *In : [HY93]*, pp. 29–43. – 1993.

[BLR93] Bartocci (A.), Larini (G.) et Romellini (C.). – A first attempt to combine GDMO and SDL techniques. *In : [FS93a]*, pp. 333–345. – 1993.

[Boo91] Booch (G.). – *Object-oriented design with applications.* – Benjamin Cummings, 1991.

[CC95] Christensen (H.) et Colban (E.). – *Information Modelling Concepts.* – Rapport technique, tinac, April 1995.

[CCI92a] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *Principles for a Telecommunications Management Network*, CCITT.M.3010, January 1992.

[CCI92b] International Telecommunication Union - Telecommunication Sector (ITU-T), *Technologies de l'Information - Interconnexion de Systèmes Ouverts - Structure des Informations de Gestion: Directives por la Définition des Objets Gérés* , CCITT.X.722, Janvier 1992.

[CCI95] International Telecommunication Union - Telecommunication Sector (ITU-T), *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 7: General Relationship Model*, CCITT.X.725, June 1995.

[CF93] Clemm (A.) et Festor (O.). – Behavior, Documentation and Knowledge: An Approach for the Treatment of OSI-Behavior. *In : Fourth International Workshop on Distributed Systems: Operations and Management.* – 1993. October 5-6, 1993, Long Branch, New-Jersey, USA.

[CFSD90] Case (J.), Fedor (M.), Schoffstall (M.) et Davin (J.), *A Simple Network Management Protocol*, RFC1157, SNMP Research Inc., Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[Cha97] Chatt (T.R.). – TMN/C++: An object-oriented API for GDMO, CMIS, and ASN.1. *In : [LSS97]*, pp. 177–191. – 1997.

[Che76] Chen (P.PS.). – The entity-relationship model - toward a unified view of data. *ACM Transaction on Database System*, vol. 1, n1, Mars 1976, pp. 9–36.

[CK97] Chen (G.) et Kong (Q.). – Integrated TMN service provisionning and management environment. *In : [LSS97]*, pp. 99–112. – 1997.

[Cle93] Clemm (A.). – Incorporating Relationship into Management Information. *In : 2nd Network Management and Control Workshop.* – IEEE, September 1993.

[Cle94] Clemm (A.). – *Modellierung und Hanhabung von Beziehungen zwischen Managementobjekten im OSI-Netzmanagement.* – Thèse de PhD, Ludwig-Maximilians-Universität München, Mai 1994.

[CM95] Chapman (M.) et Montesi (S.). – *Overall Concepts and Principles of TINA.* – Rapport technique, tinac, February 1995.

[CMRW93a] Case (J.), McCloghrie (K.), Rose (M.) et Waldbusser (S.), *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.

[CMRW93b] Case (J.), McCloghrie (K.), Rose (M.) et Waldbusser (S.), *Management-to-Manager Management Information Base*, RFC1451, SNMP Research Inc., Hughes LAN Systems Inc., Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

[CMRW93c] Case (J.), McCloghrie (K.), Rose (M.) et Waldbusser (S.), *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.

[Cod70] Codd (E.F.). – A relationnal model of data for large shared data banks. *Comm ACM*, vol. 33, n6, June 1970, pp. 377–387.

[COSS98] Cherkaoui (O.), Obaid (A.), Serhouchouni (A.) et Simoni (N.). – QOS Metrics tool using management by delegation. *In : NOMS98*, pp. 836–848. – 1998.

[CR91] Clinger (W.) et Rees (J.). – Revised Report on the Algorithmic Language Scheme. *ACM Lisp Pointers*, vol. 4, n3, 1991. – Available at http://www.cs.indiana.edu/scheme-repository/doc/standards/r4rs.ps.gz.

[CS95] Cornily (J.M.) et Stephan (T.). – Application des techniques des systèmes répartis ouverts

à la gestion des réseaux SDH. *L'echo des RECHERCHES*, vol. 161, Septembre 1995, pp. 29–38.

[CW97] Chadha (R.) et Wuu (S.). – Incorparating Manageability into Distributed Software. *In : [LSS97]*, pp. 489–502. – 1997.

[CZ92] Comité Consultatif International Télégraphique et Téléphonique (CCITT), *Specification and Description Language (SDL) 1992*, CCITT-Z-100-92, Janvier 1992.

[DB97] Dini (P.) et Boutaba (R.). – Deriving Varaible Polling Frequency Policies for Pro-active Management in Networks and Distributed Systems. *In : [LSS97]*, pp. 541–552. – 1997.

[DI/96] European Telecommunications Standards Institute, *Telecommunications Management Network (TMN); Generic managed objects*, ETSI-IETS 300 293 : DI/NA-043307, August 1996.

[Din97] Dini (P.). – Migrating Wireless Services: A Management Solution for Automatic Reconfiguration. *In : International Conference on Telecommunications*, éd. par Larvey (W.J.), pp. 1103–1108. – Swinburne University of Technology - Melbourne, Australia, April 1997.

[Dit91] Dittrich (A.). – Composite Managed Objects. *In : [KZ91]*, pp. 789–800. – 1991.

[DLT95] Derrick (J.), Linington (P.F.) et Thompson (S.J.). – Formal Description Techniques for Object Management. *In : [SRFV95]*, pp. 641–653. – 1995.

[DLW93] Deng (R.H.), Lazard (A.A.) et Wang (W.). – A Probalistic Approach to Fault Diagnosis in Linear Lightwave Network. *In : [HY93]*, pp. 697–708. – 1993.

[DMT97] DMTF. – *Common Information Model (CIM) version 1.1.* – Rapport technique, Desktop Management Task Force, September 1997.

[DvB97] Dini (P.) et v. Bochmann (G.). – Agent based Management of Distributed Systems with Variable Polling Frequency Policies. *In : [LSS97]*, pp. 553–564. – 1997.

[EMS97] Eberhardt (R.), Mazziotta (S.) et Sidou (D.). – Design and Testing of Information Models in a Virtual Environment. *In : [LSS97]*, pp. 461–472. – 1997.

[Feh89] Fehskens (L.). – An Architectural Strategy for Entreprise Management. *In : [MW89]*, pp. 41–60. – 1989.

[Fel89] Feldkhun (L.). – Integrated Network Management Systems. *In : [MW89]*, pp. 279–301. – 1989.

[Fes94] Festor (O.). – OSI Managed Objects Development with LOBSTERS. *In : Fifth International Workshop on Distributed Systems: Operations and Management, 12-16 Septembre 1994, Toulouse, France.* – 1994.

[Fes95] Festor (O.). – *Formalisation du comportement des objets gérés dans le cadre du modèle OSI.* – Thèse de PhD, Université Henri Poincaré, Octobre 1995.

[FJJV96] Fernandez (J.C.), Jard (C.), Jéron (T.) et Viho (G.). – An experiment in automatic generation of test suites for protocols with verification technology. *In : Science of Computer Programming, Special Issue on Industrial Relevant Applications of Formal Analysis Techniques.* – 1996.

[FMB97] Feldkhun (L.), Marini (M.) et Borioni (S.). – Integrated Customer-Focused Network Management: Architectural Perspectives. *In : [LSS97]*, pp. 17–30. – 1997.

[FN97] Festor (O.) et Nataf (E.). – Formalizing tmn information models for improving management applications. *In : International Conference on Telecommunications*, éd. par Larvey (W.J.), pp. 807–812. – Swinburne University of Technology - Melbourne, Australia, April 1997.

[FNA96a] Festor (O.), Nataf (E.) et Andrey (L.). – *MODE-FE : A GRM/GDMO Parser and its API : Release 1.0 Reference Manual RT 0190.* – Technical Report n0190, INRIA Lorraine, 1996.

[FNA96b] Festor (O.), Nataf (E.) et Andrey (L.). – *Simulation interactive du modèle de gestion d'interconnexion de commutateurs à l'aide de LOBSTERS et MODE. RR 2790.* – Rapport technique, INRIA, 1996.

[FS93a] Færgemand (O.) et Sarma (A.) (édité par). – *SDL'93.* – Elsevier Science Publishers B. V., 1993.

[FS93b] Fisher (J.) et Schröder (R.). – Combined Specification Using SDL and ASN.1. *In : [FS93a]*, pp. 293–304. – 1993.

[G.792] International Telecommunication Union - Telecommunication Sector (ITU-T), *Synchronous Digital Hierarchy (SDH) management information model for the network element view*, G.774, September 1992.

[G.893] International Telecommunication Union - Telecommunication Sector (ITU-T), *Architecture des réseaux de transport à hiérarchie numérique synchrone*, G.803, Mars 1993.

[G.896] International Telecommunication Union - Telecommunication Sector (ITU-T), *Application of the RM-ODP framework to the management of the transmission network*, G.851-01, November 1996.

[GM95] Graubman (P.) et Mercouroff (N.). – *Engineering Modelling Concepts (DPE architecture).* – Rapport technique, tinac, February 1995.

[Gro97] Object Management Group, *CORBAServices: Common Object Services Specification*, Object Management Group, 1997.

[Gro98] Object Management Group, *CORBA2.2/IIOP Specification*, Object Management Group, 1998.

[HK87] Hull (R.) et King (R.). – Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, vol. 19, n3, September 1987, pp. 201–260.

[HW98a] Harrington (D.) et Wijnen (B.), *An Architecture for Describing SNMP Management Frameworks*, RFC2271, January 1998.

[HW98b] Harrington (D.) et Wijnen (B.), *Message Processing and Dispatching for the Simple Network Management Protocol*, RFC2272, January 1998.

[HY93] Hegering (H.-G.) et Yemini (Y.) (édité par). – *Integrated Network Management III.* IFIP. – North-Holland, Avril 1993.

[IR94] Indulska (J.) et Raymond (K.). – Application of odp and type management concepts to network management. *Annals of Telecommunication*, vol. 49, n1-2, 1994, pp. 57–64.

[ISO84] International Organization for Standardization (ISO), *Basic Reference Model*, ISO-7498, October 1984.

[ISO87] International Organization for Standardization (ISO), *Specification of Basic Encoding Rule for Abstract Syntax Notation Number One (ASN.1)*, ISO-8825, December 1987.

[ISO89] International Organization for Standardization (ISO), *Basic Reference Model - Part 4: Management framework*, ISO-7498.4, November 1989.

[ISO91a] International Organization for Standardization (ISO), *System Management - Part 12:Test management function*, ISO-10164.12, July 1991.

[ISO91b] International Organization for Standardization (ISO), *System Management - Part 9: Objects ans Attributes for Access Control*, ISO-10164.9, July 1991.

[ISO92a] International Organization for Standardization (ISO), *Information Technology - Open System Interconnection - System Management Overview*, ISO-10040, January 1992.

[ISO92b] International Organization for Standardization (ISO), *System Management - Part 3: Attributes for Representing Relationships*, ISO-10164.3, January 1992.

[ISO92c] International Organization for Standardization (ISO), *System Management - Part 5: Event Report Management Function*, ISO-10164.5, January 1992.

[ISO92d] International Organization for Standardization (ISO), *System Management - Part 6: Log Control Function*, ISO-10164.6, January 1992.

[ISO92e] International Organization for Standardization (ISO), *Structure of Management Information - Part 1: Management Information Model*, ISO-10165.1, January 1992.

[ISO92f] International Organization for Standardization (ISO), *Structure of Management Information - Part 2: Definition of Management Information*, ISO-10165.2, January 1992.

[ISO92g] International Organization for Standardization (ISO), *Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects*, ISO-10165.4, January 1992.

[ISO92h] International Organization for Standardization (ISO), *Structure of Management Information - Part 5: Generic Management Information*, ISO-10165.5, January 1992.

[ISO92i] International Organization for Standardization (ISO), *Abstract Syntax Notation Number One (ASN.1) - Part 1: Specification of Basic Notation*, ISO-8824.1, December 1992.

[ISO92j] International Organization for Standardization (ISO), *Information processing systems – Database language SQL*, ISO-9075, 1992.

[ISO95] International Organization for Standardization (ISO), *Structure of Management Information - Part 7: General Relationship Model*, ISO-10165.7, June 1995.

[ISO96] International Organization for Standardization (ISO), *Amendment 4 to ISO/IEC 10165-4:1992. GDMO+: Specifying the Behaviour of Managed Objects*, ISO-10165.4, May 1996.

[ITU92] International Telecommunication Union - Telecommunication Sector (ITU-T), *Structure of Management Information - Part 2: Definition of Management Information*, ITUT.X.721, January 1992.

[Jma98] Jmaili (H.). – Implantation d'un service de relations dans un environnement de gestion osi. – ENSEEITH / INRIA, juin 1998. Rapport de fin de stage d'école d'ingénieur.

[JP93] Jordaan (J.F.) et Paterok (M.E.). – Event Correlation in Heterogeneous Networks Using the OSI Management Framework. *In : [HY93]*, pp. 683–695. – 1993.

[Kar93] Karner (G.). – Semantic Integration of ASN.1 into SDL. *In : [FS93a]*, pp. 305–315. – 1993.

[KD94] Keller (J.) et Dubuisson (O.). – Formal Description of OSI Management Information Structure as a Prerequisite for Formal Specifications of TMN Interfaces. *In : Toward a Pan-European Telecommunication Service Infrastructure - IS&N'94*, éd. par Kugler (H. Jürgen), Mullery (A.) et Niebert (N.). Lecture Notes in Computer Science, pp. 433–442. – Springer-Verlag, 1994.

[Kel96] Keller (J.). – An Extension of GDMO for Formalizing Managed Objects Behaviour. *FORTE*, 1996.

[Kil91] Kilov (H.). – Generic Information Modeling Concepts: A Reusable Component Library. *In : Technology of Object-Oriented Langages and Systems*, éd. par Bévézin (J.) et Meyer (B.). pp. 187–201. – Prentice Hall, 1991.

[Kil92] Kilov (H.). – Understand → Specify → Reuse : precise specification of behaviour and relationships. *In : DSOM*. – 1992.

[Kil95] Kilov (H.). – Understanding the semantics of collective behavior: the ISO General Relationship Model. *In : ECOOP95 one day workshop: Use of Object-Oriented technology for Network Design and Management. Aarhus, Denmark, August 7-11, 1995.* – 1995.

[KK91] Klemba (K.) et Kosarchyn (M.). – A Model for Object Relationship Management. *In : [KZ91]*, pp. 801–812. – 1991.

[KR94a] Kilov (H.) et Ross (J.). – Generic concepts for Specifying Relationships. *In : Network Operations and Management Symposium. Orlando, Florida.* IFIP. – efeb 1994.

[KR94b] Kilov (H.) et Ross (J.). – *Information Modeling: an Object-oriented Approach.* – Prentice Hall, 1994.

[KS96] Kabore (P.) et Schaff (A.). – Méthode abstraite de test pour l'administration de réseaux. *In : [BDBR96]*, pp. 337–355. – 1996.

[KSS96] Kalyanasundaram (P.), Sethi (A.S.) et Sherwin (C.M.). – Design of a Streadshhet Paradigm for Network Management. *In : Workshop on Distributed systems : Operations and Management.* IFIP / IEEE. – 1996.

[KSSZ97] Kalyanasundaram (P.), Sethi (A.S.), Sherwin (C.M.) et Zhu (D.). – A Spreadsheet-Based Scripting Environment for SNMP. *In : [LSS97]*, pp. 752–765. – 1997.

[KZ91] Krishnan (I.) et Zimmer (W.) (édité par). – *Integrated Network Management II.* IFIP. – North-Holland, April 1991.

[LKP98] Liotta (A.), Knight (G.) et Pavlou (G.). – Modelling Network and System Monitoring over the Internet with Mobile Agent. *In : NOMS98*, pp. 303–312. – 1998.

[LS98] Levi (D.) et Stewart (B.), *SNMPv3 Applications*, RFC2273, January 1998.

[LSS97] Lazar (A.), Saracco (R.) et Stadler (R.) (édité par). – *Integrated Management V.* IFIP. – Chapman & Hall, May 1997.

[M3195] International Telecommunication Union - Telecommunication Sector (ITU-T), *M3100 : Generic Network Information Model*, M3100, June 1995.

[MAG97] Marre (B.), Arnold (A.) et Gaudel (M.C.). – Une expérience de validation de spécification par des formalismes différents : le nœud de transit. *Technique et science informatiques*, vol. 16, n6, 1997, pp. 753–777.

[Mar93] Marshall (L S.). – Using VDM to Specify Managed Object Relationship. *In : FORTE.* – 1993.

[Maz97] Mazziota (S.). – *Spécification et Génération de Tests du Comportement Dynamique des Systèmes à Objets Répartis.* – Thèse de PhD, Université de Nice-Sophia Antipolis U.F.R. Faculté des sciences, Octobre 1997.

[Mey88] Meyer (B.). – *Object-oriented Software Construction.* – Prentice Hall, 1988.

[Mey92] Meyer (B.). – *Introduction à la théorie des langages de programmation.* – InterEditions, 1992.

[ML89] Mazumdar (S.) et Lazar (A.A.). – Knowledge-Based Monitoring of Integrated Networks. *In : [MW89]*, pp. 235–243. – 1989.

[MMP93] Mazaher (S.) et Møller-Pedersen (B.). – On the use of SDL-92 for the specification of Behaviour in OSI network Management Objects. *In : [FS93a]*, pp. 317–331. – 1993.

[MNM87] Mackert (L.F.) et Neumeier-Mackert (I.B.). – Communicating Rule Systems. *In : Proc. 7th. Int. Symp. on Protocol Specification, Testing and Veri fication, H. Rudin, C.H. West (editors), North-Holland 1987*, pp. 77–88. – 1987.

[MR91] McCloghrie (K.) et Rose (M), *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II*, RFC1213, Hughes LAN Systems Inc., Performance Systems International, March 1991.

[MT89] Margarida (L.) et Tarouco (R.). – Intelligent Network Management. *In : [MW89]*, pp. 141–155. – 1989.

[Mul97] Muller (P. A.). – *Modélisation object avec UML.* – Eyrolles, 1997.

[MW89] Meandzija (B.) et Westcott (J.) (édité par). – *Integrated Network Management I.* IFIP. – North-Holland, Mai 1989.

[N1089] International Organization for Standardization (ISO), *Modelling techniques and their use in ODP*, ISO-JTC1 SC21 WG7 N109, April 1989.

[Nat97] Nataf (E.). – Comportement des relations de la gestion osi. *In : Colloque Francophone de l'Ingéniérie des Protocoles*, éd. par Leduc (G.). – Liège - Belgique, Septembre 1997.

[Nat98] Nataf (E.). – Comportement et environnement pour les relations de la gestion osi. *Revue Électronique Réseaux et Informatique Répartie*, sera publié en septembre 1998.

[NFA97] Nataf (E.), Festor (O.) et Andrey (L.). – RelMan : A GRM-Based Relationship Manager. *In : Integrated Management V*, éd. par Lazar (A.), Saracco (R.) et Stadler (R.). IFIP, pp. 661–672. – San Diego - USA, May 1997.

[NFS96] Nataf (E.), Festor (O.) et Schaff (A.). – *Validation du modèle de gestion d'interconnexion de commutateurs à l'aide de système de transitions étiquetées RR 2769.* – Rapport technique, INRIA, 1996.

[NFSA96] Nataf (E.), Festor (O.), Schaff (A.) et Andrey (L.). – Validation d'un modèle de gestion d'interconnexion de commutateurs à l'aide de système de transitions étiquetées. *In : Colloque Francophone sur l'Ingénierie des Protocoles*, éd. par Bennani (A.), Dssouli (R.), Benkiran (A.) et Rafiq (O.). pp. 83–97. – Rabat - Maroc, Octobre 1996.

[NG95] Noble (J.) et Grundy (J.). – Explicit Relationships in Object Oriented Development. *In : TOOLS'95.* – 1995.

[NMF92] Network Management Forum, *The Ensemble Concepts and Format, Issue 1.0*, NMF, August 1992.

[NMF94] Network Management Forum, *Switch Interconnection Management: Configuration Management Ensemble*, NMF, November 1994.

[NR97] International Organization for Standardization (ISO), *GRM+ : an Extension of GRM for Specifying Managed Relationship Behaviour*, 7N1234R, June 1997, accessible à : ftp://ftp.gmd.de/documents/iso/RM-ODP/Helsinki_Input/7N1234.ps.gz.

[NS96] Najm (E.) et Stefani (J.B.) (édité par). – *Formal Methods for Open Object-based Distributed Systems.* IFIP. – Chapman & Hall, 1996.

[PK96] Popien (C.) et Kuepper (A.). – An object-oriented description of services in a distributed system. *In : [NS96]*, pp. 261–268. – 1996.

[Raf95] Rafiq (O.). – Historique et problématique des réseaux informtiques : Architecture et ingénierie des protocoles. *In : Réseaux de communication et conception de protocoles*, éd. par Juanole (G.), Serhrouchni (A.) et Seret (D.). pp. 11–33. – HERMES, 1995.

[Red96] Redberg (D.). – *Object Oriented Behavioural Specifications*, chap. 15 : The search for the linking invariant : behavioral modeling versus modeling behaviour, pp. 241–264. – Kluwer Academic Publisher, 1996.

[RM90] Rose (M.) et McCloghrie (K.), *Structure and Identification of Management Information for TCP/IP-based Internets*, RFC1155, Performance Systems International,Hughes LAN Systems Inc., May 1990.

[Ros94] Rose (M. T.). – *The Simple Book.* – Prentice Hall, 1994, seconde édition.

[RS97] Rahkila (S.) et Stenberg (S.). – Experiences on building a distributed computing platform prototype for telecom network and service management. *In : [LSS97]*, pp. 127–138. – 1997.

[Rum87] Rumbaugh (J.). – Relations as Semantics Constructs in an Object-Oriented Language. *In : Proc. of the OOPSLA'87 conference.* – 1987.

[Rum91] Rumbaugh (J.). – *Object-Oriented Modeling and Design.* – Prentice Hall, 1991.

[Sa93] Sibilla (M.) et all. – Management Information Base : Guidelines Leading From Generic Specification of Managed Object Relationship to Consistent Implementation. *In : Distributed Systems : Operation and Management.* – October 5-6 1993.

[SBP98] Susilo (G.), Bieszczad (A.) et Pagurek (B.). – Infrastructure for Advanced Network Management based on Mobile Code. *In : NOMS98*, pp. 322–333. – 1998.

[Sch92] Schneider (J.M). – *Protocol Engineering: A Rule-based Approach.* – Vieweg, 1992.

[SFL96] Stepien (B.), Frooqui (K.) et Logrippo (L.). – An experience modelling telecommunications systems using ODP-DLcomp. *In : [NS96].* – 1996.

[SH97] Soukouti (N.) et Holleberg (U.). – Joint Inter Domain Management: CORBA, CMIP and SNMP. *In : [LSS97]*, pp. 153–164. – 1997.

[Sid95] Sidou (D.). – TIMS : a TNM-based Information Model Simulator, Principles and Application to a Simple Case Study. *In : Distributed System Operation and Management.* – October 19-22 1995.

[Sid97] Sidou (D.). – *Validation of functional Behavior Specifications of Distributed Object Framework.* – Thèse de PhD, École Polytechnique Fédérale de Lausanne, 1997.

[Slo94] Sloman (M.) (édité par). – *Network and Distributed Systems Management.* – Addison-Wesley, 1994.

[SP94] Sethi (S.A.) et P. (Kalyanasundaram). – A Spreadsheet Paradigm for Network Management and Control. *In : Distributed systems : Operations and Management.* IFIP / IEEE. – 1994.

[SRFV95] Sethi (A. S.), Raynaud (Y.) et Faure-Vincent (F.) (édité par). – *Integrated Network Management IV.* IFIP. – Chapman & Hall, May 1995.

[SS89] Strutt (C.) et Shurtleff (D.G.). – Architecture for an Integrated, Extensible Entreprise Management Director. *In : [MW89]*, pp. 61–72. – 1989.

[Sta93] Stalling (W.). – *SNMP, SNMPv2, and CMIP the Practical Guide to Network-Management Standards.* – Addison Wesley, 1993.

[Sun97] Sun Microsystems Inc. – *The Java Management API*, 1997. http://java.sun.com:80/products/JavaManagement/.

[Syl89] Sylor (M.). – Guideline for Structuring Manageable Entities. *In : [MW89]*, pp. 169–183. – 1989.

[TAF97] Tata (S.), Andrey (L.) et Festor (O.). – A practical experience on validating GDMO-based Information Models with SDL'88 and SDL'92. *In : SDL'97 : Time for Testing SDL, MSC and Trends*, éd. par Cavalli (A.) et Sarma (A.). pp. 367–380. – Elsevier science publishers B.V., September 1997.

[Tat97] Tata (S.). – Modélisation d'un agent osi en sdl'92. – Université Henri Poincaré, 1997. Rapport de DEA.

[Tec94] The ATM Forum, *M4 Network-View Interface Requirement, and Logical MIB*, Technical Committee, october 1994.

[Tod97a] Todd (S.), *HMMP Overview*, Not yet assigned, Microsoft Corporation, One Microsoft Way, Redmond, WA 98053, USA, May 1997.

[Tod97b] Todd (S.), *Introduction to HMMP*, Not yet assigned, Microsoft Corporation, One Microsoft Way, Redmond, WA 98053, USA, May 1997.

[Tsa93] Tsay (J.L.). – Object-Oriented Intelligent Telecommunications Network Management System. *In : [HY93]*, pp. 557–567. – 1993.

[VG93] Vucetic (J.) et Gersht (A.). – Damage Assessment - A Network Management Application Based On An Integrated Information Model Of The Network. *In : [HY93]*, pp. 641–652. – 1993.

[VPK97] Vassila (A.), Pavlou (G.) et Knight (G.). – Active Object in TMN. *In : [LSS97]*, pp. 139–150. – 1997.

[Wal95] Waldbusser (S.), *Remote Network Monitoring Management Information Base (RMON)*, Carnegie Mellon University, February 1995.

[WH98] White (S.) et Hapner (M.). – *The JDBC Database Access API.* – Sun Microsystems Inc., may 1998, 2.0 édition. http://www.javasoft.com/products/jdbc.

[WS89] Warrier (S.U.) et Sunshine (A.C.). – A Platform for Heterogeneous Interconnection Network Management. *In : [MW89]*, pp. 13–24. – 1989.

[YT94] Yamaguchi (H.) et Tanaka (T.). – Application of Commercial Databases to Management. *In : [Slo94]*, pp. 499–515 (Chap 19). – 1994.

[YYF91] Yamamura (T.), Yasushi (T.) et Fujii (N.). – A Study on an End Customer Controlled Circuit Reconfiguration System for Leased Line Networks. *In : [KZ91]*, pp. 383–394. – 1991.

[ZS97] Znaty (S.) et Simoni (N.). – *Gestion de réseau et de service.* – InterEditions, 1997.

Université HENRI POINCARÉ NANCY 1

FACULTÉ DES SCIENCES

Monsieur **NATAF Emmanuel**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 9 novembre 1998 n° 105

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ NANCY 1 LE PRÉSIDENT

J.P. FINANCE

Résumé :

L'administration de réseaux informatique ou de télécommunication gagne en importance avec l'ampleur de leurs utilisations. La diversité des constructeurs d'éléments de réseaux (ou ressources) ne permet pas d'avoir une vision homogène des réseaux. Il est donc nécessaire de standardiser les interfaces d'accès à ces ressources dans des buts de gestion. Des langages de spécifications indépendant des constructeurs et de la technologie existent et sont utilisés dans des situations réelles. L'OSI est particulièrement impliquée et propose plusieurs modèles, dont un modèle de l'information orienté objet avec son langage associé: GDMO. Cette thèse aborde différents sujets autours des relations, de leurs modélisations et de leurs utilisations dans le cadre de la gestion de réseaux. Le formalisme actuel, tel qu'il est proposé par l'ISO dans le Modèle Général de Relation (GRM), ne concerne pas le comportement associé à ces relations. Il faut donc étendre le GRM de façon à préciser formellement le comportement qui sera imposé aux objets participant à une relation. L'utilisation de relations par des applications d'administration de réseaux nécessite une infrastructure logicielle dont nous proposons des bases. Ainsi, l'offre de services de relation et la gestion de ces relations peuvent être tirés des spécifications du GRM étendues.

Mots clés : Gestion de réseaux, Objets gérés, Relations gérées, Comportement, Spécification formelle, Plate-forme de gestion, SNMP, GDMO, GRM

Abstract:

Telecommunication or computer network management is increasing in importance and use. The diversity of networks elements manufacturers (or resources) is against an homogeneous view of the network. The standardisation of resources access interfaces is necessary to management goals. Specifications languages independent from manufacturers and technologies are existing and are used in real life situation. The OSI is specially involved and proposed several models as an information model with its associated object oriented language: GDMO. This thesis is about various views of relationships, their specification and their exploitation in network management area. The actual formalism proposed by the ISO in the General Relationship Model (GRM), does not concern the behaviour of these relationships. There is a need for an extension of the GRM in order to formally specify the behaviour that will be imposed to objects participating in a relationship. Use of relationships by network management application needs a software environment that we propose. This software allows relationships services to be extracted from extended GRM specifications.

Keywords: Network management, Managed Object, Managed Relationship, Behaviour, Formal specification, Management platform, SNMP, GDMO, GRM