Un Serveur d'Archivage de Messages

Yi Zhi You

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 36 ID

THESE

présentée par

Yi Zhi YOU

pour obtenir le titre de

DOCTEUR

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique, Image, Intelligence Artificielle et Algorithmique)

S A M

Un Serveur d'Archivage de Messages

Soutenue à Saint-Etienne le 19 Décembre 1989

COMPOSITION DU JURY

Monsieur Y. RAYNAUD	Président et rapporteur
Monsieur C. HUITEMA	Rapporteur
Madame Y. AHRONOVITZ	Examinateurs
Messieurs G. AUTIER	
P.A. PAYS	
B. PEROCHE	

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 36 ID

THESE

présentée par

Yi Zhi YOU

pour obtenir le titre de

DOCTEUR

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique, Image, Intelligence Artificielle et Algorithmique)

S A M

Un Serveur d'Archivage de Messages

Soutenue à Saint-Etienne le 19 Décembre 1989

COMPOSITION DU JURY

Monsieur Y. RAYNAUD	Président et rapporteur
Monsieur C. HUITEMA	Rapporteur
Madame Y. AHRONOVITZ	Examinateurs
Messieurs G. AUTIER	
P.A. PAYS	
B. PEROCHE	

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT ETIENNE

Directeur	:	M. Philippe	SAINT RAYMOND
Directeur des Etudes	:	M. Jean	CHEVALIER
Directeur des Recherches	:	M. François	MUDRY
Secrétaire Général	:	M. J.Claude	PIATEK

PROFESSEURS DE 1ère CATEGORIE

MM.	COINDE	Alexandre	Economie
	FORMERY	Philippe	Mathématiques Appliquées
	GOUX	Claude	Métallurgie
	LE COZE	Jean	Matériaux
	LOWYS	Jean-Pierre	Physique
	MATHON	Albert	Gestion
	PERRIN	Michel	Géologie
	PEROCHE	Bernard	Informatique
	PLA	Jean-Marie	Mathématiques
	RIEU	Jean	Mécanique-Résistance des Matériaux
	SOUSTELLE	Michel	Chimie
	VERCHERY	Georges	Mécanique et Matériaux

PROFESSEURS DE 2ème CATEGORIE

MM.	LADET	Pierre	Automatique
	TOUCHARD	Bernard	Physique Industrielle

DIRECTEUR DE RECHERCHE

M.	LESBATS	Pierre	Métallurgie

MAITRES DE RECHERCHE

MM.	BISCONDI	Michel	Métallurgie
	COURNIL	Michel	Chimie
	DAVOINE	Philippe	Hydrogéologie
	DRIVER	Julian	Matériaux
Mle	FOURDEUX	Angeline	Matériaux
MM.	GIRARDOT	Jean-Jacques	Informatique
	GUILHOT	Bernard	Chimie
	KOBYLANSKI	André	Métallurgie
	LALAUZE	René	Chimie
	LANCELOT	Francis	Génie Electrochimique-Biotechnologie
	MONTHEILLET	Franck	Matériaux
	THEVENOT	François	Génie des Matériaux
	THOMAS	Gérard	Chimie
	TRAN MINH	Cahn	Génie Electrochimique-Biotechnologie

PERSONNALITE HABILITEE A DIRIGER LES TRAVAUX DE RECHERCHE.

M.	MAGNIN	Thierry	Matériaux

PROFESSEUR A L'U.E.R. DE SCIENCES DE SAINT-ETIENNE

M.	VERGNAUD	Jean-Marie	Chimie des Matériaux

Remerciements

Je tiens à exprimer ma reconnaissance aux personnes qui ont accepté de juger ce travail ainsi qu'à toutes celles qui m'ont aidé au cours de ces années de thèse.

- Monsieur le Professeur Yves Raynaud de l'Université Paul-Sabatier qui me fait l'honneur de présider le jury de cette thèse.

- Monsieur Christian Huitema, Directeur de Recherche de l'INRIA Sophia-Antipolis, rapporteur, pour l'intérêt qu'il accorde à mes travaux.

- Madame Yolande Ahronovitz, Maître de Conférence à l'Université de Sciences de Saint-Etienne pour avoir accepté de participer au jury.

- Monsieur Guy Autier, Responsable Réseaux Unix de la compagnie BULL, pour sa participation à ce jury.

- Monsieur Paul-André Pays, Responsable de l'équipe "Réseaux et Systèmes Distribués" du Département Informatique Appliquée de l'Ecole des Mines de Saint-Etienne, pour sa compétence, sa disponibilité et la formation que j'ai reçue au cours de ces années de thèse.

- Monsieur le Professeur Bernard Péroche, Directeur du Département Informatique Appliquée de l'Ecole des Mines de Saint-Etienne, pour l'intérêt porté à cette thèse.

- Tous les membres du groupe AMIGO MHS+ du projet COST 11ter pour leurs discussions de grande valeur.

- Tous les membres du Département Informatique Appliquée pour leur amitié et leur aide.

- Monsieur Ehoud Ahronovitz pour ses conseils et son aide pendant l'écriture de cette thèse.

- Madame Marie-Line Barneoud, Messieurs Mohand Ourabah Benouamer, Bernard Kaddour, Dominique Michelucci, Antoine Pauze pour la correction de cette thèse.

- Le personnel du service de reprographie pour avoir assuré la réalisation de cet ouvrage.

- Ma mère, mon père, ma soeur, les parents et les amis pour leur affection et leur soutien.

SOMMAIRE

Introduction Générale

1. Le Cadre du Projet

La thèse présentée dans ce rapport a été entreprise dans le cadre du projet SAM (Serveur d'Archivage de Messages) au sein de l'équipe "Réseaux et Systèmes Distribués" dans le département Informatique Appliquée de l'Ecole des Mines de Saint-Etienne.

Le projet a démarré en Juin 1987, et fut très vite adopté par le groupe AMIGO[1] MHS+ du projet COST 11ter de la CEE. Le but du groupe AMIGO+ est l'étude d'extensions des recommandations X.400 du CCITT[2] et de l'ISO[3] pour le support de la communication de groupe, en plus de l'objectif initial: la communication inter-personnelle.

Historiquement, le SAM provient d'un autre projet, SAD (Serveur d'Archivage de Documents), qui devrait étudier les aspects stockage et archivage de documents ODA (Office Document Architecture) dans un environnement bureautique. Les résultats des premières études sont décrits dans [You 87a] et [You 87b]. Ce projet fut abandonné à cause de l'énorme travail nécessaire à sa réalisation et du peu de ressources disponibles (hommes, logiciels, etc), puis transformé en projet SAM.

Bien que les études du projet SAD ne soient pas incluses dans cette thèse, elles ont largement contribué à l'expérience acquise par l'auteur.

2. Le But du Projet

La messagerie électronique fut longtemps l'outil de communication pour les "informaticiens spécialistes". Elle fournit principalement le service du courrier inter-personnel. Grâce au développement rapide des réseaux et de la télématique, cet outil de communication est accepté par un nombre croissant d'utilisateurs "non-spécialistes".

Deux besoins sont ressentis par les utilisateurs de systèmes de messagerie électronique:

- Une norme internationale pour la messagerie électronique permettant aux utilisateurs du monde entier de communiquer.

---

[1] Advanced Messaging In GrOup. Le projet COST 11ter a financé les frais occasionnés par la coopération au sein d'AMIGO.

[2] Comité Consultatif International Télégraphique et Téléphonique.

[3] International Standard Organisation.

- D'autres outils de communication en Mode Messagerie, en plus de la messagerie inter-personnelle; en particulier des outils de transfert de fichiers et de communication de groupe.

Si bien qu'une norme internationale pour la messagerie (X.400) a été proposée par le CCITT et l'ISO, tandis que des outils de communication de groupe sont actuellement en cours d'élaboration et de réalisation.

Le principal domaine d'activités de l'équipe "Réseaux et Systèmes Distribués" est de développer de nouvelles applications de la messagerie électronique X.400 (par exemple, TFMM [Rollin 86] et CRMM [Brun 87]), et de les proposer à des organismes de normalisation.

A partir des expériences d'utilisation de la messagerie électronique X.400 et de la conception de différentes applications, nous avons constaté qu'un archiveur de messages est nécessaire dans toutes les applications. En fait, des messages échangés sont souvent liés, formant des conversations. Une bonne utilisation de la messagerie inter-personnelle nécessite donc un outil de stockage et d'archivage personnel dans un environnement bureautique, qui permet de stocker des messages reçus, de conserver les traces des messages envoyés, et de les organiser en conversations.

Comme la messagerie inter-personnelle, des archiveurs de messages sont indispensables pour des applications de communication de groupe. Ils sont plus compliqués, souvent partagés et distribués. Un archiveur de messages d'une telle application peut être utilisé comme un moyen de communication via des messages partagés. La conception et la réalisation d'un tel archiveur de messages sont souvent difficiles. De plus, aucun outil d'aide n'est disponible.

Nous avons donc démarré le projet SAM. Le but du projet est de concevoir un serveur d'archivage de messages "général" pour supporter toutes les utilisations dans différentes applications.

Nous avons commencé par étudier des applications MHS[1] et GC[2] et leurs besoins en stockage/archivage. Nos choix pour la représentation des messages sont basés naturellement sur la messagerie X.400. Cependant, vu l'existence et la vaste utilisation de divers autres systèmes de messageries (RFC-822, Profs, etc), nous devons aussi accepter d'autres formats que celui de X.400 et permettre des extensions spécifiques à l'application et au système de messagerie. En effet, les différentes applications n'ont pas les mêmes besoins. Le système de communication en mode messagerie dans un environnement bureautique peut avoir besoin d'un archiveur capable de stocker d'autres types d'objets (mémos, post-it-notes, autres types de documents) avec des messages et de les organiser de façon spécifique à l'utilisateur. Cependant, une application de communication de groupe (par exemple, une conférence répartie) peut avoir besoin de distribuer des messages dans plusieurs archiveurs partagés, de synchroniser ces archiveurs et de les protéger

---

[1] Message Handling System, i.e. Système de Messagerie en anglais.

[2] Group Communication, i.e. Communication de Groupe en anglais.

contre des accès illégaux. De même, des utilisateurs peuvent accéder à un archiveur suivant différents modes de communication; par exemple, par des accès directs en mode interactif, ou par des échanges de messages en mode messagerie.

Tout cela rend difficile (voire impossible) la conception d'un serveur d'archivage de messages "général". Notre solution consiste à construire un archiveur adaptable, qui doit pouvoir convenir à différentes applications n'ayant pas les mêmes besoins. Cela nous a conduit à proposer d'abord un modèle architectural d'archiveurs. Le modèle décompose un archiveur spécifique en deux parties, un composant de base qui réalise les fonctions de stockage et d'archivage, et un composant adaptable qui réalise les fonctions particulières à l'application à partir de fonctions fournies par le composant de base. Le composant de base (**MAP** pour Message Archive Processor) est commun à tous les archiveurs de SAM; il est assez général et flexible pour convenir à toutes les applications par le biais du composant adaptable (l'ensemble de **Front_Ends** de MAP).

Un autre aspect du projet est de concevoir des archiveurs réels pour de nouvelles applications, ou pour des applications existantes afin de les rendre plus faciles à utiliser. Cela nous permet aussi de valider nos idées et choix. Dans ce but, nous avons conçu plusieurs applications intéressantes et utiles:

- un archiveur personnel de messages pour un environnement bureautique.
- une extension de la stratégie de répartition pour une conférence répartie.
- un système de vote en mode messagerie qui permet d'effectuer des votes ou des sondages par courrier électronique.

Donc le projet SAM ([You 89a]) consiste à :

- étudier les besoins de stockage et d'archivage d'applications de messagerie et de communication de groupe (i.e. applications MHS et GC).
- proposer un modèle architectural adaptatif.
- concevoir et réaliser un archiveur général: MAP.
- intégrer et adapter MAP pour des applications spécifiques de messagerie et de communication de groupe.

3. Plan de la Thèse

Cette thèse comporte trois parties.
La première partie présente les principaux concepts de la messagerie électronique et des archiveurs de messages utilisés dans des applications existantes. Le chapitre 1 décrit brièvement les normes utilisées dans la conception et la spécification de SAM. Dans le chapitre 2, nous étudions d'abord certains archiveurs de messages utilisés dans des applications MHS et GC. Ensuite nous donnons les fonctionnalités qu'un archiveur de messages général doit supporter. Un exemple est également donné dans ce chapitre qui sera utilisé pour illustrer certains de nos choix. En fin du chapitre 2, nous revoyons plus en détail le projet SAM.

La deuxième partie composée de 4 chapitres présente la solution SAM. Le chapitre 3 définit le modèle architectural de SAM. Un archiveur SAM d'une application est divisé en deux niveaux: un noyau (MAP) et des Front_Ends spécifiques à l'application. Puis nous donnons le modèle conceptuel d'applications MHS et GC que SAM doit supporter; nous pouvons en déduire des objets que MAP doit traiter. Dans le chapitre 5, nous exposons la conception de MAP. Nous commençons par présenter une vue externe de MAP et des objets qu'il gère. Puis nous décrivons dans leurs grandes lignes les mécanismes principaux de MAP. Ensuite les divers aspects de MAP sont détaillés dans ce même chapitre. Pour terminer la deuxième partie, le chapitre 6 présente l'implantation de MAP, ainsi que divers outils de réalisation possibles.

La troisième partie contient des exemples d'utilisation de SAM dans des applications réelles. Elle comprend une introduction et trois chapitres. L'introduction décrit brièvement la méthodologie d'utilisation de SAM, i.e. comment il s'adapte à de nouvelles applications. Le chapitre 7 montre l'utilisation de SAM dans un environnement distribué où des archiveurs sont distribués géographiquement et où des informations sont réparties dans ces archiveurs. Un modèle de répartition y est également présenté. Le chapitre 8 est un exemple d'utilisation dans un environnement bureautique; il s'agit d'un archiveur personnel qui sert d'intermédiaire entre un utilisateur et le système de messagerie X.400. Le dernier chapitre présente une nouvelle application de communication de groupe: un système de vote en mode messagerie, et l'utilisation de SAM comme archiveur.

Première Partie

Archiveurs de Messages

Chapitre 1

Présentation de la Messagerie Electronique
et de la Communication de Groupe

Chapitre 2

Etat de l'Art : Archiveurs de Messages

Chapitre 1
Présentation de la Messagerie Electronique et Communication de Groupe

1. Communication et Outils

La plupart des activités dans les sociétés modernes dépendent de la communication pour échanger et partager des informations. Deux modes de communication existent:

- synchrone où les partenaires de la communication doivent être "en-ligne" dans la même activité et en même temps;
- *asynchrone* où les partenaires de la communication ne sont pas nécessairement tous présents en même temps pendant le processus de communication.

La communication par téléphone est un exemple de communication synchrone, le courrier normal est un exemple de communication asynchrone.
Nous nous intéressons ici à la communication asynchrone par échange de messages.

On peut aussi classer les communications en communications inter-personnelles et communications de groupe. Pendant la communication inter-personnelle, l'information s'échange entre deux individus. La communication de groupe est structurée: on échange de l'information entre un groupe structuré de personnes pour achever un travail coopératif.

Un outil de communication permet aux membres ou partenaires d'un processus de communication d'échanger et de partager de l'information. Un tel outil peut supporter un ou plusieurs modes de communication et une ou plusieurs classes de communication. Par exemple, le téléphone traditionnel permet une communication synchrone et inter-personnelle; les outils traditionnels de la messagerie électronique permettent la communication asynchrone et inter-personnelle; *Cognotor* [Foster 86] de Xerox utilise la communication synchrone en groupe afin d'offrir la résolution coopérative de problèmes en temps réel (cooperative real-time problem solving); *CRMM* [Brun 88] est un outil de conférence répartie qui utilise la communication asynchrone de groupe en mode messagerie.

Dans le reste de cette partie, nous allons présenter séparément la messagerie électronique et la communication de groupe.

2. Messagerie Electronique et Recommandations X.400

La messagerie électronique fournit un nouveau moyen de communication ([Feldman 86]). Elle permet aux utilisateurs d'ordinateurs d'échanger des messages. Généralement, un système de messagerie est construit au dessus d'un réseau d'ordinateurs qui assure le transfert des informations brutes entre ordinateurs. Conceptuellement, un système de messagerie comprend des éléments qui assurent le transport de messages et d'autres éléments qui fournissent des services de messageries spécifiques aux utilisateurs, e.g., le courrier inter-personnel. Pour la bonne coopération entre ses composants, chaque système est régi par des protocoles de communication.

Une dizaine d'années de recherche et de développement dans la communauté informatique ont introduits de nombreux systèmes de messagerie, comme le système de messagerie Internet (cf. [Allman 86]), le système de messagerie Andrew (cf. [Rosenberg 88]), ou Profs d'IBM, etc. Cependant, les utilisateurs de différents systèmes ne peuvent pas facilement communiquer entre eux à cause de conceptions et protocoles différents de ces systèmes. De plus, un système donné fournit souvent des services qu'un autre n'a pas. Un standard de système de messagerie est donc nécessaire.

2.1. Système de Messagerie X.400

En 1984, le CCITT a proposé une série de recommandations qui définissent un système de messagerie en mode "*enregistrement* et *retransmission*" (Store and Forward en Anglais). Ce système de messagerie est souvent appelé **X.400 MHS** (pour Message Handling System) ou **messagerie X.400** ([X.400 84]). Grâce aux expériences d'implantation et d'utilisation effectuées par différents organismes scientifiques et administratifs, le CCITT a publié, avec l'ISO, une nouvelle version des recommandations X.400 en 1988, appelée X.400-88 (CCITT [X.400 88]) ou MOTIS (l'appellation des recommandations de l'ISO). Les recommandations X.400 ont défini un service de transfert de messages et un service de courrier inter-personnel.

2.1.1. Modèle Fonctionnel du Système de Messagerie X.400

Le système de messagerie X.400 comprend des ensembles d'*utilisateurs*, d'*Agents Utilisateurs*, de *MTA* (Message Tranfer Agent), d'*Unités d'Accès* et de *Stockages de Messages*.

La figure 1-1 (issue de la norme X.400) illustre la vue fonctionnelle du modèle X.400 décrivantles éléments du système et les relations entre eux.

> Dans ce modèle, un *utilisateur* est un individu ou une application informatique. Les utilisateurs sont soit des utilisateurs directs (i.e. utilisant directement le service fourni par le MHS), soit des utilisateurs indirects (qui participent au système de messagerie à travers un autre système de communication, comme le courrier classique des PTT). Un utilisateur est désigné comme *expéditeur* lorsqu'il envoie des messages et *destinataire* lorsqu'il reçoit un message.

Un *Agent Utilisateur* (UA ou User Agent) est un processus d'application qui permet à l'utilisateur de préparer, émettre et recevoir des messages. L'UA dialogue avec le système de transfert de messages MTS (pour Message Transfer Systm) ou un Système de stockage de Messages MS (pour Message Store), et leur soumet des messages provenant de l'expéditeur. Un UA reçoit des messages venant du MTS directement, ou par l'intermédiaire d'un MS.

Le *MTS* remet les messages qui lui ont été soumis à un ou plusieurs destinataires, unités d'accès (AU pour Access Unit), et/ou MS. Il est aussi capable de retourner des notifications de remise à l'expéditeur si nécessaire. Le MTS se compose d'un ensemble de *MTA*, qui coopèrent pour transférer et remettre un message aux destinataires spécifiés. Le transfert de messages suit le mode **enregistrement et retransmission**. C'est à dire qu'il n'y a pas de connexion de *bout en bout* entre les correspondants; le message est transféré de proche en proche depuis le MTA de l'expéditeur jusqu'à celui du destinataire.

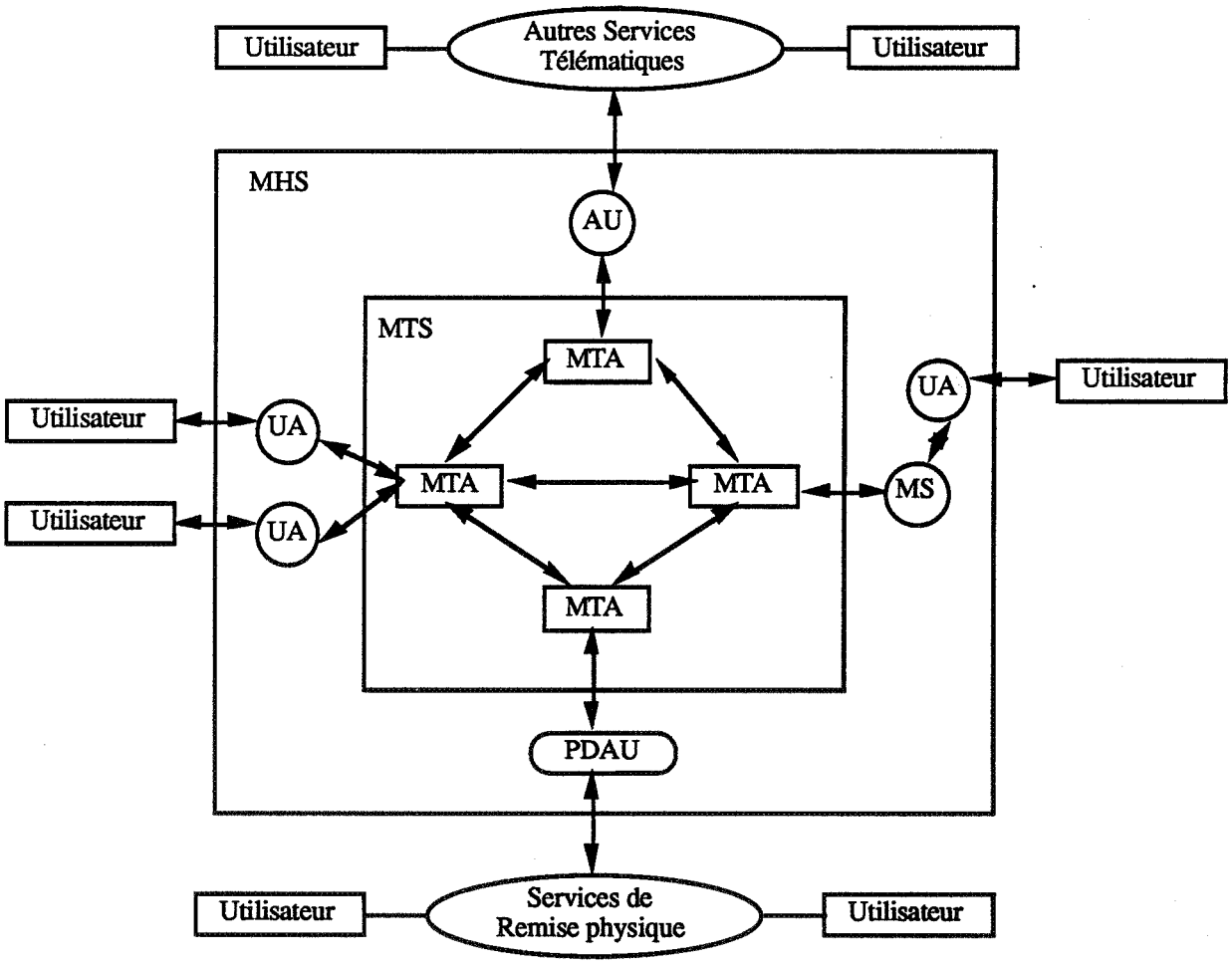


Figure 1-1. Le Modèle Fonctionnel du MTS

Une *Unité d'Accès* est l'entité qui permet aux utilisateurs indirects d'accéder au MHS. La remise de messages aux utilisateurs indirects est accomplie par les AU. L'Unité d'Accès PDAU (Physical Delivery Access Unit) remet des messages de MHS aux utilisateurs qui n'ont pas d'accès directs au MHS via les courriers normaux.

Un *Stockage de Messages* est une unité fonctionnelle du MHS. Il est optionnel dans le modèle. Un MS est dédié à un seul utilisateur, il sert d'intermédiaire entre l'UA et le MTS. Son utilisation principale est de stocker et ensuite de permettre de lire et rechercher les messages destinés à l'UA. Il accepte aussi la soumission indirecte de l'UA.

La norme a aussi défini le *format des messages* véhiculés par le MTS (figure 1-2) :

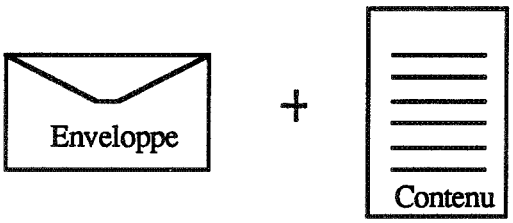


Figure 1-2. Message MTS

Un message est composé d'une enveloppe et d'un contenu. L'enveloppe contient un ensemble de champs, par exemple, l'expéditeur, les destinataires, la priorité du message, etc., dont certains sont utilisés par le MTS pendant le transfert des messages. Le contenu représente l'information que l'UA expéditeur veut remettre aux destinataires.

Les services spécifiques au-dessus du MTS peuvent être construits au moyen de classes spécifiques d'UA coopérants. Les UA coopérants d'une même classe possèdent leur propre protocole de communication et leur propre format de messages; ils utilisent le service de transfert du MTS et fournissent un service spécifique de messagerie.

2.1.2. Le Service de Courrier Inter-Personnel

Le service du courrier inter-personnel (IPM pour Inter-Personnal Messaging) est défini au dessus du MTS pour fournir le service du courrier classique. Il utilise la capacité du MTS à envoyer et recevoir des messages. Il permet aussi à un utilisateur de communiquer avec les autres utilisateurs du service en bénéficiant du service spécifique de l'IPMS (Inter-Personal Messaging System), par exemple, demander un acquittement automatique, demander une réponse du destinataire, faire référence à un autre message, etc. Le modèle fonctionnel du service IPMS est illustré dans la figure 1-3.

Le service du courrier inter-personnel est fourni par les UA et AU spécifiques au service. Les UA utilisés par le service IPMS forment une classe spécifique d'UA coopérants, appelés IPM-UA. Les unités d'accès TLMA et TLXAU permettent aux utilisateurs de Télétext et de Télex de communiquer avec le service IPMS. L'unité d'accès PDAU fournit la possibilité aux utilisateurs de l'IPMS d'envoyer des messages aux utilisateurs qui n'ont pas d'accès au MHS (par exemple, courrier normal, télécopie).

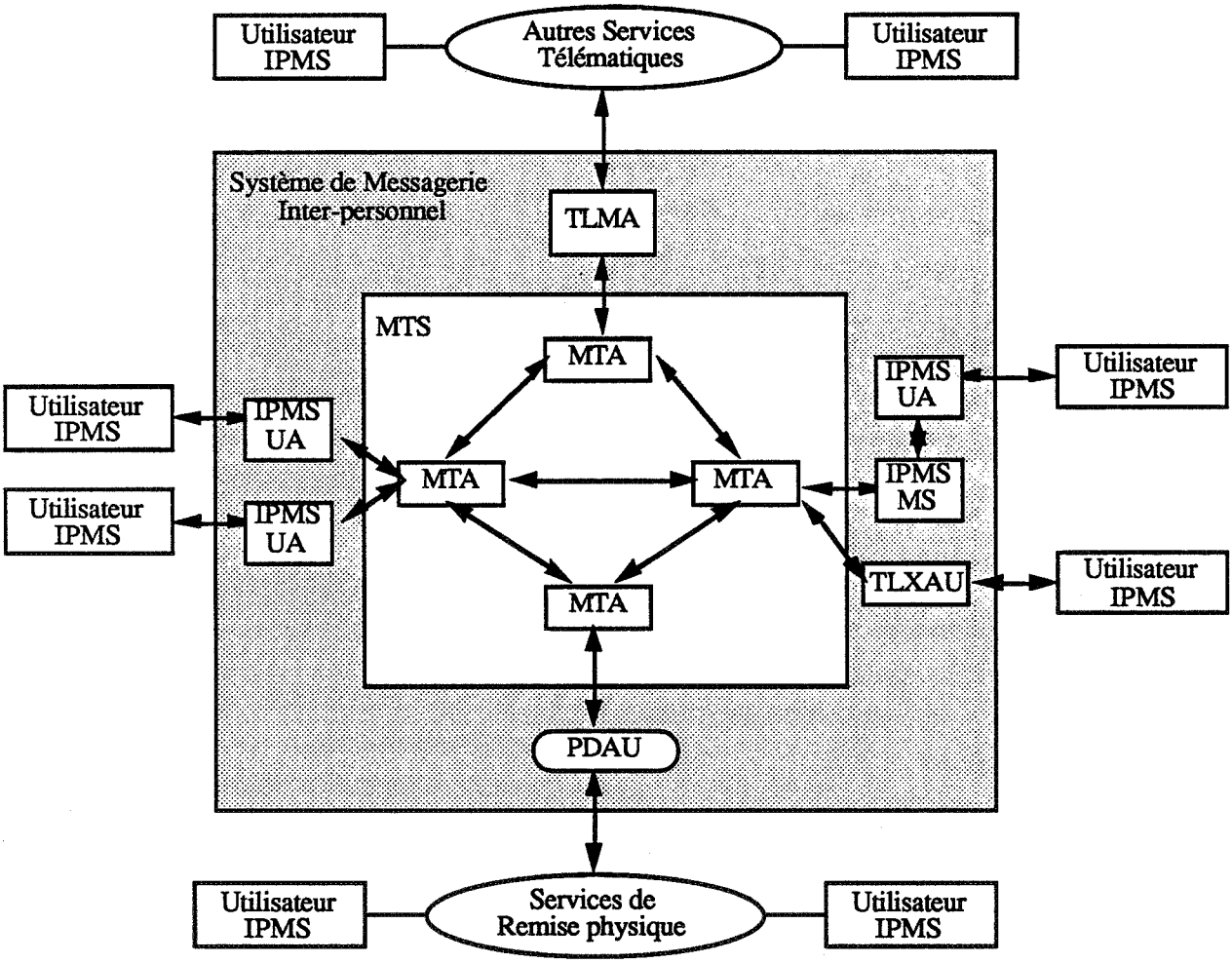


Figure 1-3. Le modèle Fonctionnel de l'IPMS

Les messages IP (Inter-Personnel Messages) sont véhiculés par les messages du MTS, et sont traités comme leurs contenus. Un IPMessage est composé d'un *en-tête* (Heading) et d'un *corps* (Body en anglais). L'en-tête correspond à l'en-tête d'une lettre sur papier: une collection de champs/attributs comme "expéditeur", "destinataires", "destinataires-en-copie", "sujet", "importance", etc. Le corps contient l'information que l'utilisateur veut communiquer. Le corps se compose d'une séquence de parties de corps (Body-Parts); chacune d'entre elles peut être du texte, du son, du graphique, de l'image, du fax, etc. La figure 1-4 donne un exemple de messages inter-personnels.

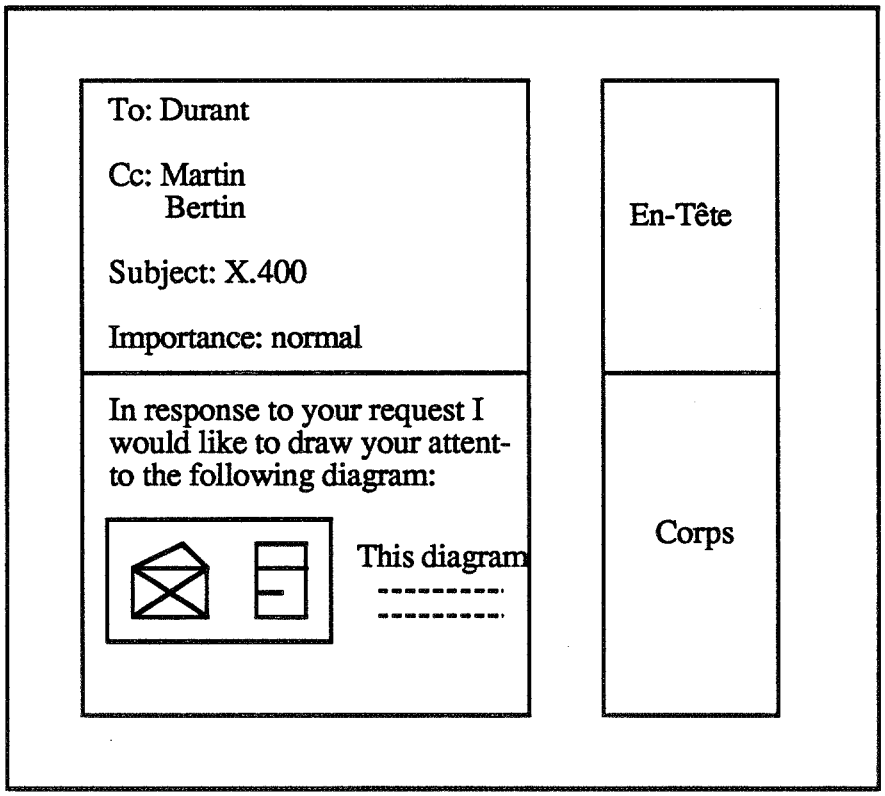


Figure 1-4. Un Message Inter-personnel

3. Communication de Groupe et Modèle d'Activité AMIGO

Nous avons vu que le courrier inter-personnel ne supporte que la simple relation entre les utilisateurs/communicateurs, i.e. *expéditeur <-> destinataire(s)*. Le flux d'information n'est pas contrôlé (cf. [Pankoke 89]). Cependant, la réalité sociale impose des relations beaucoup plus complexes. La communication a souvent pour cadre un travail coopératif. Les relations entre les communicateurs sont souvent hiérarchisées et le flux d'information est contrôlé suivant les relations entre ces communicateurs ou leurs rôles dans le travail coopératif.

Par exemple, une conférence (cf. [Brun 87]) a besoin de coopération entre ses participants. La communication est structurée et contrôlée de telle sorte que seul un sous-ensemble de participants soumet ses contributions (messages ou fiches) à la conférence. Ces contributions peuvent ensuite être lues par les *lecteurs* de la conférence. Un participant qui n'est pas *écrivain* (qui n'a pas le droit de soumettre directement ses contributions à la conférence) peut ajouter ses contributions dans la conférence via un *éditeur*. Pour une activité de *vote*, les participants coopèrent pour obtenir le résultat du vote. Les fonctions des participants sont différentes selon leurs rôles, par exemple, *initiateur*, *électeur*, *coordinateur*, etc; le flux d'information (messages) est contrôlé selon les rôles (expéditeurs ou destinataires) et les types de messages. L'édition collective d'un document et la planification d'une réunion sont d'autres exemples de communication de groupe.

Dans le reste de cette thèse, nous utiliserons l'abréviation **GC** (Group Communication) pour Communication de Groupe, les applications supportant la communication de groupe sont ainsi des *applications GC*.

3.1. Modèle d'Activité AMIGO

Un sous-groupe du groupe AMIGO (*AMIGO Advanced*) s'attache à développer les concepts évolués pour la communication de groupe dans un environnement informatique (cf. [Pankoke 89]). Un modèle d'activité a été élaboré pour modéliser et décrire de façon abstraite une activité de communication de groupe dans un environnement bureautique. Ce modèle a ensuite été vérifié par des exemples d'utilisations ([Brun 87], [Prinz 88]). La description détaillée du modèle est donnée dans [Pankoke 89] et [Danielsen 86]. Nous présentons ici simplement les concepts principaux.

- **Activité**: Une activité de communication de groupe (sans autre précision *Activité* signifie *Activité de Communication de Groupe*) est définie comme un processus de communication parmi un groupe de participants/communicateurs pour atteindre un but spécifique. L'édition collective d'un document en est un exemple. Chaque instance d'une activité a un groupe de participants qui sont des communicateurs, i.e. humains ou programmes d'ordinateurs capables d'échanger de l'information.

- **Rôles**: les participants d'une activité sont classés par *rôles*. Un rôle correspond à un ensemble de fonctions qu'un participant doit effectuer. A chaque instant, pour chaque rôle, il y a zéro, un ou plusieurs participants qui jouent le rôle, i.e. qui sont considérés comme instances du rôle.
Par exemple, dans une activité de vote, *initiateur, coordinateur,* etc. (cf. chapitre 9), sont des rôles.

- **Messages**: Ils définissent les types ou classes de messages échangés. Un type spécifie la structure et les informations qui doivent figurer dans les messages du type. Ces informations sont définies comme attributs pour chaque message.

- **Fonctions**: Elles décrivent les actions des communicateurs, par exemple, envoyer, recevoir, répondre ou acquitter un message.

- **Règles**: Les règles précisent les fonctions licites sur les messages, compte tenu des types de messages et des rôles des correspondants. Elles spécifient les procédures actuelles pour exécuter une activité.

- **Attribution des rôles**: Elle spécifie comment les rôles sont affectés aux différents communicateurs. En particulier, quels sont les communicateurs qui peuvent jouer un rôle, si une instance d'un rôle est obligatoire, si le nombre d'instances d'un rôle est limité.

A Partir des descriptions précédentes, on peut constater qu'un archiveur de messages est nécessaire pour stocker tous les types de messages échangés dans une activité. Il est partagé par tous les participants de l'activité qui jouent différents rôles, et qui ont différents droits et fonctionnalités vis-à-vis des accès dans le stockage.

4. Introduction aux Normes

Dans ce paragraphe nous allons présenter en grandes lignes quelques normes utilisées dans la conception et la spécification d'archiveurs de messages.

4.1. Conventions de Définition de Service Abstrait

Les conventions de définition de service abstrait (Abstract Service Definition Conventions) définissent un langage qui est un outil de description abstraite des systèmes informatiques distribués (DIP pour Distributed Information Processing Systems). Notons qu'une description abstraite d'un système sépare complètement la spécification du service fourni par le système de sa réalisation.

Les conventions sont les résultats d'un effort commun du CCITT ([X.407 88]) et de l'ISO ([ISO DIS 8505/4]). Elles sont incluses dans la version 88 des recommandations X.400 (présentées dans X.407) pour décrire le MHS. Il est évident que l'on ne peut pas présenter en détail ces conventions. Seules quelques idées importantes sont exposées ci-dessous.

Pour chaque système, les conventions permettent de définir le *modèle abstrait* (Abstract Model), qui spécifie les composants d'un système DIP et leurs relations, et le *service abstrait* (Abstract Service), qui définit de façon abstraite le service fourni par le système.

4.1.1. Le Modèle Abstrait

Pour décrire un système DIP, on utilise des tâches et des modules, ou plus généralement des *objets*; et des relations et liens , ou plus généralement des *portes* qui permettent de communiquer avec le monde extérieur.

Le modèle abstrait est une vue *macroscopique* du système DIP, qui spécifie les composants du système, i.e. les **objets** et leurs relations dénommées **portes**. Donc, le modèle abstrait définit une liste d'objets abstraits et les portes associées à chaque objet abstrait. Ce processus peut être récursif: les composants d'un objet sont aussi objets.

Un objet abstrait (objet) est une entité fonctionnelle, qui peut être n'importe quel système/logiciel informatique avec une fonctionnalité indépendante; par exemple, un MTA, un UA, ou un MS sont définis comme objets dans la messagerie X.400. A chaque objet est associé par définition une liste de portes permettant à l'objet de communiquer avec les autres objets. Par exemple, un MTA possède une porte de *soumission*, une porte de *dépôt* et une porte de *transfert*. La définition d'un objet indique aussi les rôles que l'objet joue à travers chacune de ces portes, i.e. *consommateur* et/ou *producteur* des services fournis à travers la porte. Des objets sont de différents types définis par le **OBJECT MACRO** (cf. [ASN.1]), qui spécifie les portes abstraites et les rôles de l'objet dans chacune des portes, pour chaque type d'objet.

Une porte abstraite est le point d'interaction entre objets. Une porte détermine les types d'interactions, i.e. *symétrique* (toutes les instances d'une porte sont identiques) ou *asymétrique* (chaque instance de la porte est soit **consommatrice**, soit **productrice**). Les portes sont aussi typées et définies par le **PORT MACRO** introduit par les conventions.

Deux objets interagissent par deux portes du même type. Lorsque le type est asymétrique, une porte doit être *consommatrice* et l'autre *productrice*. Chaque porte spécifie une liste d'opérations abstraites exécutées via cette porte.

Les composants d'un système DIP et leurs relations sont spécifiés par des **raffinements**, qui permettent de décomposer les objets en sous-objets et de spécifier les relations entre eux. Puisqu'un système DIP est lui-même un objet, ce mécanisme permet de définir complètement le modèle abstrait du système. Les raffinements sont définis en utilisant **REFINE MACRO**.

4.1.2. Le Service Abstrait

Le Service Abstrait est une description *microscopique* du système DIP. Il décrit comment le système est initialisé, contrôlé et terminé.

Le service offert par un objet est défini comme une liste d'opérations que l'objet peut exécuter après l'invocation par un autre objet, via les portes. La tâche exécutée après l'invocation d'une opération est une *procédure (abstraite)*. L'objet qui émet la requête est l'*invoker* et celui qui exécute en réponse à la requête est le *performer*.

Les services communs fournis par tous les objets sont les opérations de base: BIND, UNBIND, et ERROR. Seuls les arguments et résultats de ces opérations sont différents. L'opération BIND relie une ou plusieurs paires de portes entre l'objet *initiateur* et l'objet *répondeur*. L'opération UNBIND déconnecte une paire de portes entre l'initiateur et le répondeur. Une erreur abstraite est une condition exceptionnelle qui apparaît pendant l'exécution d'une opération abstraite.

Le *vrai* service fourni par un objet est une liste d'opérations abstraites qui sont définies par **OPERATION MACRO**. Une opération abstraite est une procédure qui peut être invoquée dans le contexte de deux portes reliées.

Concrètement, la définition du service abstrait offert par un objet consiste à spécifier une liste de portes et les opérations associées avec chacune des portes, dont l'*invoker* est indiqué comme *producteur* ou *consommateur*. Les objets d'information concernés (arguments, ...) sont définis en ASN.1.

Pour plus d'informations, voir [X.407 88] ou [Delgado 88].

4.2. ASN.1

ASN.1 est l'abréviation d'*Abstract Syntax Notation One*. ASN.1 est le standard pour la description et représentation des structures de données et des types d'objets d'information, notamment, pour spécifier les données de protocoles de communication. ASN.1 a été proposé par le CCITT ([X.208 88], [X.209 88]) et l'ISO ([ISO DIS 8824]). Il est dérivé de la norme X.409 dans la version 84 des recommandations X.400 ([X.409 84]). Cette dernière définit le langage de description des données de protocoles, ainsi que les règles de codage et décodage utilisées dans X.400 MHS de la version 1984. Les recommandations de 1988 utilisent ASN.1 pour définir toutes ces données.

Le but de ce standard est de proposer un langage de définition de types de données complexes indépendamment de leurs représentations physiques, et de fournir les règles de codage et décodage des types de données, indépendamment de l'environnement matériel et logiciel (le processeur, le langage de programmation et le compilateur utilisés). Ces deux parties sont logiquement indépendantes; elles sont respectivement définies dans deux différentes normes: [X.208 88] (le langage de description) et [X.209 88] (les règles de codage).

Ce langage de description de types de données:

• permet de regrouper des déclarations en **modules**. Les types et les données sont repérés par leurs *identifiants*. On peut exporter ou importer des identifiants en indiquant *Export* ou *Import* dans le module. Ce qui nous donne un mécanisme pour limiter la visibilité d'un identifiant, ainsi que des types de données et des valeurs.

• Associe à chaque type de donnée une *étiquette* qui se compose d'une classe et d'un numéro. La classe définit la *portée* du numéro. Les classes prédéfinies sont:

- UNIVERSAL: Cette classe ne peut être utilisée que par le standard lui-même; tous les types de cette classe sont les mêmes pour toutes les utilisations.

- APPLICATION: Un type de cette classe est unique à l'intérieur de l'application. Les types de cette classe sont conçus par les applications indépendantes ou les autres recommandations.

- CONTEXT: Les types de cette classe ne peuvent être identifiés qu'à l'intérieur d'une structure ou d'un bloc par le numéro associé.

- PRIVATE: Cette classe est réservée pour des utilisations locales.

• Définit un ensemble de types de base pour la classe UNIVERSAL:

Numéro	**Type**
1	BOOLEAN
2	INTEGER
3	BIT STRING
4	BYTE STRING
5	NULL
6	OBJECT IDENTIFIER
7	REAL
8	ENUMERATED

Les **formes** de ces types de base sont *primitifs*.

• fournit les moyens de définir des types de données complexes à partir des types de base. Un ensemble de constructeurs et un mécanisme d'étiquetage sont à la disposition des utilisateurs:

- SEQUENCE: à partir d'une liste ordonnée de types existants, il forme un nouveau type dont une valeur est une **séquence** ordonnée de valeurs, chacune d'un type existant.

- SET: à partir d'une liste de types existants, il forme une valeur du type construit comme un **ensemble** non-ordonné de valeurs, chacune d'un type existant.

- SEQUENCE OF: à partir d'un type existant, le nouveau type formé avec ce constructeur a comme valeur un ensemble ordonné de zéro, une ou plusieurs valeurs dont chacune est du type existant.

- SET OF: à partir d'un type existant, une valeur du type construit est formée comme un **ensemble** non-ordonné de valeurs, chacune du type existant.

- CHOICE: à partir d'une liste de types existants, une valeur du type formé par le constructeur est celle d'un type quelconque parmi les types donnés.

- SUBSET: à partir d'un type, les valeurs du nouveau type forment un sous-ensemble de celles du type donné, satisfaisant les conditions imposées sur les valeurs ou les relations entre valeurs.

Les types ainsi formés sont *constructeurs*, i.e. leur forme est du type *constructeur*.

Le mécanisme d'étiquetage permet d'avoir un nouveau type en associant une étiquette différente à un type existant.

• fournit un ensemble de types prédéfinis à partir des types de base. Ils sont aussi de la classe UNIVERSAL:

Numéro	**Type**
18	NumericString
19	PrintableString
20	TeletexString
21	VideotexString
22	IA5 String
25	Graphic String
26	Visible String
27	General String

• fournit un mécanisme d'**Extension**, appelé **MACRO DEFINITION**, qui permet d'introduire de nouveaux mots-clés et de nouvelles syntaxes. Ce mécanisme est utilisé par de nombreuses normes. Par exemple, OBJECT, PORT, etc. sont définis en utilisant ce mécanisme.

Bien que la description ASN.1 soit indépendante du codage et du décodage utilisés, une représentation physiquement commune des données est nécessaire pour les échanger. Ainsi, un ensemble de règles de codage et de décodage sont définies [X.209 88]. Elles permettent d'encoder une description ASN.1 en une forme indépendante de la machine et de l'environnement logiciel utilisé. Le codage est basé sur le format triplet <Type, Longueur, Valeur>, i.e. chaque valeur d'un type ASN.1 est codée comme le montre la figure 1-5 :

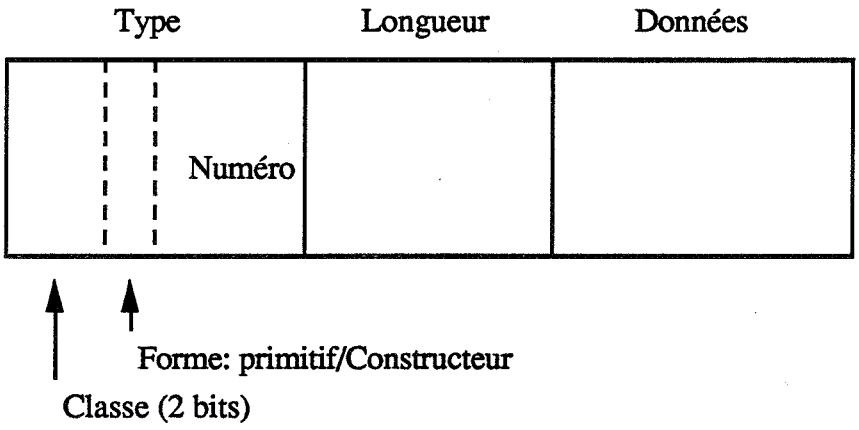


Figure 1-5. Le codage d'une valeur ASN.1

Ce schéma de codage et de décodage permet de transférer des données structurées (notamment les unités de données de protocoles) entre des environnements et/ou machines hétérogènes.

On donne ici un exemple pour terminer notre présentation de ASN.1. L'exemple définit un type de données composé d'un nom de personne, et d'un indicateur spécifiant si la personne est mariée:

Définition du Type:

```
Personne ::= SEQUENCE {
                nom IA5 String,
                mariée BOOLEAN}
```

La valeur {nom "Smith", mariée TRUE} représente la personne du nom Smith et mariée, elle peut être codée comme:

Séquence	Longueur	Contenu		
30h	0Ah			
		Chaîne-IA5	*Longueur*	*Contenu*
		16h	05h	536D697468h
		Booléen	*Longueur*	*Contenu*
		01h	01h	FFh

La première ligne code la personne (une SEQUENCE); la deuxième et la troisième codent respectivement deux composants de la personne: le nom et l'indicateur, ces deux derniers sont codés comme le contenu de la SEQUENCE. Le codage est représenté en hexadécimal.

4.3. Service de Directory X.500

La norme X.500 est le standard du *service de directory*, proposé par le CCITT ([X.500]) et l'ISO ([DIS 9594]). Son but est de faciliter l'interconnexion de systèmes informatiques et de fournir un service d'annuaire global international. On donne ici ses caractéristiques principales.

4.3.1. Le Modèle d'Information

Le service de directory est fourni par le **Système d'Annuaire** (ou de **Directory** en Anglais, DS pour Directory System). Ce système d'annuaire gère une base d'information pour des *entités de communication* qui peuvent être humaines, groupes de communication, listes de distributions, archiveurs de messages, etc. Chaque entité de communication est représentée par une **entrée** dans la base, contenant les informations sur l'entité représentée. L'ensemble de toutes les entrées forme la base d'information du système d'annuaire, cette base est appelé **DIB** pour Directory Information Base.

Chaque entrée contient un ensemble d'**attributs** représentant les propriétés de l'entrée, par exemple, le nom de famille et le prénom pour une entrée représentant un être humain. Chaque attribut a un *type* (Attribut Type) spécifiant le type d'information représentée et une ou plusieurs valeurs contenant l'information. Le *type* est globalement unique et est représenté par un *ObjectIdentifier* d'ASN.1.

Les entrées sont groupées en **classes** basées sur les types des entités de communication qu'elles représentent. Un attribut du type *ObjectClass* est associé avec chaque entrée indiquant la classe.

4.3.2. Nommage

Le schéma de nommage du service de directory suit une structure hiérarchique. Les entrées sont rangées dans un arbre correspondant aux relations organisationnelles entre les entités de communication qu'elles représentent. Cet arbre est appelé **DIT** pour Directory Information Tree (Figure 1-6).

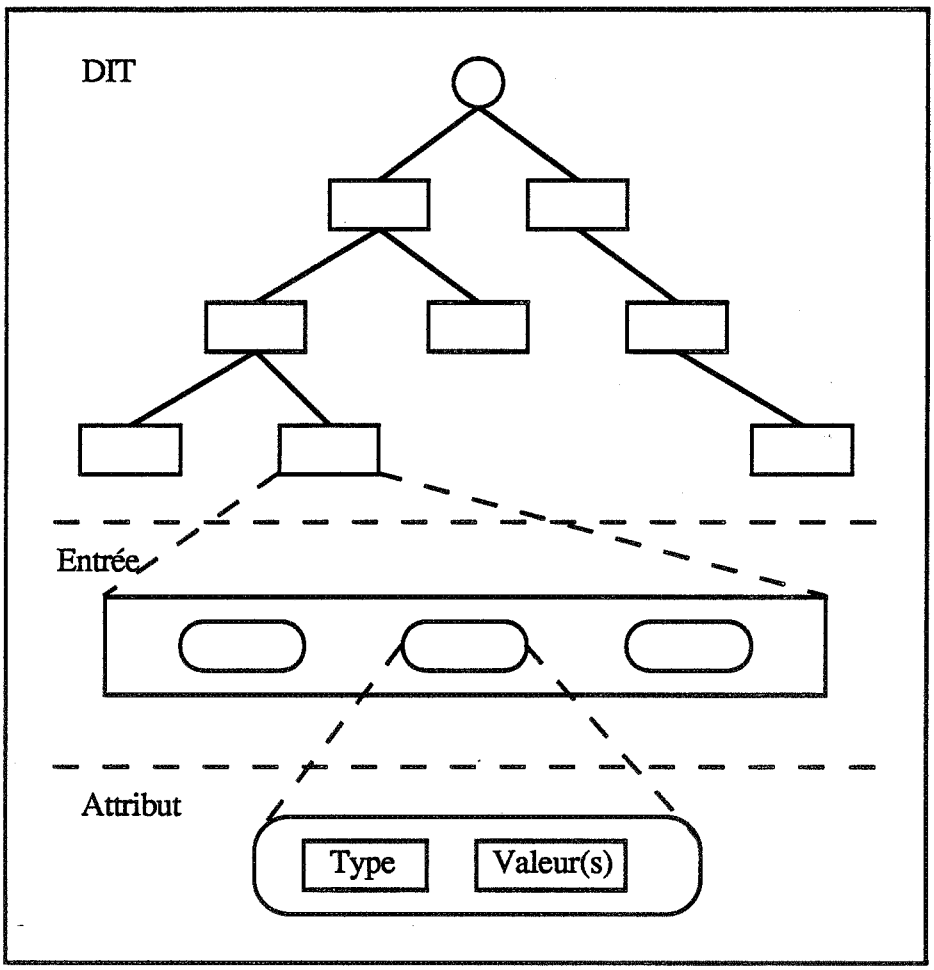


Figure 1-6. La structure du DIT et d'Entrées

A chaque noeud (une entrée) du DIT est associé un **RDN** (pour Relative Distinguished Name) composé d'un sous-ensemble d'attributs (appelé Distinguished Attributes) de l'entrée. Le RDN d'un noeud permet d'identifier le noeud parmi ses *frères*. L'affectation du RDN pour une entrée est faite par l'autorité de nommage représentée par son père (le noeud au-dessus dans le DIT), la responsabilité de nommage est donc *distribuée.*

Le nom globalement unique d'une entrée est formé de la séquence ordonnée des RDN des noeuds dans le chemin du DIT à partir de la racine jusqu'à l'entrée. Ce nom est appelé **DN** pour Distinguished Name, il est unique dans la DIB.

Une entrée peut avoir des *alias* (surnom en Français) qui sont obtenus en créant des entrées spéciales appelées **entrées d'alias** (ou alias entry) liées à l'*entrée d'objet* (*objet* par rapport à *alias*). Ceci fournit un mécanisme pour avoir des noms alternatifs.

Un processus appelé *Name Verification* permet de vérifier si un nom composé d'un ensemble ordonné d'attributs est valide ou non dans le DIT.

Un ensemble de règles de définition et de contraintes appelées **schémas** définit la structure de DIT, les classes d'entrées, les types d'attributs et leurs syntaxes. Pour chaque classe d'entrées,

le schéma spécifie les attributs obligatoires et les attributs optionnels pour les entrées de la classe. Il spécifie aussi la règle de comparaison des valeurs pour chaque type d'attributs.

4.3.3. Service Abstrait

Un ensemble de portes et d'opérations associées a été défini dans le standard, qui spécifie le service fourni par le DS. Les opérations forment le *Directory Access Protocol* (DAP) et fournissent aux utilisateurs les fonctionnalités pour *lire*, *chercher* et *modifier* de l'information dans la DIB. En particulier, un utilisateur peut ajouter, remplacer ou supprimer des attributs d'une entrée. Les portes et les opérations sont:

Porte	**Opération**
Read	Read
	Compare
	Abandon
Search	List
	Search
Modify	Modify Entry
	Add Entry
	Remove Entry
	Modify RDN

4.3.4. Modèle Fonctionnel et Opérations Distribuées

Le système d'annuaire est distribué; il est fourni par un ensemble d'entités d'applications appelées *Directory System Agents* (DSA). Chacun gère une partie du DIT. Chaque DSA s'interface avec des agents utilisateurs du service de directory (DUA) et coopère avec un ou plusieurs autres DSA. Un utilisateur accède au service de directory via un DUA. Le modèle fonctionnel est montré dans la figure 1-7:

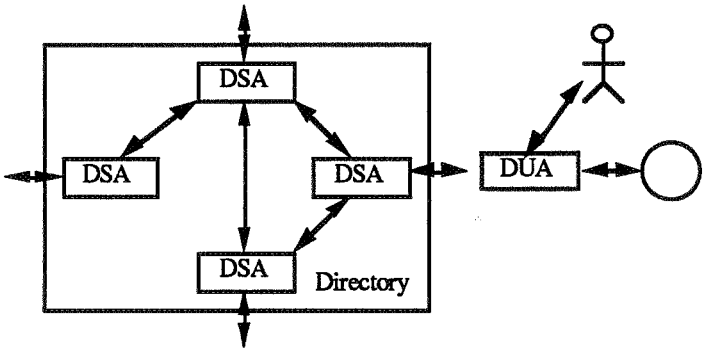


Figure 1-7. Le modèle Fonctionnel du Service de Directory

Chaque DSA peut utiliser plusieurs modes d'interaction, ce sont:

• **Chaining**: le DSA passe les opérations qu'il ne peut pas exécuter (par exemple, lire une entrée qui n'est pas dans la fraction de la DIB gérée par le DSA) à un autre DSA. Ce processus peut être récursif.

• **Multi-Casting**: identique au *chaining*, sauf que le DSA passe une opération manipulant une entrée inexistante dans le DSA à plusieurs autres DSA simultanément ou séquentiellement. Notons qu'au plus un seul DSA peut donner le bon résultat.

• **Referral**: le DSA négocie avec les autres DSA pour obtenir le nom et la localisation d'un autre DSA que l'on doit contacter pour manipuler une entrée qui n'existe pas dans le DSA en question.

Le standard permet aux DSA de choisir leur méthode d'interaction parmi l'ensemble de choix possibles.

Chapitre 2
Archiveurs de Messages

1. Rôle d'un Archiveur de Messages

Le but commun dans toutes les communications est d'**échanger**, afin de **partager** de l'information entre les partenaires des communications. L'information échangée et partagée existe sous forme de *documents* pouvant être assez compliqués, contenant du texte, du graphique, de l'image, du son, etc. Pour les communications en mode messagerie, l'information échangée est véhiculée par les messages, i.e. transférée sous forme de messages.

Donc il est clair qu'un archiveur de messages (partagé) est nécessaire pour **toutes** les applications de communication de groupe comme pour la messagerie inter-personnelle. En fait, un archiveur de messages peut être utilisé dans une application MHS et GC comme :

- Un outil d'*archivage* qui archive les messages échangés. Par exemple, un nouveau membre d'une conférence pourra ainsi obtenir tous les anciens messages de la conférence.
- Un outil de *communication* puisque l'archiveur est partagé: un utilisateur stocke un message qui pourra ensuite être lu par d'autres.

La messagerie inter-personnelle peut utiliser un archiveur comme ([Palme 85]):

- Un outil personnel de *sauvegarde* dans le cas de perte de messages envoyés.
- Un mécanisme de *vérification/authentification.* On pourra utiliser le résultat de la recherche d'un message par un archiveur de messages indépendant et certifié, comme une preuve que le message est déjà envoyé à certains destinataires.

Cependant l'archiveur de messages a ses propres besoins qui ne sont pas satisfaits par les bases de données conventionnelles (voir sections suivantes); ceci implique la nécessité d'avoir un outil universel de stockage/archivage de messages supportant les applications MHS et GC.

Dans le reste de cette partie, nous allons analyser les fonctionnalités nécessaires aux archiveurs de messages. Dans ce but, nous allons d'abord présenter les caractéristiques spécifiques des *messages*, puis des exemples d'archiveurs de messages existants.

2. Caractéristiques des Messages

Un Message est l'unité de transfert (et d'échange) entre les partenaires d'une application MHS et GC. Il est l'unité d'information dans les applications MHS et GC. Différents des objets d'information locaux (par exemple, fichiers) ou des objets d'information bien structurés (une relation dans une base de données relationnelles), les messages ont les caractéristiques suivantes:

- Un message est une unité d'échange dans un groupe de deux ou plusieurs personnes: il y a un expéditeur et un ou plusieurs destinataires.

- Un message contient l'information que l'expéditeur veut véhiculer. De plus, il a d'autres attributs que l'information elle-même, par exemple, l'expéditeur, le(s) destinataire(s), le sujet, l'importance, etc. du message. Un document peut être véhiculé par un ou plusieurs messages.

• Les messages ne sont pas indépendants; ils sont souvent liés entre eux. Par exemple, un message est une réponse à un autre message, ou une nouvelle version d'un autre message.

• Les messages ne peuvent pas être changés une fois qu'ils ont été créés et envoyés, ni par l'expéditeur ni par le(s) destinataire(s).

• La taille des messages n'est pas fixe. Un message peut être long (plusieurs certaines de kilo-octets), ou très court (un ou deux mots). On pourra imaginer qu'un message contient un rapport de recherche et qu'un autre indique une faute orthographe dans le rapport. Une base de données relationnelle classique ne peut stocker les messages de ce genre.

• Les messages ne sont pas *complètement* structurés contrairement aux objets dans les systèmes orientés-objet purs (SmallTalk [Goldberg 83], Eiffel [Meyer 88]). Des messages de même type peuvent contenir différents champs et différents types d'informations. Les différentes applications MHS et GC utilisent des messages de différentes structures.

• Il est possible que les messages soient incohérents vis-à-vis d'un utilisateur ou d'un archiveur de messages à cause de l'insécurité et du délai aléatoire de transfert des messages. Par exemple, un message répondant à un autre message peut parvenir à un utilisateur avant le message auquel il répond. Le message contenant la question peut même être perdu.

3. Etat de l'Art: Exemples d'Archiveurs de Messages

Dans cette section, nous allons présenter quelques archiveurs de messages utilisés dans des systèmes de messagerie et des applications de communication. Les fonctionnalités d'un archiveur peuvent être fournies par un serveur indépendant (Modèle X.413, CRMM, etc) ou intégrées avec le reste du système qui manipule directement le médium de stockage (fichiers, répertoires, disques) et les messages (par exemple, MH, Andrew, ...). Les aspects suivants d'archiveurs de messages sont décrits dans notre analyse:

• *Environnement*: dans lequel se situe l'archiveur par rapport aux autres composants du système, i.e. les relations avec le reste du système.

• *Nombre d'Utilisateurs*: l'archiveur est-il mono-utilisateur ou multi-utilisateurs? La notion de *mono* ou *multi*-utilisateurs signifie que les utilisateurs **partagent** les messages. Il faut noter qu'un serveur de stockage peut gérer plusieurs archiveurs de messages mono-utilisateur, et qu'un système de messagerie peut contenir plusieurs archiveurs de messages: un pour chaque utilisateur.

• *Modèle de Données*: quels sont les objets d'information, comment sont-ils organisés, quels sont les types ou formats de messages supportés? Quelles sont des opérations disponibles pour les manipuler?

• *Méthode d'Accès*: comment un utilisateur accède-t-il à l'archiveur de messages, par la manipulation directe via une interface interactive, ou via un protocole sans connexion en mode messagerie?

• *Fonctionnalités*: Quelles sont les fonctionnalités fournies aux utilisateurs, quelles sont les opérations disponibles pour manipuler des objets d'information (messages, ...)? En particulier, peut-un utilisateur faire la recherche? peut-on faire une lecture sélective des messages? comment est gérée la nouveauté? etc.

• *Contrôle d'Accès*: Un archiveur de messages multi-utilisateurs nécessite un mécanisme de contrôle d'accès, il implique aussi qu'il faille d'abord avoir un modèle d'utilisateurs.

Il faut noter qu'à cause de la complexité des archiveurs de messages, les fonctionnalités de la plupart des archiveurs existants sont souvent limitées volontairement.

3.0. Quelques Notions

Avant de commencer notre analyse, nous allons préciser quelques notions:
Le **service de stockage/archivage** est une vue externe d'un **système de stockage/archivage**, il indique les fonctionnalités du système d'archivage. Le service d'archivage est fourni par un système d'archivage; un système d'archivage est donc un **serveur d'archivage** qui fournit le service de stockage et d'archivage. Un **système d'archivage** pourra être **distribué** (dans ce cas, on dit aussi que le service d'archivage est **distribué**), et se composera alors d'un ou plusieurs **archiveurs de messages** dans un environnement distribué (Figure 2-0). Ces archiveurs sont souvent coopérants, mais ils peuvent aussi être indépendants selon les applications. Le système d'archivage peut être un archiveur quand il contient un seul archiveur. Sans autre précision, la présentation d'un système d'archivage (ou un archiveur si le système ne contient qu'un archiveur) inclut le service d'archivage et les archiveurs individuels. Les **clients** d'un système d'archivage accèdent au système via un des archiveurs du système. Les archiveurs individuels sont les **points d'accès** du système d'archivage.

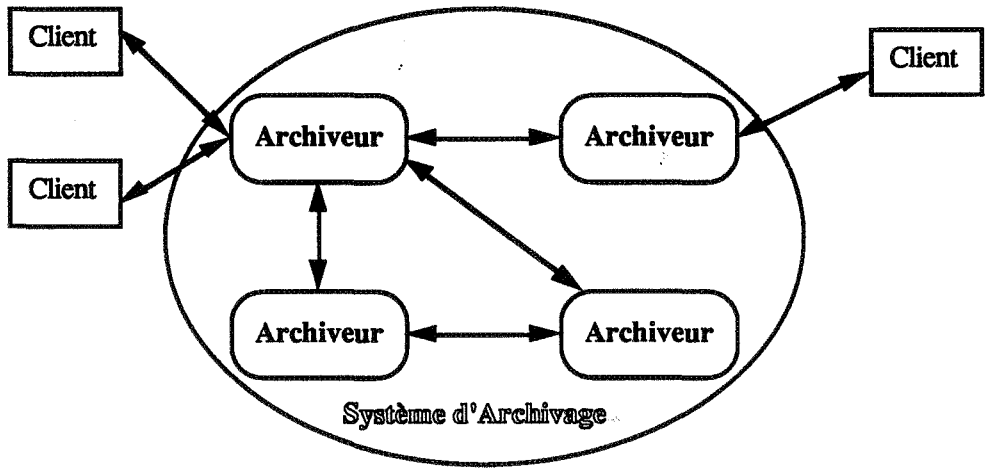


Figure 2-0: Système d'Archivage

3.1. Archiveurs Temporaires Simples

Ce type d'archiveurs se situe entre l'agent de remise de messages et l'interface des utilisateurs (souvent sur une même machine physique) (cf. Figure 2-1). Par exemple, les Boîtes aux Lettres Systèmes dans le système de messagerie standard UNIX (/usr/spool/mail/UserName ou

/usr/mail/UserName) et les Boîtes aux Lettres du X.400 GIPSI [Laborie 87] entre un MTA et ses UA sont des archiveurs temporaires.

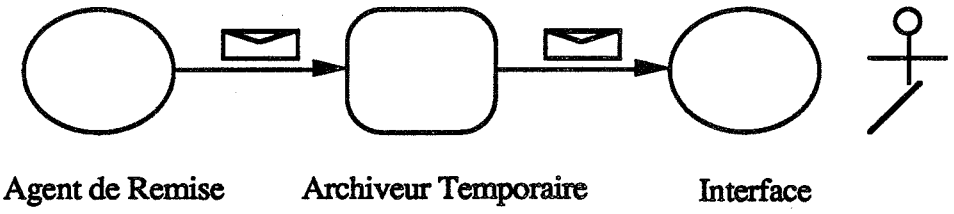


Figure 2-1. Archiveur Temporaire

Chaque archiveur de ce type correspond à un utilisateur et a pour but de stocker temporairement les messages remis à l'utilisateur. L'utilisateur peut lire ces messages via l'interface utilisateur, et ensuite les stocker à plus long terme avec un autre archiveur.

Les messages sont généralement organisés linéairement d'après la date de réception, soit dans un fichier prédéfini dont un segment correspond à un message, soit dans un répertoire prédéfini dont chaque message correspond à un fichier numéroté séquentiellement. Les formats des messages sont généralement déterminés par le système de messagerie, RFC-822 (cf [RFC-822]) pour sendmail, et X.400 P1+P2 ([X.400 84]) pour Gipsi X.400.

L'*agent de remise* et l'*interface* accèdent directement aux messages dans la **boîte** par des opérations "Stocker", "Lire", "Détruire", etc. Une fonction ***Résumer*** est disponible dans ces systèmes pour donner une vue très brève des messages dans la boîte (Expéditeur, Date de réception, Sujet, numéro de message dans la boîte, etc). Aucun moyen d'organisation n'est fourni.

Un contrôle des nouveautés est souvent fourni afin qu'un utilisateur retrouve facilement les nouveaux messages (qui n'ont pas encore été lus par le destinataire). Ceci est fait dans le système *sendmail* en supprimant chaque message lu de la boîte temporaire (il est ensuite stocké ailleurs, ce mécanisme jouant au niveau de l'archiveur et au niveau de l'interface). Dans le système Gipsi, un compteur indique le plus grand numéro de message lu (fait totalement par l'interface).

Ce genre d'archiveur est très limité au niveau des fonctionnalités et de l'organisation. Il nécessite un archiveur *permanent* et *un basculement* entre les deux archiveurs (souvent fait par l'archiveur permanent lui-même). Son grand avantage est sa simplicité: il est facile à réaliser et permet d'utiliser facilement un archiveur de messages sophistiqué comme archiveur permanent. Ce dernier n'est malheureusement pas toujours disponible.

3.2. MH

Le MH ou Message Handler est une interface sophistiquée du système de messagerie UNIX. Elle est souvent utilisée comme l'interface et l'archiveur de messages pour le logiciel *sendmail.*

L'archiveur géré par MH est un archiveur de messages *permanent* et mono-utilisateur. Il est intégré dans MH avec ses fonctions d'interface (figure 2-2).

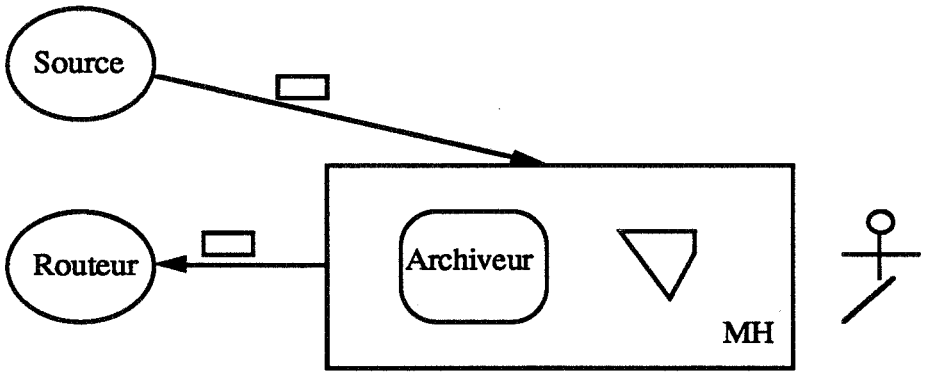


Figure 2-2. MH

Les messages stockés sont au format RFC-822 uniquement (format ASCII avec des paires Mot-Clé/Valeur). Ils sont stockés dans des dossiers (**folders**). Un *folder* contient un ensemble de messages. Il se présente comme un répertoire sous UNIX. Deux folders sont prédéfinis: **inbox** contient tous les messages reçus; une fonction spéciale *inc* (incorporate) permet à l'utilisateur d'extraire les messages de la boîte de réception temporaire et de les stocker dans **inbox**. L'autre *folder* prédéfini est **drafts** qui stocke les messages composés par l'utilisateur avec MH (envoyés ou non).

L'utilisateur peut créer ou détruire des folders, stocker ou supprimer un message d'un folder, déplacer ou copier des messages entre deux folders. Une autre fonctionnalité intéressante est la **recherche** qui permet à l'utilisateur de *sélectionner* des messages dans un folder d'après leurs attributs (e.g., Sujet, Expéditeur, Destinataires, etc). Le résultat de la *recherche* est une liste de numéros des messages qui forme une **séquence**. Il existe une séquence prédéfinie pour chaque *folder*, i.e. **all,** qui est la séquence de tous les messages dans le folder. La recherche est en effet effectuée dans l'une des séquences (par défaut, **all**) du folder, cela permet de faire de la recherche *incrémentale*, puisque le résultat d'une recherche est une séquence dans laquelle on peut effectuer d'autres recherches.

MH est largement utilisé dans la communauté des utilisateurs d'UNIX comme l'interface E_Mail (Mail ou Mailx) associé à la boîte aux lettres personnelle. Une version graphique de l'interface utilisateur existe sous X Windows.

3.3. Service de Stockage de Messages X.400

Les recommandations version 88 de X.400 ont inclus une norme du service de stockage de messages. L'archiveur de messages défini est mono-utilisateur; il se situe entre l'UA et le MTS comme un intermédiaire (figure 2-3). C'est un archiveur temporaire de messages. Sa fonction principale est d'accepter la remise de messages et de les garder (stocker) pour que l'UA de l'utilisateur final puisse ensuite les retrouver et les consulter. Le MS (pour *Message Store*) fournit aussi les services de soumission indirecte de messages et d'administration indirecte. Le but principal d'un tel service est de permettre aux utilisateurs possédant des micro-ordinateurs de pouvoir consulter et envoyer leurs courriers: cela peut se faire en mettant l'UA dans le micro-ordinateur et le MTA dans une machine *centrale* qui gère un archiveur de messages pour l'UA.

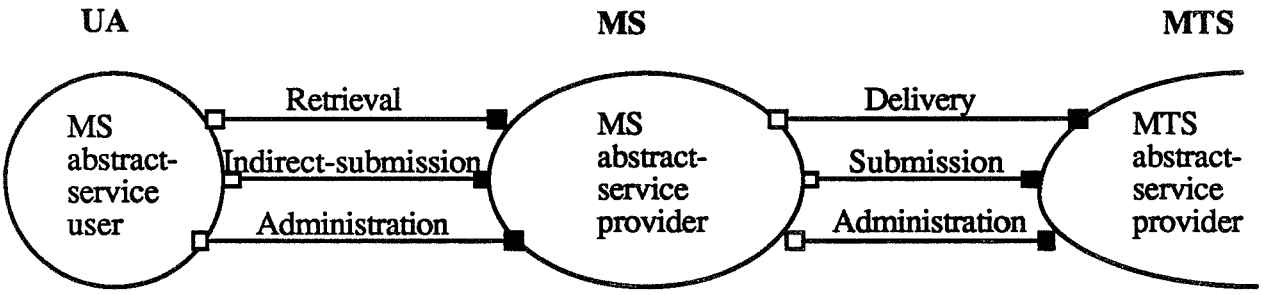


Figure 2-3: Le modèle fonctionnel de MS

Le modèle de MS est montré dans la figure 2-3, ainsi que les relations entre le MS et l'UA, le MS et le MTS. L'UA accède au MS en mode *question-réponse*. Les messages stockés sont dans le format X.400. Ils sont représentés comme des *entrées*. Toutes les entrées forment la **base d'information** du MS. Les entrées sont numérotées dans l'ordre croissant de leur date de réception. Le numéro est l'identificateur de l'entrée dans la base d'information. Chaque message peut être représenté par une ou plusieurs entrées avec une liaison de *parenté*. Par exemple, un message reçu par un UA-IP peut être représenté par une seule entrée, ou par deux entrées liées dont l'une contient l'enveloppe MTS et l'autre la partie IP. Une entrée est composée d'un ensemble d'attributs, par exemple, *expéditeur*, *sujet*, *contenu*, etc.

Les services sont fournis à l'UA via les portes du MS. La porte *Soumission-Indirecte* (*Indirect-Submission*) permet à l'UA de soumettre un message au MTS via le MS. Ce dernier utilise le service de soumission du MTS. La porte *Administration* fournit les fonctions d'administration. La porte *Recherche (Retrieval)* permet à l'UA de *lister*, *rechercher*, *extraire* et *supprimer* des messages.

Le MS se comporte comme un client (consommateur) du service du MTS; trois portes (*Delivery*, *Submission* et *Administration*) connectent le MS avec le MTS. La porte *Delivery* remet les messages destinés à l'UA dans le MS.

Un attribut particulier, *entry-status* (statut d'entrée), est associé à chaque entrée indiquant la nouveauté de l'entrée pour l'utilisateur. L'attribut prend une des trois valeurs **New**, **Listed** ou **Processed** qui signifient respectivement que le message est complètement nouveau, seulement listé dans un résumé ou déjà lu. Le statut est changé automatiquement par le MS en fonction des opérations exécutées par l'utilisateur.

Un inconvénient du MS est qu'il ne fourni pas de moyen efficace d'organisation de messages (folders, etc) et que l'utilisateur ne peut pas attacher des informations supplémentaires (commentaires, mots-clés) aux messages reçus dans le MS.

3.4. Stockage et Recherche de Données dans l'Environnement X.400

Jacob Palme ([Palme 87]) a proposé une spécification de stockage de documents dans l'environnement MHS. Puisqu'un document dans l'environnement MHS est soit un message soit un ensemble de messages liés, on pourra donc considérer la spécification comme une proposition d'archiveur de messages.

Le modèle fonctionnel de la proposition est donné dans la figure 2-4. Un agent de stockage (MSSA pour Message Store Service Agent) gère le stockage et la recherche de messages dans une base de messages. Les agents utilisateurs (UA) de MHS communiquent avec le MSSA en utilisant une extension du courrier IP (les messages IP avec des champs additionnels). La communication est faite en mode messagerie. Le MSSA peut aussi être accédé interactivement par un agent utilisateur dédié (MSUA pour Message Stocke User Agent).

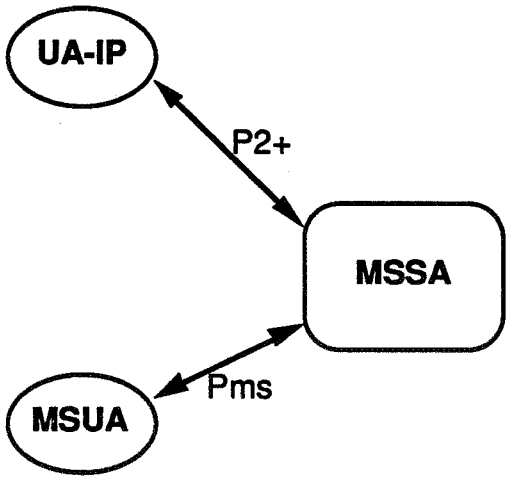


Figure 2-4

Les messages stockés sont des IPMessages avec certains nouveaux attributs proposés dans l'article comme l'extension de P2 (le protocole entre deux IPM-UA dans la version 84). Les messages sont groupés en **ensembles** de messages et sont liés par les relations comme *In-Reply-To* (En-Réponse-à), définies dans le courrier IP, et par les nouvelles relations proposées dans l'article.

L'agent de stockage fournit les fonctions pour *rechercher, extraire, détruire,* ... un ou plusieurs messages. La recherche s'effectue non seulement d'après les attributs mais aussi d'après les relations entre les messages.

Le contrôle d'accès est traité par le système de directory, i.e. les informations de contrôle, comme, par exemple, "ceux qui sont autorisés à stocker un message", "ceux qui sont autorisés à rechercher des messages", etc. sont stockées dans le système de directory. De même, l'archiveur (une entité de communication) ou certains messages individuels assez importants sont enregistrés dans le directory en indiquant leur localisation et les informations de contrôle d'accès.

3.5. COM

COM est un système de conférence développé à l'université de Stockholm en Suède en 1977 ([Palme 85a], [Palme 85b]). C'est un système centralisé qui offre simultanément un service de courrier inter-personnel et un service de conférence.

Le composant principal de COM est un archiveur de messages centralisé qui gère les *messages* et les *activités*. Une activité est liée à un ensemble de messages; elle peut servir comme une boîte aux lettres personnelle ou comme une boîte partagée dans le cas d'une conférence.

Un message est un morceau de texte lié à une ou plusieurs activités. Un seul exemplaire du message est gardé pour toutes les utilisations locales de COM. Les messages peuvent être aussi liés entre eux suivant les relations **InReplyTo** et **CommentOn**, même s'ils appartiennent à des activités différentes.

Un utilisateur peut créer un message et le lier à une ou plusieurs activités. Ainsi un message est communiqué aux propriétaires des activités (si les activités représentent des boîtes aux lettres) ou aux lecteurs des conférences (si les activités représentent les archiveurs de messages des conférences). Un utilisateur peut aussi détruire un message, lier un message à un autre. Une activité peut être créée pour avoir une nouvelle boîte aux lettres ou une boîte partagée de conférence. Le fait qu'un message puisse être lié à plusieurs activités permet de partager dynamiquement des messages entre plusieurs boîtes aux lettres personnelles et des conférences.

Une facilité est donnée aux utilisateurs pour suivre les liens, par exemple, pour trouver un message lié à un message donné.

Alors qu'un utilisateur a tous les droits sur sa boîte aux lettres personnelle, les accès aux conférences sont obligatoirement contrôlés. Pour une conférence, les utilisateurs sont différenciés d'après leurs rôles qui déterminent les fonctions/opérations qu'ils peuvent effectuer. Par exemple, les *lecteurs* d'une conférence peuvent lire les messages de la conférence, les *écrivains* peuvent lier des nouveaux messages à la conférence, etc. Chaque utilisateur a un ou plusieurs rôles qui sont utilisés pour contrôler les accès à la conférence.

Le contrôle de nouveauté est fait de la façon suivante:

- les liens entre une activité et les messages sont numérotés avec un numéro de séquence unique dans chaque activité.

- pour chaque utilisateur, on garde le plus grand numéro de message qu'il a lu, pour chaque activité à laquelle il a souscrit.

- Pour décider si un message dans une activité est nouveau, il suffit de comparer le numéro de séquence du lien entre l'activité et le message avec celui gardé pour la même activité. Pour éviter de voir le même message plusieurs fois dans le cas où le message est lié à plusieurs activités et qu'un utilisateur a souscrit à toutes, la comparaison est faite pour l'ensemble des activités souscrites par l'utilisateur.

3.6. CRMM

CRMM (Conférence Répartie en Mode Messagerie, [Pays 87] et [Brun 87]) est un système de conférence distribuée développé par l'Ecole des Mines de St-Etienne et le Centre National d'Etudes des Télécommunications de France.

Le modèle de CRMM est illustré dans la figure 2-5. Le service de conférence est fourni par un ensemble de BBGA (Bulletin Board Group Agents) coopérants qui communiquent entre eux au protocole P30 en mode messagerie. Ces BBGA sont conceptuellement distribués, chacun sert un ensemble d'utilisateurs *locaux* communiquant avec le BBGA via un agent utilisateur (BBUA pour Bulletin Board User Agent).

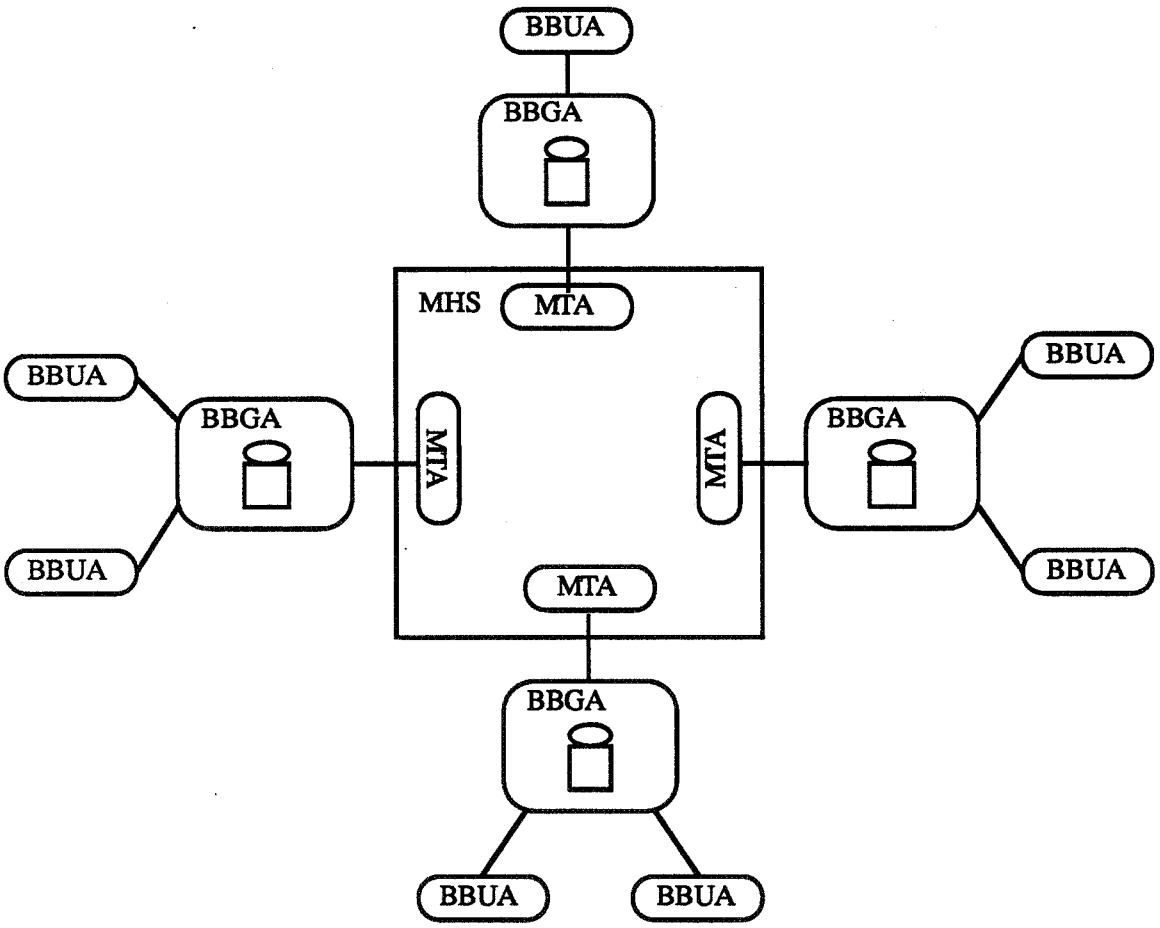


Figure 2-5. Modèle de CRMM

Le service d'archivage de messages est également distribué. En fait, chaque BBGA maintient un exemplaire de l'archive complète.

Les objets d'information de CRMM s'appellent **fiches**; c'est un type spécifique de messages qui se situe au même niveau que les IPMessages. Une fiche se compose d'un ensemble d'attributs prédéfinis par CRMM: *auteur*, *éditeur*, *sujet*, *mots-clés*, etc. Les fiches sont liées entre elles par des liens typés, comme *InReplyTo*, *VersionOf*, etc. Au maximum un lien peut exister entre deux fiches. Une nature est associée à chaque lien spécifiant la sémantique du lien. Les opérations manipulant les fiches respectent les sémantiques des liens. Quatre natures sont spécifiées:

• Séquentiel: un lien séquentiel indique la relation d'ordre entre deux fiches. L'ordre de **lecture** respecte la chronologie d'ajout dans la base de messages. *InReplyTo* en est un exemple, la *question* doit normalement être lue avant la *réponse*.

Vis-à-vis de la **destruction** de la fiche antécédente, la seconde fiche peut perdre sa valeur ou non. Dans le premier cas, le lien est Séquentiel avec Héritage de l'Effacement. *InReplyTo* est un exemple de lien de cette nature. Une réponse "Oui" à la question "Pouvez-vous être présent dans la réunion du 18/01" n'a plus de sens quand la question n'existe plus (effacée). Sinon, le lien est Séquentiel sans Héritage de l'Effacement. *CommentOn* est un exemple de lien de cette nature. Un

commentaire peut être valable même si la fiche qu'il commente est effacée. Par exemple, une fiche présente une opinion, alors un commentaire peut exprimer une contre-opinion. La contre-opinion a toujours sa valeur, que la fiche originelle existe ou non.

• Prioritaire: un lien prioritaire modifie l'ordre de **lecture** de fiches: la fiche ancienne est lue après la fiche récente. Un exemple est *VersionOf*, il sera très agréable de pouvoir lire directement la version la plus récente.

• Neutre: un seul lien neutre existe. Il ne porte pas de sémantique particulière.

Un ensemble d'opérations est fourni pour manipuler les fiches et la base de fiches:
- Ajout: ajouter une nouvelle fiche dans la base.
- Sélection: sélectionner les fiches satisfaisant une certaine condition.
- Lecture: lire une fiche, la lecture tient compte des liens non-neutres entre fiches.
- Effacement: détruire la fiche sauf son nom et les liens.
- Destruction: détruire la fiche entièrement.
- Ajout de liens.
- Suppression de liens.

Puisque la base de fiches est dupliquée sur tous les BBGA, un mécanisme de distribution et de synchronisation doit être mis en place pour assurer la cohérence entre ces bases. En fait, les utilisateurs ne manipulent directement que la base locale gérée par le BBGA contacté. Ils ne sont pas exposés aux problèmes de distribution et de synchronisation de leurs manipulations. CRMM a adopté un mécanisme simple. Parmi toutes les bases dupliquées, il y en a une qui joue un rôle de **référence**. Certaines manipulations ne peuvent être effectuées dans la base de référence (e.g., ajout de liens, suppression de liens, etc). Les modifications d'une base (référence ou non) sont simplement diffusées à toutes les autres. Le lecteur minutieux peut constater que des états incohérents peuvent exister à certain moment. Par exemple, à cause du délai de transfert de messages, la réponse peut arriver avant la question; ainsi un lien est pointé sur une fiche inexistante. Ce problème est résolu dans CRMM par l'introduction des fiches *fictives* qui sont vides lors de la création et sont remplacées dès que les *vraies* fiches arrivent. Il faut noter que les bases ne sont pas assurées identiques à tout moment, mais à long terme elles sont complètement dupliquées. En fait, dans un environnement distribué, la notion de temps est tout de même différente de site en site. Il est donc tout à fait acceptable et normal d'avoir des états différents pour différents sites à un moment donné.

Le contrôle d'accès de CRMM s'appuye sur les rôles que les utilisateurs jouent dans la conférence. Les rôles d'une conférence CRMM sont

- Lecteur
- Auteur
- Ecrivain
- Editeur
- Administrateur

Les utilisateurs jouant différents rôles ont différents droits d'accès. Par exemple, un lecteur peut lire des fiches, un écrivain peut créer sa fiche et l'ajouter dans la base, l'auteur peut créer une fiche

mais doit la soumettre à un éditeur afin de l'ajouter dans la base, l'administrateur s'occupe de la gestion totale de la base de fiches et des affectations des rôles.

Le BBUA de chaque utilisateur attache une *marque* à chaque fiche lue, les marques sont gérées par le BBUA comme ses données internes.

3.7. Modèle AMIGO

Un service d'archivage de messages a été spécifié par le groupe AMIGO+ ([AMIGO+ 88], [Smith 89]). On en donne ici les idées principales.

Le système de stockage est distribué, il est composé d'un ou plusieurs ASA (Archive Service Agent). Chaque ASA gère un ensemble d'*environnements*. Un environnement est une association ([Benford 88]) logique entre les utilisateurs et un ensemble d'objets d'information.

Un environnement est logiquement dupliqué entre ASA. Les stratégies de duplication et synchronisation entre les différents ASA sont basés sur la technique **maître-esclave**. La modification ne peut s'effectuer que sur une instance maître, elle sera ensuite propagée vers tous les autres ASA, qui contiennent un esclave.

Les objets d'information stockés sont classés en deux catégories (suivant le modèle conceptuel de données d'AMIGO+):

• **Atomique**: ce sont des objets de base ou "Basic Building Blocks". Les objets atomiques peuvent ensuite être utilisés pour former des objets **composites**. Les objets atomiques représentent des objets individuels dans le monde, par exemple, messages, documents simples, etc.

• **Composite**: les objets composites sont définis comme une collection d'autres objets, atomiques ou composites.

Chaque objet (atomique ou composite) contient un ensemble d'attributs formant la partie *descriptive* de l'objet. Outre cette partie, un objet atomique a un *corps* contenant l'information, un objet composite a une partie *composants* contenant l'ensemble des objets composants.

On peut *stocker, lire, rechercher, supprimer, ...* un objet d'information.

Une liste de contrôle d'accès est logiquement spécifiée pour chaque objet, elle indique le droit d'accès de chaque utilisateur.

Notre travail est largement influencé par ce modèle: en fait, l'auteur a participé partiellement à la spécification du service d'archivage AMIGO+.

4. Un Exemple

Nous donnons ici un exemple d'applications et les archiveurs utilisés dans l'exemple. Cet exemple est purement illustratif, nous allons l'utiliser tout au long de cette thèse (en particulier dans le chapitre 4) pour illustrer nos choix et les mécanismes fournis par SAM.

Dans ce but, nous avons choisi un exemple "bien connu", il s'agit d'une **conférence répartie** sur "Messagerie Multi-Média" (MMM). Cette conférence a plusieurs **sous-thèmes**:

- discussion générale
- interface usager
- protocoles de communication multi-média
- traitement de l'image
- traitement de la parole
- traitement du graphique
- archivage de messages multi-média

A chaque instant, il y a un ensemble de messages sur chaque sous-thème.

La conférence est répartie sur un ensemble de **sites** (figure 2-6), classés en trois niveaux:

• Un site au niveau 0, qui se situe à l'université de Carnegie-Mellon aux Etats Unis. C'est le *site de référence* de la conférence.

• *n* sites en *duplication* qui se situent au niveau 1. Chaque site niveau 1 contient une copie complète de la conférence. Les sites se trouvent physiquement à l'*INRIA* (pour la France), au *GMD* (pour l'Allemagne), et à l'université de *Nottingham* (pour l'Angleterre).

• *m* sites au niveau 2. Chaque site niveau 2 stocke un ou plusieurs sous-thèmes et fait référence pour tout le reste au site niveau 0 ou 1 le plus "proche". On appelle ce site niveau 0 ou 1 le site de référence du site niveau 2. Par exemple, le site "EMSE" stocke les sous-thèmes "archivage" et "protocoles", et a pour le site de référence INRIA.

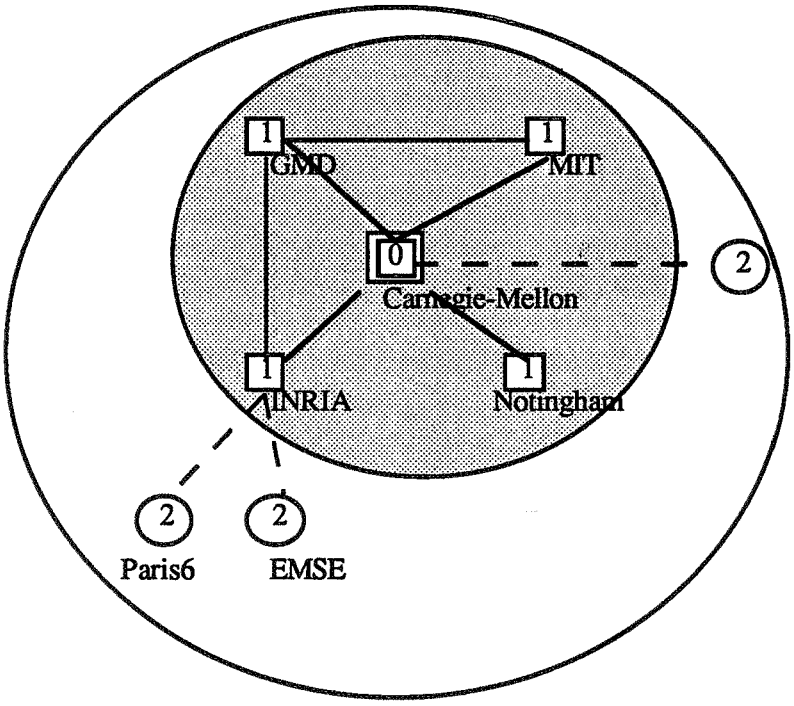


Figure 2-6. Les Sites de la Conférence MMM

La distribution d'informations entre les différents sites de différents niveaux se fait de la manière suivante:

• Les informations (en particulier les nouveaux messages) sont distribués *automatiquement* entre les sites niveau 0 ou 1.
• Les information sont distribuées à un site niveau 2 *à la demande* de ce site, depuis son site de référence niveau 0 ou 1.

Les communications entre les sites sont en mode messagerie : par échanges de messages.

Dans les paragraphes suivants, nous détaillerons les divers aspects intéressants de la conférence. Aucune description formelle ne sera présentée.

4.1. La Base d'Information de la Conférence

La base d'information de la conférence *Messagerie Multi-Média* est l'ensemble des messages de la conférence, organisée en sous-thèmes. Elle est dupliquée totalement ou partiellement dans les sites de la conférence.

Les messages sont liés par des liens orientés et typés. Par exemple, le message B est une réponse au message A, le message C est un commentaire du message A (la figure 2-7). Chaque message est composé d'un ensemble d'attributs et d'un corps. Les attributs sont:

- Identifiant global qui permet d'identifier le message dans n'importe quel site. Cet identifiant est généré par le site auquel le soumet son auteur. Il se compose du nom (ou alias) du site et d'un numéro uniquement dans le site.
- Identifiant local qui est différent d'un site à l'autre, il permet d'accéder rapidement au message.
- Auteur du message.
- Date de péremption.
- Sujet auquel se rapporte le message.
- Mots-clés.
- Liens avec les autres messages. Seuls les liens sortants sont associés au message.

Le corps du message est une séquence de parties de corps. Chaque partie de corps peut être du texte, du graphique, de l'image, du son, etc. Un exemple est donné dans la figure 2-7.

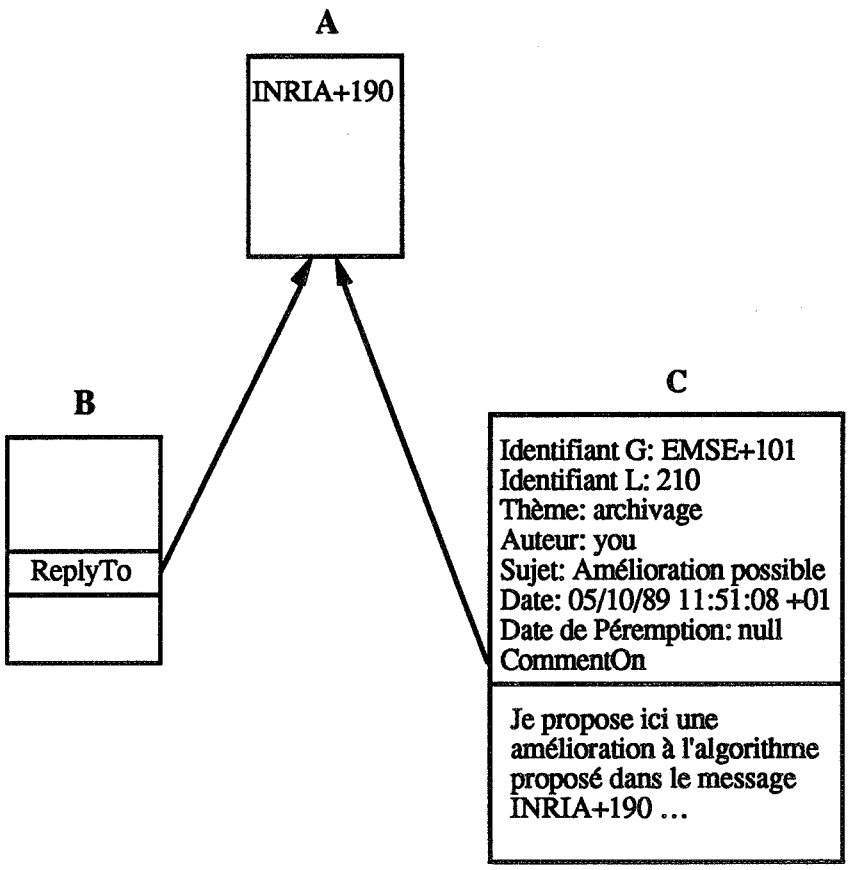


Figure 2-7. Les Messages

4.2. Manipulations de la Base d'Information

Les opérations suivantes sont fournies aux utilisateurs pour manipuler la base d'information de la conférence:

- Soumettre un message à la conférence dans un thème.
- Effacer un message; deux options sont possibles:
  (1) effacer complètement le message,
  (2) effacer le message sans toucher aux identifiants et aux liens.
- Ajouter un lien entre deux messages.
- Supprimer un lien.
- Lire un message.
- Rechercher des messages concernant un thème selon un critère.

Cependant, ces opérations ne sont pas toutes disponibles sur tous les sites. Si les sites niveau 0 ou 1 peuvent exécuter directement toutes les opérations, un site niveau 2 ne peut exécuter directement que les opérations lire et rechercher. Les autres opérations sont "passées" à son site de référence. Cela implique que les sites niveau 2 sont relativement "autonomes" vis-à-vis des autres sites, puisque les manipulations *locales* de la conférence dans un site niveau 2 ne peuvent changer l'état de la conférence vu des autres sites.

4.3. Utilisateurs

Chaque utilisateur de la conférence est "*attaché*" à un site, il est reconnu par le site comme un *utilisateur local*, et joue un ou plusieurs rôles. Les *rôles* de la conférence regroupent les fonctionnalités des utilisateurs, et donc définissent les droits d'accès des utilisateurs. Les rôles de la conférence MMM sont:

- *Administrateur*: chaque site a un administrateur qui gère la base d'information de la conférence du site, les utilisateurs locaux et les relations avec l'extérieur. L'administrateur a tous les droits.
- *Auteur*: la fonctionnalité d'un auteur est de soumettre les messages à la conférence; un auteur a donc le droit de soumettre ou ajouter un message dans la conférence.
- *Lecteur*: un lecteur peut lire ou rechercher des messages dans la conférence.

Les rôles sont attribués aux utilisateurs locaux d'un site par son administrateur. Le rôle de l'administrateur est lui-même affecté à un utilisateur pendant l'initialisation du site.

Par exemple, dans le site "EMSE", l'administrateur est "pays"; "aro", "kaddour", "pays" et "you" sont auteurs, tous les autres membres inscrits sont "lecteurs".

4.4. Aspects Locaux

Certaines libertés sont laissées aux administrateurs des sites pour manipuler et organiser la conférence, sans influence sur les autres sites.

- Pour un site niveau 0, 1 ou 2, l'administrateur peut ajouter des classeurs locaux à l'intérieur d'un thème, lorsque les messages du thème sont trop nombreux.
- Pour un site niveau 2, l'administrateur peut aussi effacer les messages qu'il considère inutiles ou simplement trop "*gros*" de son site. De plus, les sites niveaux 2 peuvent retransmettre automatiquement les nouveaux messages de la conférence dans les BAL de certains abonnés de la conférence.

4.5. Distribution

Les modifications de la conférence (ajout ou effacement d'un message, ajout ou suppression d'un lien) ne peuvent être effectuées directement que dans les sites niveau 0 ou 1. Les sites niveau 2 retransmettent automatiquement ces opérations au site de référence. Une modification est propagée/distribuée automatiquement par le site où s'effectue la modification, à tous les autres sites niveau 0 ou 1, via une liste de distribution.

Un site niveau 2 obtient les modifications de la conférence, faites dans les sites 0 ou 1, à partir de son site de référence. Il demande périodiquement à son site de référence les modifications depuis sa dernière demande. Suite à la demande d'un site niveau 2 attaché, le site de référence niveau 0 ou 1 lui envoie les modifications appropriées.

4.6. Les Archiveurs

Le noyau de chaque site est un archiveur de messages, il effectue les tâches suivantes:

(1) Stocker les messages de la conférence et les organiser selon les thèmes.
(2) Fournir les fonctions pour manipuler les messages.
(3) Classer éventuellement les messages d'un thème localement.
(4) Gérer les accès des utilisateurs locaux à la conférence selon leurs rôles.
(5) Gérer les relations avec les autres sites et la distribution d'informations entre les sites voisins: entre les sites niveau 0 ou 1, et/ou entre les sites niveau 0/1 et ceux niveau 2.
(6) Selon l'environnement où se situe le site, fournir des interfaces appropriées avec les utilisateurs en mode de communication appropriée. Par exemple, le site "EMSE" fournit deux interfaces/protocoles d'accès à l'archiveur: l'une interactive, permet des dialogues interactifs entre l'archiveur et les utilisateurs de la conférence; l'autre en mode messagerie, retransmet automatiquement les nouveaux messages dans les BAL de ses abonnés.

La figure 2-8 montre l'archiveur du site "EMSE".

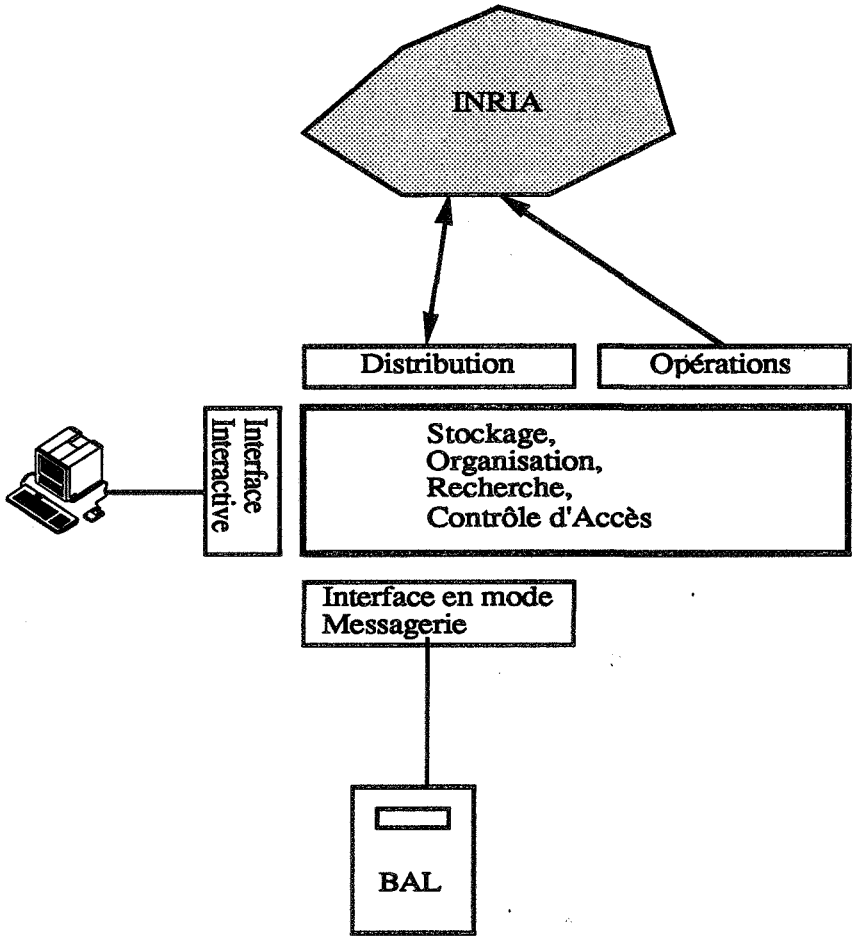


Figure 2-8. L'Archiveur EMSE

Les rectangles représentent différents modules fonctionnels. Ces modules sont généralement différents de ceux des sites niveau 0/1, **sauf le module central** qui gère les messages. L'archiveur du site "INRIA" est donné dans la figure 2-9.

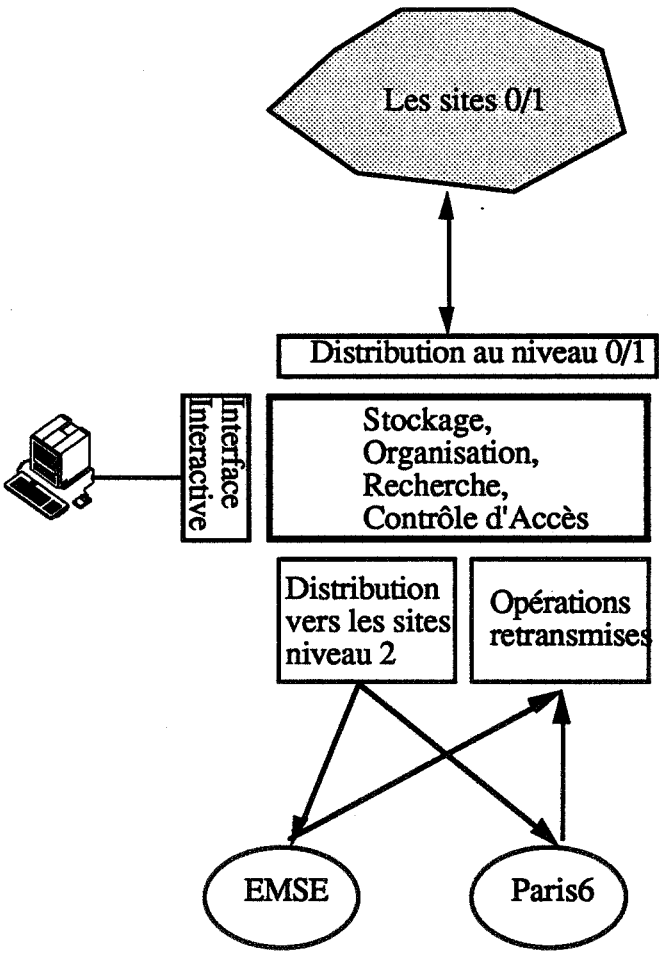


Figure 2-9. L'archiveur INRIA

Nous illustrerons dans le reste de la thèse, comment les mécanismes fournis par SAM peuvent être utilisés pour réaliser les différents archiveurs et comment ils peuvent apporter des extensions intéressantes.

5. Fonctionnalités d'un Archiveur Général de Messages

Dans cette section, nous allons étudier les fonctionnalités que l'archiveur de messages général SAM doit avoir, afin de supporter les différentes applications MHS et GC (cf. [You 89a]).

Deux aspects différents des messages doivent être traités par SAM pour la gestion des messages: un message est une unité d'échange et une unité d'information. Pour traiter les messages comme des unités d'échange, SAM doit communiquer avec le système de transfert de messages et gérer les relations avec l'expéditeur, les destinataires, etc. Pour traiter les messages comme des unités d'*information*, SAM doit gérer les informations contenues dans les messages et les relations entre ces informations. Par exemple, un document complexe peut être découpé et envoyé dans plusieurs messages, il faut donc relier ces messages et garantir l'intégralité du document à la réception.

SAM doit assurer les stratégies d'utilisation *centrale* (par exemple, celle de COM) et *distribuée* (e.g., celle de CRMM). La stratégie centrale permet à un SAM d'être utilisé comme un

ensemble d'archiveurs individuels et de supporter un ensemble d'applications (Boîtes aux lettres, Conférences, ...).

SAM doit pouvoir être utilisé comme un archiveur *mono-utilisateur* et *multi-utilisateurs*. Le dernier cas implique la nécessité de contrôles d'accès. Par exemple, nous devons savoir, dans un archiveur multi-utilisateurs, quels sont les utilisateurs qui ont le droit d'ajouter un message, de lire un message, de supprimer un message, etc. En général, le contrôle d'accès doit être fait en fonction de l'utilisateur qui effectue l'accès, de l'opération qu'il effectue et du message qu'il veut manipuler par cette opération.

Les différents types de protocoles sont demandés par différentes applications. Comme on l'a montré, ces différents protocoles utilisent des modes de communication différents, par exemple, P7 en mode interactif ou P2+ en mode messagerie. Il est aussi possible que la même application ait besoin de plusieurs protocoles pour accéder à l'archiveur. Le SAM doit être assez souple pour supporter tous ces protocoles et les différents modes de communication.

L'archiveur de messages gère une base de messages. Il doit fournir les fonctions de base de stockage, par exemple, ajouter un message, supprimer un message, lire/extraire un message ou une partie d'un message. Des exécutions automatiques de ces fonctions peuvent parfois être demandées, par exemple, supprimer un message après sa date de péremption.

Le SAM doit fournir des facilités pour structurer et grouper les objets/messages, par exemple, les *folders* ou *cabinets* de fichiers dans une boîte aux lettres personnelle, ou *thèmes* d'une conférence. Ces facilités peuvent être utilisées comme outils de partage entre plusieurs applications, par exemple, les activités dans COM.

SAM doit posséder un mécanisme de *recherche* puissante en plus de la simple *liste*. La recherche doit être capable de comparer des attributs de messages, des relations entre messages et/ou utilisateurs, et d'utiliser les résultats de groupement.

Contrairement à une base de données relationnelles, la recherche *textuelle* peut être importante. Par exemple, le texte à comparer peut être le sujet, les mots-clé, ou un mot dans le corps du message. Les autres extensions futures concernent le *graphique*, l'*image*, le *son*, etc.

SAM doit être un *véritable* archiveur de messages, mais pas une base de données générale. Il doit comprendre certaines sémantiques des messages, leurs structures, et fournir des supports appropriés. Par exemple, SAM doit pouvoir supporter les liens comme *InReplyTo*, *VersionOf* entre les messages.

Bien que les propriétés originelles d'un message ne puissent pas être changées une fois que le message est envoyé et stocké, il est souvent très agréable de pouvoir attacher des attributs additionnels à un message, ou établir des liens entre des messages dans le SAM, en particulier dans un environnement bureautique. Par exemple, on pourra ajouter un attribut à chaque message pour indiquer s'il est *intéressant*; cet attribut est affecté par l'utilisateur dynamiquement quand il le lit, et ensuite utilisé pour trouver tous les messages intéressants. Les autres exemples d'attributs pour une boîte aux lettres personnelle sont *mots-clé*, *référence-croisée*, etc.

En plus des fonctionnalités de base, SAM doit fournir les fonctionnalités d'administration et de gestion, par exemple, effacer une partie périmée d'un message. Il doit fournir aussi des facilités pour gérer la nouveauté dans l'archiveur pour les utilisateurs, ainsi que les facilités pour gérer les versions.

Enfin, comme toutes les bases de données, SAM doit garder les informations **logs** ou d'historique. Ces informations sont souvent utiles pour la sauvegarde et la récupération après un incident. Les autres aspects comme l'*accès concurrent*, ... doivent aussi être gérés par SAM.

6. Projet SAM

Nous pouvons constater que des archiveurs sont indispensables et jouent des rôles importants pour les applications MHS et GC, et que leurs constructions sont souvent difficiles.

A cause des caractéristiques spécifiques des archiveurs de messages, les bases de données conventionnelles ne sont pas adaptées, il n'existe aucun outil pour construire des archiveurs de messages dans un environnement distribué. De plus, les archiveurs de messages existants sont souvent incomplets au niveau fonctionnalité, et sont spécifiques aux applications. Ils ne sont pas extensibles et donc ne peuvent être utilisés pour construire de nouveaux archiveurs d'applications différentes. Cela rend la conception et la réalisation de nouvelles applications MHS et GC difficiles.

Le projet SAM consiste à construire un archiveur de messages général (SAM) comme un outil de construction des archiveurs de messages pour les applications MHS et GC; SAM doit être adapté à chaque utilisation d'une application spécifique. Chaque instance de SAM est un archiveur de messages spécifique à une application. Dans ce but, les travaux suivants sont effectués dans le cadre du projet:

- Spécifier les besoins d'archiveurs de messages en étudiant des exemples d'archiveurs de messages.

- Proposer un modèle architectural de SAM: quel est le composant commun de toutes les instances, quel est le composant spécifique aux archiveurs d'applications?

- Définir le noyau universel, son modèle conceptuel, son modèle fonctionnel et son modèle interne. Réaliser un prototype du composant commun.

- Valider et montrer la puissance de SAM en spécifiant/réalisant des instances pour des applications spécifiques, par exemple, l'archiveur de messages personnel bureautique, du système de conférence répartie, du système de votes, etc.

Deuxième Partie

Solution SAM

Chapitre 3

Modèle Architectural

Chapitre 4

Modèle Conceptuel d'Applications de SAM

Chapitre 5

Conception de MAP

Chapitre 6

Réalisation d'un Prototype

Chapitre 3
Modèle Architectural de SAM

1. Introduction

SAM est un archiveur général de messages pour supporter des applications MHS et GC. Il est général puisque les archiveurs de la plupart des applications MHS et GC peuvent être réalisés en adaptant SAM, i.e. chaque archiveur spécifique peut être construit à partir de SAM. En fait:

- Des applications différentes ont besoin de modes de communication différents: interactifs ou sans connexion, e.g., *store and forward.*

- Une application accède à ses archiveurs suivant un protocole spécifique à l'application. En plus, la façon de communiquer avec le système de messagerie est différente selon le système de messagerie utilisé et son implantation.

- Les stratégies de distribution et de répartition d'informations sont spécifiques à l'application, ainsi que les relations entre ces archiveurs.

- Les fonctionnalités comme le contrôle des nouveautés, la gestion de versions, et les traitements automatiques sont très spécifiques aux applications. Les différentes stratégies sont souvent adaptées aux différentes applications selon leurs archiveurs spécifiques. Par exemple, le contrôle des nouveautés peut être effectué dans l'archiveur (comme COM) ou par des agents utilisateurs (comme CRMM). De plus, la notion de *nouveauté* peut être différente, par exemple, un message est nouveau "s'il est entré dans l'archiveur après la dernière consultation", ou "s'il n'est pas encore lu", X.413 a même défini plusieurs états pour ce fait.

  Il est donc très difficile, sinon impossible d'avoir une stratégie universelle ou de fournir toutes les stratégies possibles. De plus, même si une stratégie universelle était possible, la réalisation d'un tel archiveur serait trop complexe.

- De plus, les applications ont souvent des fonctionnalités qui leur sont spécifiques, par exemple, l'auto-retransmission ou l'auto-réponse pour un archiveur personnel. Il est impossible d'énumérer ici toutes les fonctions possibles que peuvent supporter les différentes applications.

Il est donc nécessaire de commencer par définir un modèle architectural qui indique quels sont les composants de SAM, les fonctions de chacun de ces composants ainsi que leurs relations, les composants communs et les composants spécifiques à chaque application.

Ce modèle architectural doit traiter les fonctions essentielles de l'archiveur de messages et les fonctions spécifiques à l'application. Il doit aussi permettre aux concepteurs d'applications d'ajouter ou de modifier facilement les fonctionnalités spécifiques de leurs archiveurs (cf. [You 89a]).

2. Modèle

Dans ce but, on a conçu un modèle à deux niveaux, tel que montré dans la figure 3-1.

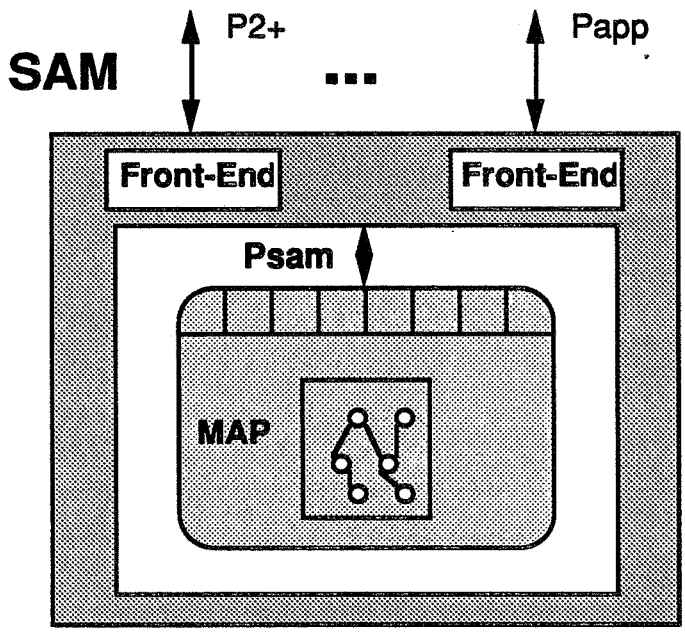


Figure 3-1. Architecture de SAM

Chaque instance de SAM se compose d'un noyau **MAP** (pour *Message Archive Processor*) commun à toutes les instances et un ou plusieurs **FE** (pour *Front_Ends*).

Le noyau fournit les outils de base pour stocker et gérer les objets d'information d'une application MHS et GC, par exemple, *messages*, *conversations*, *dossiers*, etc. Le noyau gère les relations entre les objets d'information (e.g., InReplyTo, CommentOn, etc), et les relations entre les objets d'information et les utilisateurs de l'application (e.g., expéditeur, destinataires, auteur, etc). Il s'occupe aussi du contrôle de bas niveau. De plus, MAP fournit des supports de base pour la réalisation des fonctionnalités avancées ou spécifiques, comme la répartition, la distribution et la synchronisation d'informations, le contrôle de nouveauté, la gestion de versions, etc. MAP n'a pas de connaissance spécifique des objets vis-à-vis d'une application particulière.

MAP est parfaitement défini, il communique avec les Front_End via le protocole d'accès unique **Psam**. Ce protocole définit parfaitement l'interface de MAP avec ses clients.

Chaque archiveur d'une application est une instance de SAM qui possède un ou plusieurs FE spécifiques à l'application. Ces FE adaptent SAM à l'application spécifique. Le rôle d'un FE est détaillé comme suite:

- Il fournit la sémantique spécifique des objets dont a besoin l'application. Un exemple d'objets est celui de *conversation* avec les opérations associées, un autre exemple est le concept de *folder* dans un archiveur personnel. Cependant ces deux sortes d'objets sont traités dans MAP de manière identique comme une *collection* d'objets.

• Il gère les relations avec les autres archiveurs, càd distribuer et synchroniser des informations entre ces archiveurs.

• Il fournit les fonctionnalités avancées et spécifiques au besoin de l'application, mais aussi des facilités de gestion adaptées à l'application, tout en se basant sur celles fournies par MAP. Par exemple, le contrôle de nouveauté, le schéma d'*accounting* dans une conférence, ou le contrôle d'accès particulier dans un environnement bureautique sont généralement réalisés par des FE en utilisant les facilités fournies par MAP.

• Il fournit aux utilisateurs de SAM (humains ou programmes) le protocole et le mode de communication appropriés. Plusieurs FE peuvent coexister et fournir des protocoles et des modes de communication différents. Par exemple, un archiveur de messages peut fournir un protocole interactif pour des utilisateurs ayant des connexions directes, et en même temps un protocole en mode messagerie pour ceux qui communiquent via le système de messagerie, au travers des FE différents.

MAP est ainsi conçu de telle manière que les FE sont généralement faciles à construire. De plus, il est possible d'avoir une boîte à outils de FE. Il faut noter qu'il est possible que la fonction des FE soit intégrée dans l'application elle-même; et dans ce cas, MAP devient lui-même un serveur d'archivage de messages général de "*bas niveau*".

Pour donner un exemple, prenons les archiveurs des sites de la conférence MMM (cf. chapitre 2 §4). L'archiveur du site "EMSE" (figure 2-8) peut être réalisé par SAM de la façon suivante:

• Le module central est MAP. Il stocke, organise et gère les messages de la conférence.
• Quatre FE réalisent respectivement les quatre modules d'interfaçage:
  • le premier fournit le protocole d'accès interactif, qui permet d'utiliser l'archiveur interactivement par les utilisateurs.
  • le deuxième réalise un protocole d'accès en mode messagerie. Il dépose directement les nouveaux messages dans les boîtes aux lettres de certains abonnés.
  • le troisième gère la distribution avec le site niveau 1, i.e.le site INRIA. Il s'agit simplement de demander des modifications au site INRIA et de mettre à jour la base de messages en exécutant des modifications reçues.
  • le dernier gère la retransmission automatique des opérations de modification. En fait, pour chaque opération de modification, il suffit de vérifier simplement le droit d'accès de l'utilisateur qui demande l'opération, et ensuite de l'envoyer au site INRIA.

Il est également très simple de réaliser l'archiveur du site INRIA (figure 2-9) en utilisant SAM, ceci consiste à construire quatre front-end spécifiques:

• le premier fournit un protocole interactif comme celui de l'archiveur EMSE.
• le deuxième gère la distribution au niveau 0-1. Ceci consiste à envoyer simplement chaque modification effectué à la liste de distribution, et reçoit les modifications des autres sites 0-1.

• le troisième gère la distribution aux sites niveau 2. A la demande d'un site niveau 2 attaché, il envoie les modifications effectuées sur les thèmes abonnés par le site niveau 2, depuis la dernière demande.
• le quatrième traite les retransmissions automatiques des opérations venant des sites niveau 2; il les exécute comme si c'étaient les opérations demandées par des utilisateurs locaux.

3. Conclusion

Le modèle architectural que l'on vient de présenter est particulièrement flexible. Il est possible d'ajouter, de modifier ou de retirer un FE d'un SAM existant pour l'enrichir ou mieux l'adapter à l'application. L'inverse est aussi valable, i.e. il est possible de remplacer MAP, de façon transparente, par une nouvelle version/implantation de MAP mieux adaptée ou plus efficace, grâce à l'unique protocole Psam.

D'un point de vue génie logiciel, ce modèle permet de réutiliser un composant très valable et très complexe: MAP; il permet également de construire une boîte à outils de FE.

Chapitre 4
Modèle Conceptuel d'Applications de SAM

1. Généralités

Pour supporter des applications MHS et GC, il est nécessaire de savoir quels sont les objets que les applications MHS et GC manipulent, leurs relations, et comment ces objets sont représentés et manipulés. Dans ce but, nous allons donner dans ce chapitre le *Modèle Conceptuel d'Applications* du point de vue de l'archiveur SAM. Il décrit de façon abstraite les données manipulées par les applications MHS et GC. Ce modèle est basé sur le modèle général d'information d'applications MHS et GC, proposé par le groupe AMIGO+ ([Benford 88], [Smith 89]).

Le modèle conceptuel indique non seulement à MAP les objets à gérer, mais il fournit aussi aux applications un outil de modélisation et d'utilisation de MAP. Le rôle du modèle conceptuel est donnée dans la figure 4-1.

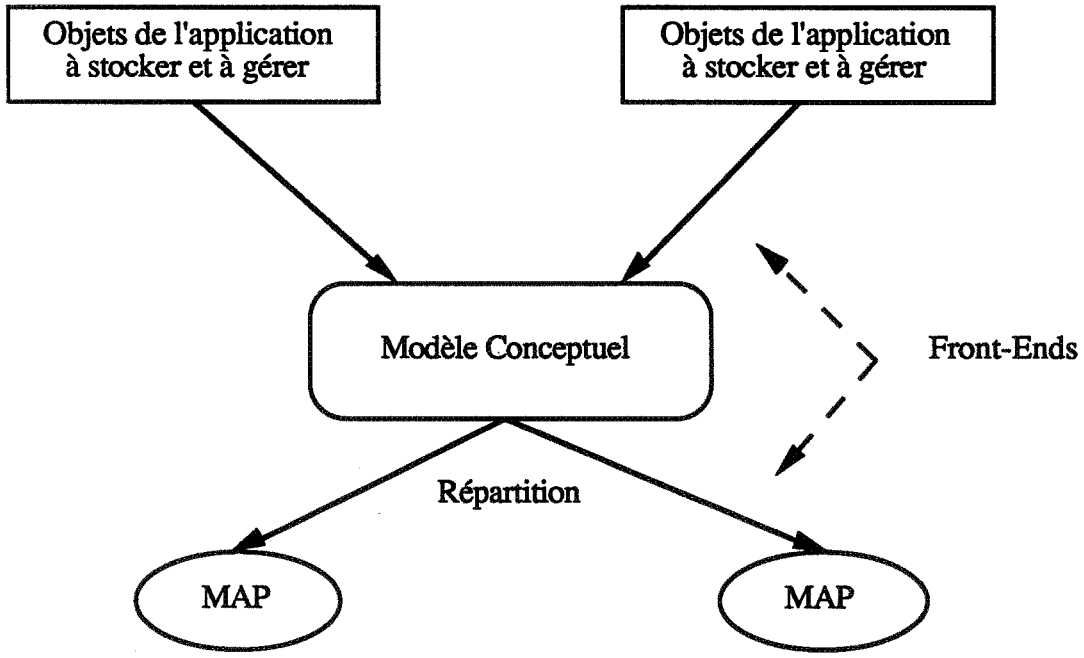


Figure 4-1. Le rôle du Modèle Conceptuel

Sur la figure 4-1, nous pouvons constater que le modèle conceptuel supporte plusieurs applications, et qu'une application peut répartir ses données (dupliquées ou distribuées) sur plusieurs bases MAP.

2. Composants du Modèle

Une application de communication de groupe est un processus de communication; il consiste à manipuler des **informations** par **un groupe d'entités de communication.**

Une entité de communication est une entité qui participe au processus de communication. Elle peut être un humain, un groupe, une liste de distribution, un programme d'ordinateur, un agent de l'application, etc; c'est donc une entité qui est capable de communiquer.

Les informations manipulées sont représentées par les **objets d'information**. Un objet d'information est une unité d'information qui est échangée, partagée ou manipulée durant le processus de communication. Ainsi les documents, les messages ou les conversations sont des objets d'information.

Les objets d'information manipulés pendant le processus de communication appartiennent à **un espace d'information**. Un espace d'information pourra être partagé par plusieurs processus de communication, il fournit donc un moyen de partager et d'échanger des informations entre plusieurs applications.

Les interactions entre les entités de communication et les objets d'information durant le processus de communication ont lieu dans un **environnement**. Un environnement est une association entre un groupe d'entités de communication et un ensemble d'objets d'information. Il peut aussi spécifier les stratégies d'accès, les protocoles de communication appropriés, la stratégie de contrôle d'accès, le(s) nom(s) des archiveur(s) où les objets d'information sont stockés, etc. C'est donc un espace de travail et de gestion pour les entités de communication. De ce point de vue, un environnement peut avoir des sous-environnements qui définissent des sous-espaces pour le même groupe ou un sous-groupe. Une application MHS et GC possède au moins un environnement.

La figure 4-2 montre les relations entre un environnement, les objets d'information et les entités de communication. Les entités de communication (utilisateurs, petits bons hommes dans la figure) accèdent et manipulent les objets d'information (petits rectangles) à travers l'environnement (le grand carré). Ce dernier contrôle les accès des utilisateurs aux objets d'information. De plus, les entités de communication peuvent aussi avoir des relations sémantiques avec les objets d'information (e.g., un message est destiné à un utilisateur).

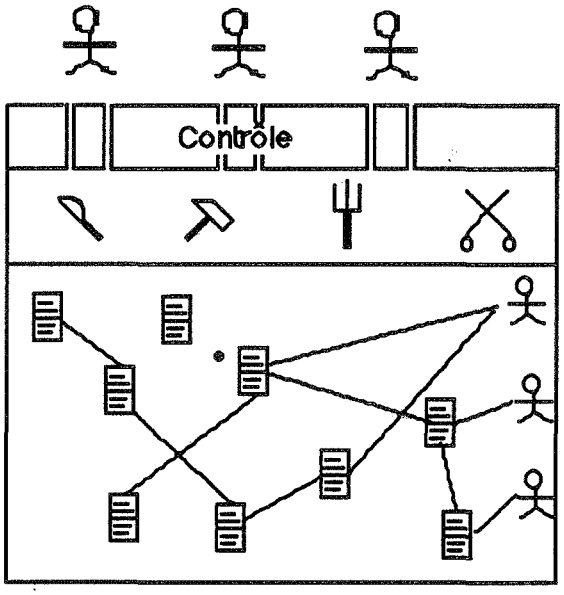


Figure 4-2. Les composants

Le rôle du système de stockage/archivage est de gérer l'espace d'information, de stocker et de fournir des primitives pour manipuler les objets d'information de l'espace. Il doit aussi faire

face aux entités de communication (i.e. les utilisateurs du service de stockage fourni) et les paramètres des environnements qui spécifient le contrôle d'accès.

3. Identifications

3.1. Identifications des Entités de Communication

Le Directory X.500 fournit une manière globale et homogène de nommer, d'enregistrer et de gérer les entités de communication; il affecte à chaque entité de communication un nom unique DN (pour Distinguished Name) dans le Directory. Les attributs de l'entité de communication sont aussi enregistrés dans le Directory avec l'entité.

Cependant, en l'absence du Directory X.500, de nombreux autres schémas de nommage sont utilisés, e.g., O/RName dans la messagerie X.400, ou Internet *User Naming Schéma.* Ces noms peuvent être gérés par un système de directory global ou local. Les applications s'appuyant sur ces schémas de nommage doivent aussi être supportées par l'archiveur.

Donc, nous supposerons qu'un service de directory existe (global ou local à l'application, géré explicitement par un système de directory ou implicitement par les FE de l'archiveur) dans l'environnement où l'archiveur est utilisé, et que ce système de directory permet de nommer une entité de communication de façon globalement unique et de l'enregistrer avec ses attributs.

3.2. Identification des Objets d'Information

Chaque objet d'information doit avoir un nom globalement unique pour qu'il puisse être identifié et manipulé dans un environnement distribué. En théorie, il est possible qu'il soit enregistré dans le DS et ait un DN. Cependant, cela voudrait dire qu'on doit enregistrer tous les objets d'information dans le DS, ce qui est pratiquement impossible. Il est raisonnable de n'enregistrer dans le DS que les objets d'information *importants* et à longue durée de vie.

De plus, un objet d'information peut exister dans plusieurs contextes et être utilisé dans plusieurs applications, il peut donc avoir plusieurs identifications. Par exemple, un message qui contient un document ISO a un IPM-ID dans le cadre du X.400 P2 en plus de son numéro de document ISO, il peut aussi avoir un DN s'il est enregistré dans le Directory comme un message *important.*

Bien qu'il n'y ait pas de schéma universel pour le nommage d'objets d'information, une règle existe: l'identification d'un objet d'information se compose de deux parties, l'*Identification de Contexte* dans lequel l'objet est nommé et un *Identificateur Spécifique au Contexte* qui est unique dans le contexte. L'identification ainsi formée est unique.

3.3. Identification des Environnements

Le principe est le même que celui des entités de communication. Un environnement est supposé être enregistré dans un Directory. Donc, il a un nom de DS où il est enregistré avec les attributs associés.

4. Les Objets d'Information

Dans cette section, nous allons d'abord étudier des exemples d'objets d'information des applications MHS et GC, nous en déduirons ensuite les objets d'information à supporter par MAP de façon plus abstraite.

4.1. Analyse d'Objets d'Applications

A chaque instant, un objet d'information existant est soit en cours de transmission, soit stocké dans un ou plusieurs archiveurs.

4.1.1. Messages et Messages Stockés

Les messages sont les unités d'information de base dans un environnement MHS. Généralement, un message se compose d'*un ensemble d'attributs* et d'un *corps*. Si les attributs sont généralement structurés et contiennent des informations *courtes* (les tailles sont prévisibles), le corps peut contenir de grandes quantités d'informations non-structurées comme par exemple, des textes, ou des images. Cependant, il est possible que l'information que le message représente soit complètement structurée et que le corps soit vide. La plupart des attributs définis par le système de messagerie ne sont pas obligatoirement présents dans chaque message, l'expéditeur a le droit de choisir de remplir un sous-ensemble d'attributs. Les messages sont souvent liés entre eux, ils peuvent même s'emboîter, i.e. le corps d'un message contient un ou plusieurs autres messages. Un message possède souvent un identificateur unique composé du nom de son environnement de création et d'un identificateur unique dans l'environnement.

Par exemple, un message du MTS X.400 (cf. figure 1-2) se compose d'une enveloppe et d'un contenu; l'enveloppe est un ensemble d'attributs prédéfinis par le standard, le contenu est une chaîne d'octets non-structurée du point de vue du MTS. Un message IP X.400 (cf. figure 1-3) est composé d'un en-tête et d'un corps. L'en-tête est un ensemble d'attributs prédéfinis, le corps est une séquence de parties de corps qui peuvent être du texte, de l'image, du son, ou d'autres messages IP *forwarded*. RFC-822 est un autre standard couramment utilisé, les messages RFC-822 sont aussi composés d'un ensemble d'attributs et d'un texte dont la taille n'est pas fixée, la figure 4-3 en donne un exemple.

```
From you@asi11.emse.fr Thu Aug  3 17:39:05 1989
Return-Path: <you@asi11.emse.fr>
Received: from asi11.emse.fr by moon.emse.fr (3.2/SMI-3.2)
       id AA03658; Thu, 3 Aug 89 17:39:08 BST
Date: 03 Aug 89 17:39:05
From: Y-Z YOU <you@asi11.emse.fr>
To: <you@moon.emse.fr>
Subject: Format RFC-822
Message-Id: <618162701.135.0@asi11.emse.fr>
Le format RFC-822 contient un ensemble d'attributs et un segment de
texte. Les attributs suivent la convention "Mot-Cl'e Valeur" cod'es
en ASCII texte.
       Ce message lui-meme est un exemple.
```

Figure 4-3. Un message RFC-822

Outre les informations initialisées par l'expéditeur, un message stocké contient souvent des informations additionnelles, par exemple:

- les attributs associés au message pour les besoins du stockage, e.g., l'identificateur interne, la date de la dernière consultation, la date d'entrée dans l'archiveur, etc.

- les attributs imposés par l'environnement ou par l'application; les attributs personnels ajoutés par l'utilisateur qui peuvent ne pas être présents dans le message originel, e.g., la date de péremption, les mots-clé, le niveau d'intérêt, etc.

- les liaisons additionnelles avec les autres objets, e.g., l'utilisateur peut relier deux messages par *RelatedTo*, dynamiquement dans un archiveur. Il est possible que l'expéditeur d'un message ne connaisse pas l'autre message.

Ces informations sont souvent structurées et sont représentées comme attributs supplémentaires ou *additionnels* dans l'archiveur.

Les attributs additionnels ne sont pas obligatoirement statiques, contrairement aux attributs originels d'un message, puisqu'on a souvent besoin de les modifier (ajouter, changer la valeur, supprimer, etc) dynamiquement.

4.1.2. Objets de Bureau

Il est souvent pratique de stocker de *petits* objets d'information associés aux messages d'un archiveur de messages dans un environnement bureau. Par exemple, on attache souvent un *post-it-note* à un message indiquant certaines des propriétés du message et/ou des commentaires; un autre exemple est les *mémos*. Permettre de stocker ces objets de bureau dans un archiveur de messages garde les liens sémantiques entre les messages et les objets de bureau, et fournit un environnement de travail homogène aux utilisateurs.

Comme les messages, un objet de bureau est composé d'un ensemble d'attributs et d'un corps contenant les informations principales. Cependant, les objets de bureau sont souvent dynamiques.

4.1.3. Conversations

Une conversation est une forme de base pour la communication humaine. Dans un environnement messagerie, une conversation est composée d'un ensemble de messages relié par des liens sémantiques, en particulier, *InReplyTo* et *CommentOn*. Les messages d'une conversation sont souvent reliés directement ou indirectement à un message originel. Par exemple, le message originel pose une question ou exprime une opinion discutable; les autres messages répondent ou commentent directement ce message ou indirectement d'autres messages de la conversation.

Une conversation a les caractéristiques suivantes:

- Elle a souvent un identificateur globalement unique, qui peut être utilisé pour l'enregistrer dans le système de directory. L'identificateur ne change pas quand l'ensemble des messages de la conversation change (arrivée d'un nouveau message, suppression d'un message existant, etc).

• Elle contient un ensemble de messages, qui change dynamiquement. Il est souvent possible de décrire cet ensemble par une règle, on l'appelle la *règle de dérivation*. Par exemple, "tous les messages répondent ou commentent le message EMSE-103".

• Elle contient aussi un ensemble d'attributs représentant les propriétés de la conversation, par exemple, la description de la conversation, le coordinateur de la conversation, etc.

On doit pouvoir

• Créer une conversation en spécifiant la règle de dérivation. Si la conversation est maintenue par un utilisateur, la règle de dérivation peut être absente.
• Installer une conversation à partir de messages existants, un sous-ensemble des messages étant inclus dans la conversation.
• Détruire une conversation. Les messages inclus peuvent aussi être détruits, selon que les messages sont inclus ou non dans les autres conversations.
• Fermer une conversation. Une conversation fermée ne peut plus avoir de nouveau message.
• Extraire tous les messages d'une conversation.
• Lire une conversation. L'ordre de lecture est déterminé par les liens sémantiques entre les messages de la conversation. Par exemple, une question doit être lue avant les réponses.
• Ajouter un message dans une conversation. Quand la règle de dérivation est spécifiée, ceci est fait automatiquement.
• Supprimer un message d'une conversation. Dans une conversation avec une règle explicite, cela peut être récursif, e.g., la suppression d'un message supprime aussi tous les commentaires du message détruit.

4.1.4. Classeurs

Les applications ont souvent besoin de moyens de classification et d'organisation de messages dans un archiveur, par exemple, les *cabinets*, *folders*. On appelle ces objets *classeurs*. Un classeur regroupe un ensemble de messages de façon hiérarchique, i.e. un classeur peut avoir des sous-classeurs en plus des messages contenus. L'ensemble de messages (directs ou indirects) d'un classeur peut être décidé par une règle de dérivation, ou classifiés explicitement par des utilisateurs.

Un classeur a aussi des attributs associés représentant les propriétés du classeur, e.g., l'identificateur du classeur, la description du classeur, les relations avec d'autres classeurs, etc.

Les opérations, *créer, détruire, ajouter, supprimer, modifier, ...* sont souvent fournies pour manipuler les classeurs.

4.1.5. Collections

Certains archiveurs d'applications fournissent la notion plus générale de *collection*. Une collection contient un ensemble d'objets d'information, et un ensemble d'attributs. Les objets d'information peuvent être eux-mêmes des collections; les collections forment ainsi une hiérarchie.

Comme les conversations ou les classeurs, chaque collection a un identificateur unique et/ou une règle de dérivation. On pourra aussi attacher un attribut de contrôle d'accès à la collection et/ou aux objets contenus dans la collection.

4.1.6. Documents Complexes

Dans un environnement messagerie, un document complexe de grande taille (un livre, un chapitre de livre, un rapport de recherche, etc) est souvent formé de plusieurs parties (résumé, chapitre, section, ou paragraphe) chacune représentée par un message ou un autre document complexe. Les relations entre ces parties sont représentées par les liens sémantiques entre des messages. Par exemple, si on décompose un document en un résumé et plusieurs chapitres, dans l'environnement messagerie le document est composé d'un message contenant le résumé et d'un ensemble des messages, chacun représentant un chapitre lié par le lien ContinuationDe au chapitre précédent. Cela permet d'envoyer d'abord le résumé, et s'il est intéressant pour un destinataire, on lui enverra ensuite les différents chapitres. Un document complet est donc une hiérarchie de messages et de documents complexes. Chaque document (top-niveau ou non) peut avoir des attributs représentant le profil du document (mots-clé, référence, auteur, version, établissement de publication, etc).

Un document complexe contient donc un ensemble d'objets d'information qui sont des messages ou des documents, et un ensemble d'attributs. Un aspect spécial lié aux documents complexes est la gestion des versions, une partie (ou le document lui-même) peut avoir des versions différentes. La gestion des versions est un travail difficile quand chaque partie peut avoir ses propres versions dans un environnement distribué.

On peut ajouter ou supprimer une partie, ajouter une nouvelle version d'une partie, ou lire le document.

En résumé, nous donnons ici les principales caractéristiques des objets d'information présentés dans cette section:

Messages stockés: Un message est un véhicule de l'information, il est généralement composé d'un *en-tête* et d'un *corps*. L'en-tête est un ensemble d'attributs; ils peuvent être divisés en deux parties: les attributs *originels* qui sont générés par l'expéditeur ou l'agent utilisateur de l'expéditeur; et les attributs *additionnels* qui sont dynamiquement attachés au message par l'utilisateur ou l'archiveur. Les attributs originels sont statiques, contrairement aux attributs additionnels qui sont dynamiques. Le corps d'un message se compose d'une séquence de parties de corps, chacune peut être du texte, de l'image, du graphisme, du son, ..., ou un autre message. Le corps d'un message est statique. Un message a souvent un identificateur généré par l'agent utilisateur de l'expéditeur. L'utilisateur peut modifier la partie dynamique de l'en-tête.

Objets de bureau: Les objets de bureau contiennent des informations dynamiques dans un environnement bureautique. Ils sont souvent associés aux messages, et se composent eux aussi d'un ensemble d'attributs et d'un corps. Cependant contrairement aux messages, les objets de bureau sont dynamiques; l'utilisateur peut modifier les attributs (ajouter, supprimer, modifier) et le corps.

Documents Complexes: Dans un environnement messagerie, un document complexe structuré est souvent véhiculé par un ensemble de messages. Chaque message de l'ensemble représente une partie du document, les liens entre les messages représentent les relations entre les différentes parties du document. En particulier, la relation *ContinuationDe* permet de décomposer linéairement un document en une séquence de sous-parties. Les messages avec la relation *partie-de* décomposent un document hiérarchiquement, chaque message du document se compose:

- d'un ensemble d'attributs
- d'une partie du document (e.g., un chapitre d'un rapport), ou des références aux messages qui constituent cette partie (e.g., section).

Un utilisateur peut manipuler les attributs et les références.

Conversations: Une conversation se compose d'un ensemble d'attributs et d'une liste de références aux messages constituant la conversation. Deux méthodes de la constitution d'une conversation existent:

- les références sont obtenus *automatiquement* par l'archiveur selon une *règle de dérivation.*
- les références sont ajoutées *manuellement* par les utilisateurs.

Les utilisateurs peuvent donc manipuler les attributs d'une conversation ou les références aux messages lorsque la conversation est manuelle.

Classeurs/Collections (folders): Un classeur contient une liste de références aux autres objets (messages, conversations, classeurs, etc). Les classeurs fournissent donc un moyen d'organisation et de classification hiérarchique. On manipule (ajouter ou supprimer) les références pour organiser des objets. Un classeur peut aussi être *automatique* et effectuer la classification automatique: Il classifie automatiquement les messages lorsque ces derniers sont stockés dans l'archiveur (par l'agent de remise, par exemple), selon une *règle de filtrage*. Les attributs et la liste de références d'un classeur est dynamique, l'utilisateur peut les manipuler dynamiquement.

4.2. Objets Atomiques et Objets Composites

On classifie les objets en deux catégories, *atomiques* et *composites*.

4.2.1. Objets Atomiques

Un objet atomique représente un seul objet réel, par exemple, un document simple, un message non-emboîté, un post-it-note, etc. Les objets atomiques sont les unités de base de construction, ils sont utilisés pour créer des objets composites. Chaque objet atomique a un nom global qui permet de l'identifier pour effectuer une opération.

Un objet atomique est *semi-structuré*, il se compose d'un ensemble d'attributs structurés et d'un corps contenant des informations non-structurées. Les attributs et le corps peuvent être dynamiques. On peut donc créer, lire, modifier ou détruire un objet atomique.

4.2.2. Objets Composites

Un objet composite représente un objet réel qui contient d'autres objets, atomiques ou composites, par exemple, des documents complexes, des conversations, des collections, ou des messages avec d'autres messages emboîtés. Un objet composite est aussi *semi-structuré*, il est composé d'un ensemble d'attributs et d'un corps contenant des références aux autres objets et/ou des informations non-structurées de taille importante.

Comme un objet atomique, un objet composite a un nom global. Un objet composite peut avoir une règle de dérivation qui génère automatiquement le corps de l'objet, au besoin. Cette règle est une description des attributs des objets composants, elle peut spécifier aussi les relations entre ces objets composants. Un exemple simple est: `tous les messages ayant une haute importance` **ou** `tous les messages répondant ou commentant directement` **ou** `indirectement un message originel.`

En plus des opérations permises pour les objets atomiques, on peut énumérer les composants d'un objet composite, ajouter ou retirer un composant d'un objet composite. La destruction d'un objet composite peut entraîner la destruction de tous les composants, ou simplement supprimer les liaisons entre l'objet et ses composants. Il est aussi possible de rechercher des objets parmi les composants d'un objet composite, par exemple, chercher des messages dans un classeur.

4.3. Structure d'Objets d'Information

Les objets d'information sont semi-structurés, ils se composent d'un ensemble d'attributs et d'un corps qui peut contenir des informations non-structurées. Le corps d'un objet d'information peut être vide si toutes les information d'un objet sont représentés par ces attributs.

Les attributs d'un objet représentent la vue externe de l'objet. Chaque attribut a un type et une ou plusieurs valeurs. Le type doit être globalement unique, il est souvent désigné par un *ObjectIdentifier* défini dans ASN.1 et distribué suivant un schéma hiérarchique. Le type d'attribut spécifie la *classe* sémantique de toutes les *instances* de l'attribut. La valeur d'un attribut donne la sémantique à une instance de l'attribut. Un objet peut avoir une ou plusieurs instances d'un même attribut, dans ce dernier cas, on parle de *multi-valeurs*, sinon de *mono-valeur*.

On pourra constater que certaines parties d'un objet sont statiques, par exemple, les attributs originels d'un message; et que certains autres sont dynamiques, e.g., les attributs additionnels d'un message. Donc un attribut peut être *statique* ou *dynamique*. Un attribut statique d'un objet ne peut être modifié, i.e. on ne peut ni ajouter, ni supprimer, ni remplacer d'instance de l'attribut dans l'objet. L'identificateur d'un objet est statique.

5. Entités de Communication

Chaque application MHS et GC a un *groupe* d'utilisateurs qui participent au processus de communication, i.e. un groupe d'entités de communication. Les fonctions des différents entités sont différentes dans le processus de communication, elles sont regroupés par *rôles*: ils représentent la structure de l'organisation de communication. A chaque instant, chaque utilisateur

joue un ou plusieurs rôles dans le processus de communication. Par exemple, le groupe des participants d'un Bulletin-Board a des rôles comme *administrateur*, *éditeur*, *auteur*, *lecteur*, etc. Une entité de communication peut jouer le rôle de lecteur et d'éditeur en même temps.

D'un autre coté, une entité de communication peut participer à plusieurs activités de communication (i.e. le processus de communication) et jouer différents rôles dans les différentes activités. Par exemple, une entité de communication peut être administrateur d'un Bulletin-Board ou lecteur d'une autre conférence, et en même temps envoyer et consulter des messages inter-personnels. On pourrait donc imaginer qu'une entité de communication possède plusieurs *instances* (le nombre est théoriquement illimité), la différence entre elles réside dans un sous-ensemble d'attributs associés; en particulier l'activité à laquelle l'unité de communication participe et le rôle qu'elle joue dans l'activité. Chaque recherche ou manipulation d'objets d'information est faite par une instance spécifique. Les droits d'accès aux objets d'information diffèrent d'une instance à l'autre; l'instance qui joue le rôle d'administrateur et celle jouant le rôle de lecteur ont des droits d'accès différents. Le contrôle d'accès s'appuie donc sur les instances.

6. Contrôle d'Accès

Le contrôle d'accès est effectué au niveau de l'environnement. Pour chaque pièce d'information manipulé (objets d'information, attributs, corps, parties de corps), le mécanisme de contrôle d'accès doit permettre de spécifier quelles opérations licites d'un communicateur peut effectuer telles opérations. Le contrôle d'accès consiste donc à gérer des triplets <Information, Opération, Communicateur>. Une technique est d'associer une liste de contrôle d'accès (ACL pour Access Control List) à chaque pièce d'information à manipuler, qui spécifie les droits pour les communicateurs (i.e. les droits d'effectuer les opérations, par exemple, le droit de *lire*, le droit de *créer*, le droit de *détruire*, etc.).

Chapitre 5
Conception de MAP

1. Introduction

1.1. Principes de Conception

MAP est l'outil de base pour gérer des objets d'information et les relations avec d'autres types d'objets. Il est l'élément commun à tous les archiveurs SAM (**Remarque**: pour MAP, un archiveur SAM est son **application**). La conception du MAP est dirigée par les principes suivants:

• MAP doit supporter le modèle conceptuel d'applications MHS et GC de SAM que nous avons décrit dans le chapitre précédent; en particulier, il doit stocker et gérer tous les objets d'information, traiter les environnements et les entités de communications (i.e. *utilisateurs* de MAP).

• MAP doit supporter la répartition et la distribution des objets d'information, et stocker les objets d'un environnement distribué. Le modèle conceptuel est indépendant du système de stockage et de sa réalisation, un objet d'information dans le modèle est bien connu et unique dans l'espace d'information. Cela n'est pas le cas pour les archiveurs distribués, MAP doit traiter les problèmes ainsi causés.

• MAP doit être simple. La *simplicité* signifie deux choses: l'utilisation est simple, et la réalisation est simple. Par exemple, seules les primitives sont fournies pour manipuler des objets d'information, cela permet d'avoir une réalisation simple; et les FE peuvent ne pas utiliser du tout certains mécanismes fournis par MAP, cela permet une utilisation simple.

• MAP doit être fonctionnellement complet et puissant (tout en gardant sa simplicité). Il doit être théoriquement possible de réaliser toutes les fonctionnalités envisageables à partir de celles fournies par MAP. Les primitives fournies doivent être complètes.

• MAP doit être efficace. Ceci est lié avec la simplicité. Une conception simple permet une utilisation et une implantation efficaces, mais une conception trop simple peut rendre l'utilisation inefficace et difficile. Par exemple, on a souvent besoin de lister des résumés d'un ensemble d'objets: les noms, les sujets, les expéditeurs, etc. Puisqu'on peut lire chaque objet individuellement, les résumés peuvent être obtenus en effectuant une série de *lire*. Ceci est inefficace, et a besoin d'autant d'interactions avec MAP que le nombre d'objets. Une meilleure solution est de fournir une opération *résumer* qui donne les résumés pour un ensemble d'objets. Donc pour garantir l'efficacité, MAP doit fournir non seulement les primitives, mais aussi les opérations fréquemment utilisées, qui peuvent être exécutées efficacement seulement si elles sont réalisées directement par MAP.

• MAP doit être assez flexible (flexibilité) pour pouvoir supporter différentes applications, et éventuellement des applications imprévues. Cela implique que MAP ne doit pas limiter les types d'objets et les types d'attributs qu'il peut supporter même s'il ne les traite pas tous. Par exemple, l'utilisateur souhaite souvent stocker avec les messages des objets comme: *post-it-note*, *mémo*, etc.

Il est évident que les buts sont parfois contradictoires, par exemple, la flexibilité dégrade souvent la performance puisqu'il complique le système. Dans la mesure du possible, nous choisirons le meilleur compromis.

1.2. Environnement Distribué

Les environnements où se situent nos applications sont distribués. Cela introduit des caractéristiques spécifiques. Dans un tel environnement, un archiveur seul ne gère pas tous les objets d'information, il doit coopérer avec les autres archiveurs pour fournir un service complet. Comme nous l'avons indiqué auparavant, la coopération est gérée par les FE de MAP. Cependant, pour pouvoir stocker un sous-ensemble d'objets d'information d'un environnement distribué, et fournir des supports pour la distribution et la synchronisation de l'information, il est nécessaire d'avoir d'abord un modèle du monde d'objets d'un tel environnement, et ensuite d'indiquer le rôle de MAP et des informations gérées par MAP dans ce monde. Le premier nous donne une vue distribué du monde d'objets, le second nous donne le modèle conceptuel de MAP. Une vue en couches de ces modèles est présentée dans la figure 5-1.

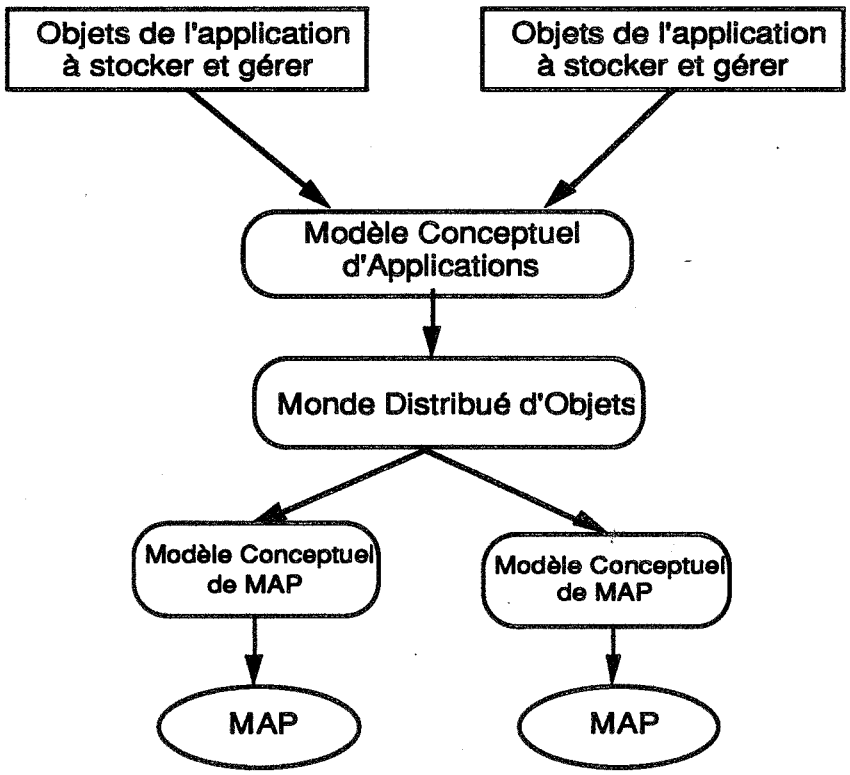


Figure 5-1. Vue en Couches des Modèles

2. Le Monde Distribué d'Objets

Les objets que nous avons spécifiés dans le modèle conceptuel d'applications sont en pratique distribués et répartis en différents endroits (Archiveurs, Agents de Service de Directory, etc). Dans ce chapitre, nous allons présenter des objets après leurs distributions dans un environnement distribué au point de vue des archiveurs de messages.

Le monde d'objets est composé de trois espaces différents mais liés.
L'*Espace d'Information* contient l'ensemble de toutes les instances des objets d'information.
L'*Espace d'Utilisateurs* contient toutes les entités de communication.
L'*Espace de Contextes* contient tous les contextes qui associent des utilisateurs aux instances d'objets d'information.

2.1. Espace d'Information

Chaque objet dans un environnement distribué est représenté par une ou plusieurs *instances* situées en différents endroits de l'environnement (différents archiveurs, par exemple). L'espace d'information contient toutes les instances de tous les objets d'information de l'environnement. Chaque objet d'information a au moins une instance. Les instances d'un objet atomique sont atomiques. Les instances d'un objet composite sont composites, elles se composent des instances des composants de l'objet. La figure 5-2 montre la relation entre des objets d'information et leurs instances.

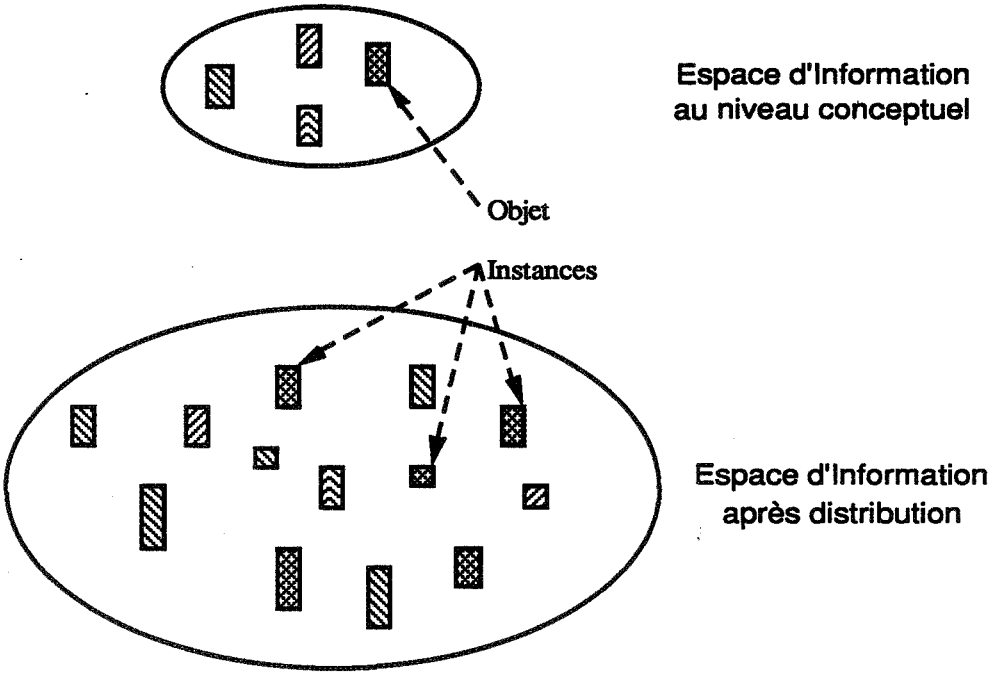


Figure 5-2. Espace d'Information

Les instances d'un même objet d'information sont réparties, chacune d'elles est dans un endroit différent. Chaque instance peut être stockée dans au plus un archiveur (sinon elle est en transmission ou est *perdue*). Les instances d'un même objet d'information ne sont pas obligatoirement identiques ou dans le même état. Elles sont même souvent différentes. En particulier, certaines instances peuvent être *incomplètes*: elles ne contiennent qu'une partie des informations considérées comme les caractéristiques essentielles de l'objet, par exemple, seuls le nom et quelques attributs de stockage (on l'appelle *ObjetNom*).

Il peut exister des relations autres que la *duplication* entre les instances d'un même objet. Par exemple, parmi les instances, il peut y avoir un *maître* qui est la version *standard* de l'objet, et

seulement les attributs globaux de celui-ci peuvent être modifiés; les autres instances sont dites *esclaves* de celle-ci, la relation entre le maître et l'esclave est dite *maître-esclave*. Il faut noter que les différentes stratégies de distribution et de synchronisation des instances d'objets définissent différentes relations entre les instances.

On pourra aussi avoir des contraintes sur le nombre maximum d'instances qu'un objet peut avoir; ainsi on a des objets *mono-instance* pour qui une seule instance est possible.

Les instances des objets conservent les relations entre les objets, toutes les relations entre deux objets existent aussi entre deux instances de ces deux objets. Cependant, les instances peuvent avoir des relations spécifiques aux deux instances. Cela est peut-être dû à l'archiveur ou simplement au fait que les instances sont dans des états différents.

2.1.1. Bases d'Information

L'espace d'information est regroupé en *bases d'information*. Une base d'information contient un ensemble d'instances où il n'existe pas deux instances du même objet. Une instance appartient au plus à une base d'information. Les différentes bases peuvent stocker les mêmes objets d'information, mais de différentes instances. Chaque base correspond à une **unité de stockage**, ainsi la distribution d'un objet consiste à distribuer ses instances dans les différentes bases et à gérer la cohérence entre ces instances dans ces différentes bases.

Puisqu'une instance appartient au maximum à une base d'information, les instances contenues dans une base d'information sont dites *spécifiques à la base*. Si une instance contenue dans une base est celle d'un objet *mono-instance*, l'objet est *local* à la base. Ni la distribution ni la synchronisation ne sont nécessaires pour les instances des objets locaux.

La figure 5-3 montre les relations entre l'espace d'information, les instances et les bases d'information.

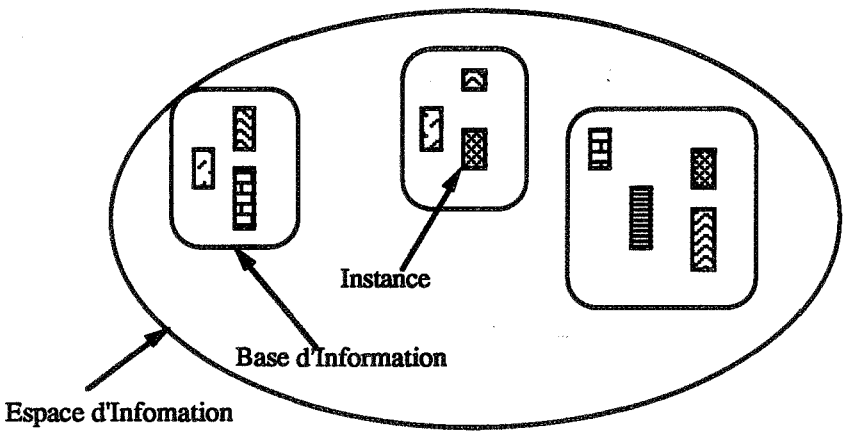


Figure 5-3. Bases d'Information

Les relations et la cohérence entre de différentes bases et les instances de ces bases sont décidées par les applications et sont assurées par les FE.

2.2. Espace d'Utilisateurs

L'espace d'utilisateurs comprend tous les utilisateurs. Les utilisateurs sont gérés par le système de *directory*, la gestion peut être distribuée (X.500 Directory) ou centralisée. Ce qui nous intéresse sont leurs relations avec les objets d'information et leurs images vues dans le monde d'objets.

Deux sortes de relations existent entre les objets d'information et les utilisateurs:

• Les utilisateurs manipulent les objets d'information (plus précisément leurs instances), ils lisent, recherchent ou modifient les objets d'information. Dans ce cas, les utilisateurs et les objets d'information sont liés par les droits d'accès des utilisateurs sur les objets d'information. Les droits d'accès permettent de contrôler les manipulations des utilisateurs sur les objets d'information. Ces relations sont enregistrées dans les *contextes* (les environnements dans le modèle conceptuel) comme informations de contrôle d'accès. Les images des utilisateurs vues dans le monde d'objets d'information sont les *instances* des entités de communication, dont certains attributs comme *activité* et *rôle* sont utilisés pour contrôler les accès.

• Les utilisateurs ont des liaisons sémantiques (par exemple, auteur d'un message) avec des objets d'information, elles sont présentées comme attributs des objets d'information. Ces relations sont stockées dans les bases d'information avec les objets.

2.3. Espace de Contextes

Un *contexte* associe des utilisateurs à des instances d'objets d'information d'**une base d'information** . Il fournit un environnement de travail pour les utilisateurs. Un contexte peut être:

• une instance d'un *environnement* du modèle conceptuel. Un environnement peut correspondre à un ou plusieurs contextes.

• une unité de distribution qui indique l'ensemble d'objets d'information à distribuer.

• une unité d'administration.

• un environnement de travail.

Les utilisateurs accèdent les instances d'information via les contextes qui spécifient la manière d'interaction et les relations entre les objets d'information et les utilisateurs.

Un contexte peut avoir des *sous-contextes,* chacun d'entre eux associe un sous-ensemble d'instances d'information à un sous-ensemble d'utilisateurs. L'espace des contextes est une *forêt* qui se composent de hiérarchies de contextes. Il peut y avoir aussi d'autres relations entre les contextes. Par exemple, un contexte peut être la *copie* d'un autre, i.e. toutes les instances référencées dans un contexte seront doublées dans sa copie. La maintenance de cette relation permet de distribuer un sous-ensemble d'objets d'information (dont les instances sont référencées dans le contexte) à une autre base située dans un autre archiveur. Les relations autres que *sous-contexte-de* ne sont pas définies au niveau MAP, elles doivent être gérées par les FE ou d'autres composants fonctionnels de l'application.

La figure 5-4 montre la relation entre les trois espaces.

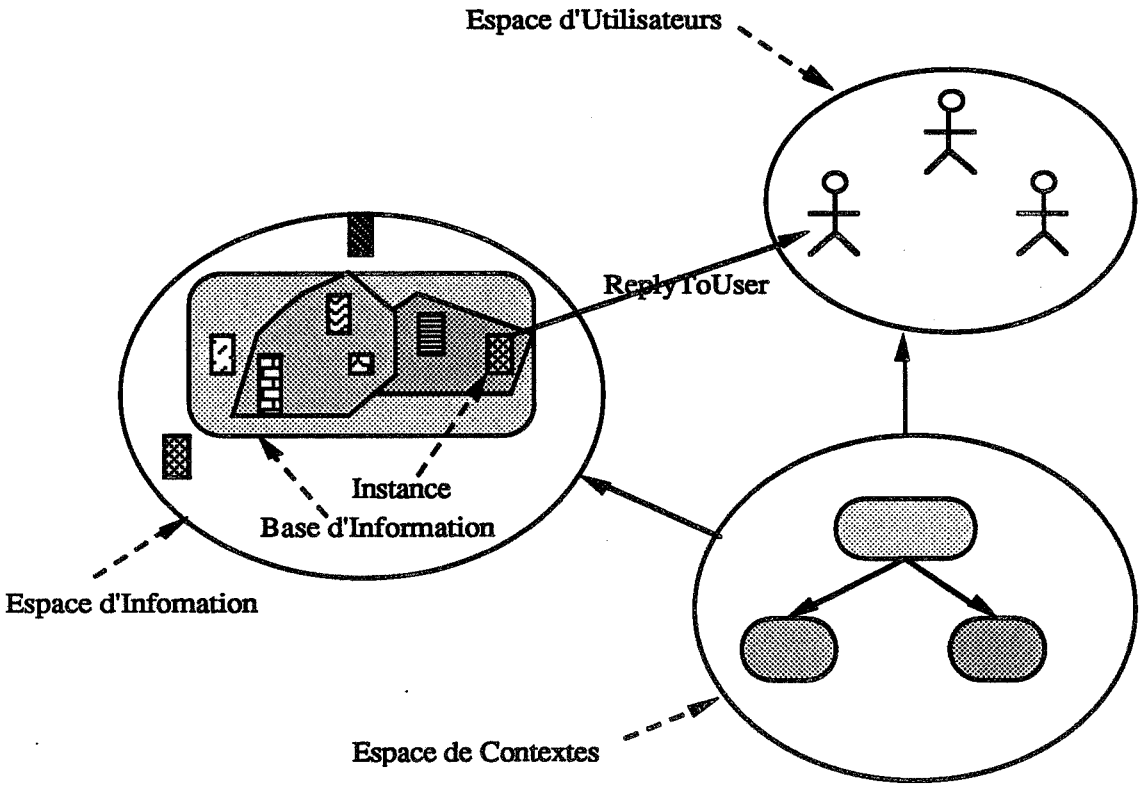


Figure 5-4. Relations entre les trois Espaces

3. Vue Externe de MAP

Chaque MAP stocke **une base d'information** et **un hiérarchie de contextes** qui permet aux utilisateurs d'accéder la base d'information. MAP gère la base d'information, la hiérarchie des contextes et les relations entre les utilisateurs et les instances des objets d'information de la base, ainsi que les contrôles d'accès (nous utiliserons *objets* pour les instances d'objets d'information par la suite dans le cas où il n'y a pas confusion, car un objet ne peut avoir qu'une seule instance dans une base).

MAP fournit des opérations pour lire, rechercher et manipuler les objets d'information et gérer les relations entre les objets d'information dans la base. Il fournit aussi les opérations pour manipuler les contextes, et les opérations pour gérer les relations entre les utilisateurs et les objets.

MAP gère aussi d'autres aspects concernant l'archiveur, il fournit un mécanisme de *trace* ou *historique* qui enregistre les opérations manipulant les objets d'information et les versions des objets d'information. Ce mécanisme permet d'utiliser intelligemment l'historique de la base.

Donc, on peut voir MAP comme une **machine virtuelle** qui fournit un service de stockage bien-défini via l'interface/protocole d'accès unique *Psam*. Cette machine gère:

- Une base d'information contenant des objets d'information.

• Un ensemble de contextes structurés (en hiérarchie) à travers lesquels les utilisateurs accèdent à la base d'information et manipulent les objets dans cette base.
• Les relations entre les objets d'information et les utilisateurs, les droits d'accès et les relations sémantiques.
• Le contrôle des accès des utilisateurs aux objets d'information d'après leurs relations avec les objets d'information.
• L'historique qui enregistre l'historique de la base d'information.

La figure 5-5 montre la vue *machine virtuelle* de MAP.

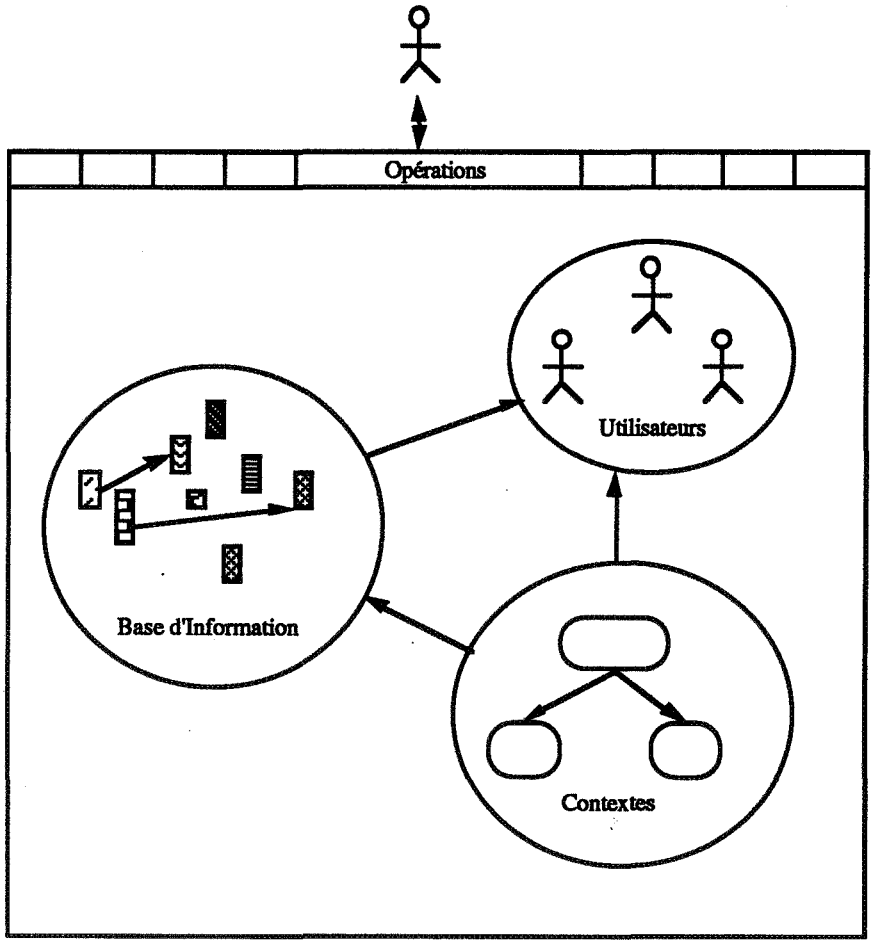


Figure 5-5. Machine Virtuelle MAP

Dans le reste de ce chapitre, nous allons présenter MAP et ses composants.

4. Caractéristiques Principales de MAP

MAP est un serveur général d'archivage pour des systèmes d'archivage d'applications MHS et GC. Combiné avec de différents FE, il forme des archiveurs SAM différents utilisés dans les diverses applications MHS et GC. Un unique protocole est défini entre les FE et MAP; sa conception et sa description formelle sont basées sur ASN.1 ([You 89f]). La tâche principale de MAP est de stocker des objets d'information et de gérer la base d'information ainsi formée. MAP

maintient de plus l'historique de ces objets. Il permet une recherche sophistiquée et des accès restreints via un mécanisme de contrôle d'accès.

La version actuelle de MAP stocke la base d'information et toutes les autres informations qu'il gère dans le système de fichiers de la machine hôte. La base d'information est stockée dans une zone centrale et accédée dans un environnement distribué. MAP communique avec les FE via le protocole Psam du style "question-réponse". Les FE peuvent ou non se situer sur la même machine que MAP.

Dans cette section, nous allons présenter les caractéristiques principales de MAP, et les illustrer avec l'exemple de la conférence MMM (cf. chapitre 2 §4).

Objets de MAP

Le modèle d'archivage/stockage de MAP est une combinaison du modèle *hypertexte* et du modèle *orienté-objet*. Il est basé sur les objets suivants: items, attributs, liens, relations item-utilisateur, classes et contextes. Les *items* sont des objets d'information de MAP. Chaque item se compose d'un ensemble d'attributs et d'un contenu. Les items peuvent être utilisés pour représenter des messages, des documents, des classeurs, etc. Les *attributs* sont attachés aux items. Ils représentent toutes sortes d'informations associées aux items, par exemple, le sujet ou les mots-clé d'un message. Les *liens* définissent des relations entre les items. Ils sont "généralement" représentés comme des attributs d'items. Par exemple, "InReplyTo" définit la relation "Question-Réponse" entre un item contenant la question et un item réponse. Il est représenté comme un attribut de l'item réponse. Les *relations item-utilisateur* représentent les liaisons sémantiques entre des items et des utilisateurs, par exemple, "auteur" entre un message et l'utilisateur qui l'a écrit; ou "expéditeur" entre un message et celui qui l'a envoyé. Les relations item-utilisateur sont aussi représentées comme des attributs. Les *classes* regroupent ensemble des items de même nature. La classe "classeur" regroupe tous les items représentant des classeurs, la classe "fiches" regroupe les messages de la conférence MMM. Les contextes partitionnent la base d'information de MAP. Chaque contexte contient un ensemble d'items, et fournit un environnement de travail aux utilisateurs pour accéder à ces items. Nous allons maintenant reprendre plus en détail chacun des éléments introduits ci-dessus.

Les **items** sont des objets d'information dans MAP. Ils forment la base d'information. Les items sont conçus pour représenter les divers objets d'information réels, atomiques ou composites, comme des messages, documents complexes, conversations, collections, classeurs, etc.

Un item est composé d'un ensemble d'attributs et d'un contenu. Les *attributs* sont basés principalement sur la norme X.400 Messages Inter-Personnels, et sur les propositions d'extensions du groupe AMIGO ([Smith 89], [Palme 85]). Il permet aussi des extensions spécifiques à l'application. Le *contenu* d'un item est un ensemble (ordonné ou non) d'éléments appelés parties de contenu (ou *CP* pour Content-Part). Un CP peut contenir du texte, de l'image, du son, ... de taille variable, mais aussi une référence à un autre item (un tel CP est appelé *RCP* pour Reference-Content-Part).

Une telle structure nous permet donc de représenter différentes sortes d'objets d'information par des items:

- La représentation des messages est triviale, puisque la structure des items est basée sur les Messages IP.

• Les objets *composites,* comme les classeurs ou les collections, sont représentés par des items dont les attributs décrivent ces objets. Les contenus contiennent les références à leurs "membres".

Par exemple, les messages de la conférence MMM sont représentés comme des items à chaque site. Les éventuels classeurs sont aussi des items dont les RCP contiennent les références aux messages dans les classeurs. Ils forment la base d'information de l'archiveur du site.

MAP permet aussi l'utilisation d'items *implicites.* Le contenu d'un tel item est un ensemble de références (RCP) obtenues dynamiquement selon une règle de dérivation, lorsque l'item est lu. Les items implicites sont utilisés pour représenter des objets comme des conversations automatiques. Par exemple, il sera intéressant de grouper des messages autour d'un message initial par une conversation automatique. Elle permet non seulement de lire directement les messages de la conversation, mais aussi les descriptions de la conversation représentées comme attributs de la conversation. Dans la conférence MMM, la règle de dérivation peut être "`tous les messages liés directement ou indirectement au message initial INRIA+190`".

Suite au besoin local et à cause des défauts de communication, un item peut être *incomplet* dans un environnement distribué de messagerie. MAP supporte une telle notion d'"incomplétude", et permet d'avoir des items incomplets. Par exemple, après l'effacement d'un message dans la conférence MMM, l'item représentant le message devient incomplet, et ne contient plus que les identifiants et les liens.

Les **attributs** sont attachés aux items. Ils donnent une sémantique aux items et représentent les liens entre des items ou les relations entre des items et des utilisateurs. Un attribut est identifié par son type, et a une valeur et un ensemble fixe de descripteurs. On appelle la composition <type valeur descripteurs> une instance de l'attribut. Plusieurs attributs (instances) du même type peuvent être attachés au même item.

Pour faciliter et simplifier tant la réalisation que l'utilisation, un "grand" sous-ensemble d'attributs est prédéfini (cf. Annexe A), il est basé sur les attributs des messages inter-personnels, des classeurs et des extensions proposées par AMIGO+. Ces attributs ont des sémantiques prédéfinies, par exemple, la date de la dernière consultation, l'identificateur, l'auteur de l'item, etc. Seuls ceux-ci peuvent être utilisés dans la recherche des items.

Outre les attributs prédéfinis, MAP permet aux FE d'introduire leurs propres attributs spécifiques à l'application. Cependant MAP ne connaît pas leurs sémantiques et ne permet pas de les utiliser pour la recherche.

Les valeurs d'attributs peuvent être des chaînes de caractères, des entiers, des booléens, des types définis spécifiques à MAP comme ObjectDescriptor, OID (identificateur objet), ou des chaînes d'octets. La chaîne d'octets peut être vue comme un type de données générique (polymorphisme), qui permet aux FE de définir leurs propres types de valeurs et de les stocker comme des chaînes d'octets dans MAP. Les attributs attachés aux messages de la conférence MMM sont tous prédéfinis dans MAP

Avoir un ensemble prédéfini d'attributs garantit performance et simplicité dans la réalisation de MAP, tandis que permettre des attributs additionnels spécifiques à l'application rend MAP flexible et extensible.

Un **lien** définit une relation entre un item source et un item destination. Il est typé et orienté. Outre le type et les références à l'item source et à l'item destination, un lien peut avoir une information

associée qui décrit le lien. Par exemple, l'item source est lié à tel CP (Partie de Contenu) de l'item destination si seul un CP de l'item destination est concerné. On peut suivre un lien dans les deux sens.

Les types des liens et des attributs partagent le même espace d'identification (à savoir ObjectIdentifier défini dans ASN.1). MAP stocke un lien soit comme un attribut de même type attaché à l'item source (le type de l'attribut correspond au type du lien et la valeur est une référence à l'item destination), soit comme un RCP de l'item source qui représente un type particulier de lien: *membership* (appartenance) et qui contient la référence à l'item destination. MAP prédéfinit un ensemble de liens en prédéfinissant les attributs. Il accepte aussi tous les liens spécifiques à l'application. A l'exception des liens *membership*, il traite tous les liens de façon identique. Tous les liens peuvent être utilisés pour la recherche. Il est interdit d'avoir une boucle sur les liens *membership*. De plus les liens *membership* ont des utilisations spécifiques dans la recherche d'items (voir la section correspondante).

On peut non seulement suivre les liens individuellement comme dans un système hypertexte, mais il est aussi possible de rechercher récursivement les items d'après les liens en mode "batch". Par exemple, deux types de liens ont été définis dans la conférence MMM: "InReplyTo" qui lie une réponse à la question posée et "CommentOn" qui relie un commentaire au message qu'il commente. Ils sont représentés comme des attributs des messages sources (i.e. la réponse ou le commentaire). Les messages de la conversation autour du message "INRIA+190" peuvent être obtenus en cherchant "`tous les items liés directement ou indirectement au message` *`INRIA+190`* `par les liens des types` *`InReplyTo`* `ou` *`CommentOn`*".

Une **relation item-utilisateur** représente une liaison sémantique entre un item et un utilisateur. Par exemple, l'utilisateur est l'"auteur" de l'item. Chaque relation item-utilisateur est typée et représentée comme un attributs de l'item concerné. Le type de la relation et le type de l'attribut sont identiques. MAP prédéfinit un ensemble de relations item-utilisateurs et permet aux FE d'introduire leurs propres relations item-utilisateur. Il les traite toutes de la même façon, qu'elles soient prédéfinies ou spécifiques. On peut rechercher des items d'après leurs relations avec les utilisateurs, il est possible de trouver "`tous les items` *`concernant`* `(qui ont une relation quelconque avec) l'utilisateur YOU`".

Comme dans un système orienté-objet, une **classe** regroupe un ensemble d'items de même nature. Elle définit le type de ces items et stocke les actions spécifiques à cet ensemble. Le type d'un item spécifie les attributs que cet item doit avoir (i.e. les attributs obligatoires) et les conditions que leurs valeurs doivent satisfaire. La sémantique des actions spécifiques n'est pas particulière à MAP. Les actions sont simplement stockées comme des chaînes d'octets. Chaque item appartient à une classe dont il est une *instance directe*. Les classes forment une hiérarchie dont la racine (nommée *items*) est la super-classe de toutes les autres classes. Tous les items sont des instances directes ou indirectes de cette super-classe. Les FE peuvent dynamiquement ajouter ou supprimer des classes.
Par exemple, les messages de la conférence MMM forment une classe "fiches", les attributs obligatoires sont "identifiant global", "identifiant local", "auteur" et "sujet".

Les **contextes** partitionnent la base d'information. Ils fournissent des environnements de travail, et des unités d'administration et de distribution. Un contexte définit la visibilité (les attributs

visibles) de ses *éléments* (les items qui appartiennent au contexte), et contrôle les accès aux éléments. Un utilisateur accède à la base d'information via un contexte. MAP supporte les contextes comme des unités logiquement indépendantes, en fournissant à chaque contexte son propre historique conceptuel. Les contextes forment une hiérarchie. Chaque contexte peut avoir zéro, un ou plusieurs parents (super-contextes) et zéro, un ou plusieurs fils (sous-contextes). Un parent a tous les éléments de ses fils comme éléments indirects; il peut lire et rechercher parmi tous les éléments du contexte, qu'ils soient directs ou indirects. Un seul contexte peut ne pas avoir de parent. Il est top-contexte, qui est la racine de la hiérarchie, à partir de laquelle tous les éléments sont visibles. Il est possible de créer, supprimer ou modifier un contexte dynamiquement.

Par exemple, les sous-thèmes de la conférence MMM correspondent aux contextes, ils ont besoin d'être administrés et distribués indépendamment.

Contrôle d'Accès

MAP fournit un mécanisme de sécurité pour contrôler les accès à la base d'information via les contextes. Il permet aux FE de définir un ensemble de **contrôleurs** et d'associer un contrôleur à chaque contexte. Ce contrôleur est utilisé pour contrôler tous les accès via le contexte. La partie essentielle d'un contrôleur est une liste de contrôle d'accès (ACL pour Access Control List) qui accorde les droits d'accès aux utilisateurs.

Le modèle utilisateur est basé sur celui de l'"AMIGO Activity Model" ([Pankoke 88]). Chaque utilisateur d'une *activité* joue un ou plusieurs *rôles* et accède à la base avec un rôle précis. Les utilisateurs sont donc décrits dans une ACL par leurs *identificateurs*, les *activités* auquelles ils participent et les *rôles* qu'ils jouent dans ces activités. Tous ces paramètres sont affectés par les FE spécifiques à chaque application. Par exemple, la conférence MMM est une activité à trois rôles: administrateur, auteur et lecteur.

Les droits d'accès sont aussi définis par les FE, ils sont traités par MAP comme des entiers. Chaque accès à la base d'information via un contexte réclame un ou plusieurs droits d'accès et spécifie l'utilisateur qui demande l'exécution de l'opération, i.e. son identificateur, l'activité à laquelle il participe et le rôle qu'il joue lorsqu'il lance l'opération. Ces droits d'accès sont vérifiés auprès du contrôleur associé au contexte.

Un contrôleur est dynamique, et on peut dynamiquement modifier l'ACL d'un contrôleur. Il est aussi possible de changer dynamiquement le contrôleur d'un contexte.

Ce mécanisme est basé sur le fait que MAP coopère avec les FE pour réaliser un archiveur. Ce sont donc les FE qui identifient l'utilisateur et spécifient les droits d'accès nécessaires. Ce mécanisme est particulièrement flexible: lorsque les données à manipuler (items, attributs, etc) sont différentes, les FE peuvent réclamer de différents droits d'accès pour différentes exécutions de la même opération.

En utilisant ce mécanisme, le contrôle d'accès de la conférence MMM est très simple, un seul contrôleur est associé à chaque contexte. Pour construire l'ACL du contrôleur, il suffit d'affecter un numéro à chaque groupe d'opérations, et l'associe aux rôles appropriés: <administrateur: tous les droits>, <auteur: stocker et lire>, <lecteur: lire>.

Mécanisme d'Historique

MAP fournit un mécanisme d'historique pour enregistrer et gérer les informations historiques de la base d'information. L'ensemble des informations historiques de la base d'information est appelé **historique**. L'historique enregistre les **événements** manipulant la base et les **versions** des

items, automatiquement ou à la demande des FE. Les événements et les versions peuvent ensuite être utilisés par les FE pour la distribution/synchronisation entre différents archiveurs et/ou la restauration/récupération de la base après un incident. MAP permet aussi de détruire des événements et des versions inutiles.

Dans la conférence MMM, l'archiveur du site INRIA (niveau 1) utilise l'historique pour mémoriser toutes les opérations modifiant la base d'information en gardant les événements correspondants. Lorsqu'une demande de distribution arrive d'un site niveau 2 qui lui est attaché (EMSE ou Paris6), l'archiveur trouve, dans l'historique de la base, toutes les modifications faites dans les contextes correspondants aux sous-thèmes abonnés par le site niveau 2, depuis la dernière demande du même site niveau 2 .

Manipulations de la Base d'Information

MAP fournit six catégories d'opérations pour manipuler la base d'information formée par les items, les attributs attachés, les liens entre les items et les relations item-utilisateur:

- *Opérations Stocker* stockent un nouvel item ou complètent un item existant. Une opération stocker prend des données spécifiques à l'item et rend un identificateur interne de l'item.
- *Opérations Effacer* suppriment ou effacent une partie d'un item.
- *Opérations Modifier un attribut* ajoutent, suppriment ou remplacent un attribut (ou une instance) attaché à un item.
- *Opérations Modifier le contenu* remplacent le contenu, ajoutent, suppriment un partie du contenu (CP) d'un item. Puisque les liens et les relations item-utilisateur sont représentés comme des attributs et des RCP, les opérations "modifier" permettent de les manipuler.
- *Opérations Rechercher* trouvent dans un contexte les items parmi les éléments (directs ou indirects) du contexte, selon un critère de choix sur les attributs prédéfinis, les liens, les relations item-utilisateur et les types de CP. Le résultat d'une telle opération est une liste d'identificateurs et optionnellement un sous-ensemble d'attributs attachés aux items (i.e. *résumer*).
- *Opérations Lire* comprennent les opérations pour lire un item, obtenir les liens entre les items et obtenir les relations item-utilisateur. Il est possible de lire seulement une sous-partie d'un item comme un sous-ensemble d'attributs, ou un sous-ensemble de CP qui ne contiennent que du texte. Cela est particulièrement économique lorsque certains CP ne peuvent pas être lus à cause des contraintes matérielles ou logicielles (par exemple, une image quand on ne possède qu'un écran alphanumérique). Il est aussi possible d'obtenir les liens entrants d'un item.

Le lecteur trouvera les descriptions détaillées de tous les mécanismes, de tous les objets et leurs manipulations dans les sections qui suivent.

5. Modèle d'Information de MAP

Dans cette section, nous présentons le modèle d'information de MAP, i.e. les objets d'information de MAP, leurs représentations, leurs propriétés et les relations entre eux ainsi que leurs relations avec des utilisateurs, et les opérations disponibles pour les manipuler.

Avant de présenter les détails, on doit noter que l'objectif de MAP est de définir un outil puissant et flexible; cependant les utilisations *correctes* et *cohérentes* de l'outil sont à la charge des FE d'applications. En particulier, au niveau administratif, MAP ne garantit pas la cohérence absolue s'il n'est pas utilisé correctement.

5.1. Objets d'Information

MAP gère une base d'information qui contient d'objets d'information (les instances des objets d'information). Les objets d'information supportés par MAP doivent pouvoir représenter les différentes sortes d'objets d'information des applications (identifiés dans le modèle conceptuel), i.e. objets composites et objets atomiques, et leur permettre d'avoir différentes propriétés, comme les objets locaux, objets globaux, etc.

La solution de MAP consiste à proposer **un type général** d'objets d'information, appelés **items**. Tous les items ont la même structure globale et subissent les mêmes opérations dans MAP. Dans le texte suivant, *items* et *objets* seront utilisés indifféremment pour désigner un objet d'information dans la base de MAP.

5.1.1. Structure d'un Item

La structure choisie est dérivée directement des structures d'objets d'information dans le modèle conceptuel. Chaque item est composé d'un **en-tête** et d'un **contenu**. L'en-tête est un ensemble d'attributs représentant de façon structurée les différents aspects de l'objet réel, par exemple, l'identificateur, le sujet, etc. Le contenu contient la *vraie* information de l'objet réel, souvent de façon non-structurée, comme le corps d'un message, par exemple. Le contenu est composé d'un ensemble (ordonné ou non) de **parties de contenu** (ou **CP** pour Content Part), chaque CP peut être un segment de texte (d'une taille théoriquement illimitée), une image, un graphe, un son, ... ou une référence à un autre objet. La figure 5-6 montre la structure d'un item.

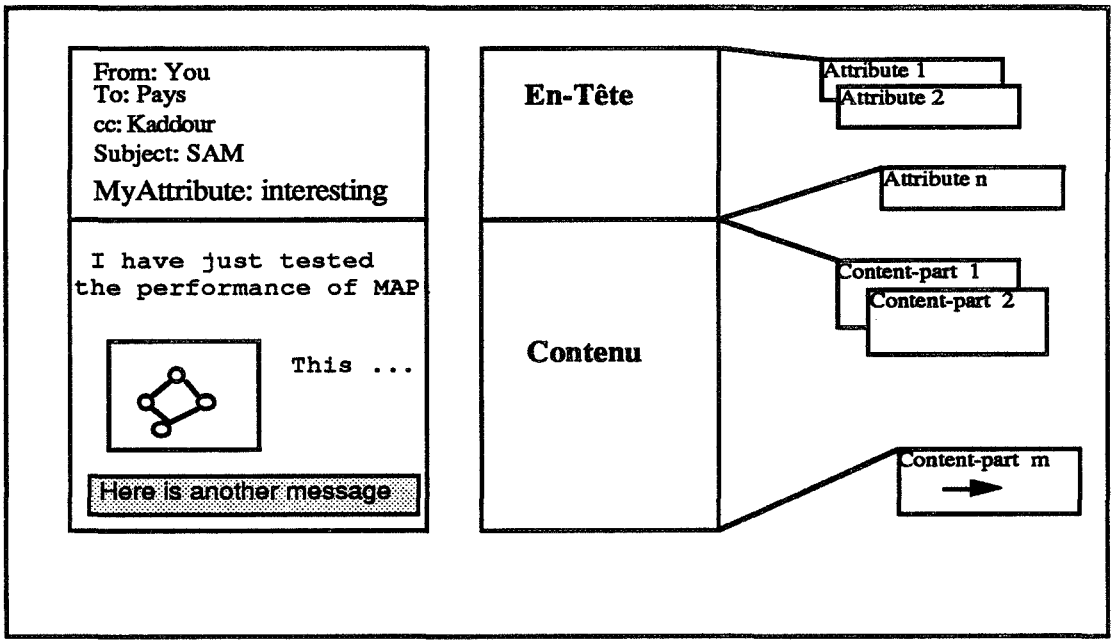


Figure 5-6. La structure d'un Item

5.1.2.Identification d'Items

Pour pouvoir référencer et retrouver des items, au moins un identificateur doit être attaché à chaque item de la base. MAP permet à chaque item d'avoir deux identificateurs différents:

(1) Un **identificateur interne** (OID pour Object IDentifier) qui est attribué à chaque item par MAP quand l'item est stocké pour la première fois dans la base. L'OID identifie uniquement l'item dans la base et permet des accès rapides à l'item. Un OID est un numéro de séquence. On appelle aussi l'OID d'un item son *nom interne.*

(2) Un objet global peut avoir optionnellement un **nom externe** qui l'identifie globalement. De plus, des objets sont souvent créés en dehors de l'archiveur qui gère la base; ils sont référencés par leurs noms externes. Ceci implique que MAP doit supporter des noms externes au moins comme références. Par exemple, chaque message IP a un identificateur externe (IPM-ID) qui est utilisé par l'expéditeur et ses destinataires pour référencer le message. Pour les messages de la conférence MMM, chacun d'entre eux a un identificateur global qui est affecté lors de sa création dans un site; cet identificateur se sert du nom global du message dans tous les sites de la conférence.

Les formats des noms externes peuvent être différents d'un espace d'information à l'autre. Ils peuvent être très compliqués. Cependant, la plupart des formats (sinon tous) ont une représentation en *chaîne de caractères*. En fait, il est nécessaire de présenter les noms sous un format humainement lisible pour n'importe quel format d'identification. Par exemple, un IPM-ID (Identificateur d'un message IP) est représenté comme un O/RName plus une chaîne de caractères imprimable. Et un O/RName est représenté en ASCII par une convention <mot-clé, valeur> (cf. [Grimm 89]), soit par exemple, <C="fr"; ADMD="atlas"; PRMD="emse"; O="emsesmc"; S="you">. Donc nous définissons un nom externe comme une chaîne de caractères. Le coût de cette simplification est la perte de structure au niveau MAP, mais cela peut être très bien géré au niveau de Front_Ends par une convention, <mot-clé, valeur>. Dans ce dernier cas, le *pattern*

matching sur chaîne de caractères permet de manipuler la structure des noms. Par exemple, il est possible de chercher tous les messages créés à l'école des Mines de St-Etienne en cherchant les noms contenant <PRMD="emse"> dans le cadre de la messagerie X.400.

Donc, chaque objet d'information dans la base a un identificateur interne et/ou un nom externe. Le nom interne est affecté par MAP, le nom externe est donné par l'application de MAP. MAP gère la correspondance des noms. Le nom interne permet non seulement une identification simple et efficace (un numéro de séquence au lieu d'une chaîne de caractères ayant une structure implicite), mais aussi permet d'identifier des objets locaux qui peuvent ne pas avoir de noms globaux externes.

5.1.3. Attributs

Chaque item comprend un ensemble d'attributs représentant ses différents aspects et caractéristiques:

- Noms: les identificateurs d'un item sont représentés comme des attributs de l'item. Deux attributs sont définis: *OID* et *ExternalName* représentant respectivement le nom interne et externe de l'item.

- Liens entre l'item et d'autres items: les attributs peuvent représenter les liens ou relations avec d'autres items, comme InReplyTo (RéponseA), CommentOn (CommentaireA), VersionOf (VersionDe), etc.

- Relations avec les utilisateurs: ils représentent les relations sémantiques avec les utilisateurs, comme ReplyToUser (RéponseAUtilisateur), Expéditeur, Destinataires, etc.

- Propriétés diverses: il permettent de représenter de diverses propriétés d'un objet, par exemple, Importance, Priorité, Sujet, etc.

On peut manipuler les attributs d'un objet dynamiquement: ajouter un attribut, supprimer un attribut, lire un ou plusieurs attributs d'un item sont des opérations licites.

5.1.3.1. Composition d'un Attribut

Chaque attribut d'un item est composé d'un *type*, d'une ou plusieurs *valeurs*, et d'un ensemble de *descripteurs*. Le type spécifie la *classe* de l'attribut, les valeurs sont des *instances* de la classe associées avec l'item, les descripteurs spécifient les caractéristiques de l'attribut et des instances, par exemple, la structure des valeurs (un entier, une chaîne de caractères, ou une donnée structurée), si l'attribut est modifiable, etc. On dit qu'un item contient l'attribut du type **T**, si une ou plusieurs instances du type sont attachées à l'item.

5.1.3.2. Type

Chaque type est identifié par un *ObjectIdentifier* (cf. [ASN.1]) qui est une séquence ordonnée de nombres permettant une affection hiérarchique. Cette représentation permet d'interfacer MAP avec les autres standards si besoin. Il faut noter que la représentation *ObjectIdentifier* peut être complètement locale, puisque seule la syntaxe est définie par ASN.1.

En théorie, MAP accepte **tous** les types d'attributs. Mais il en distingue deux catégories:

- Attributs Prédéfinis: Les sémantiques de ces attributs sont connues par MAP, les identificateurs des types et certains descripteurs (en particulier, la structure de valeurs) sont définis par MAP.

- Attributs Spécifiques: les attributs de cette catégorie ne sont pas connus à l'avance par MAP, ils sont définis dynamiquement par l'application. Les raisons de cette classification et ses conséquences seront présentées dans la section 5.1.3.6.

Parmi les attributs prédéfinis, il existe un ensemble d'attributs qui sont générés et modifiés automatiquement par MAP; ce sont souvent des attributs liés au stockage: la date d'entrée, la date de la dernière modification, etc.

5.1.3.3. Valeurs

Chaque attribut d'un item est représenté par une ou plusieurs valeurs qui sont les instances de l'attribut. Dans le premier cas, on dit que l'attribut est *mono-valeur*, sinon il est dit *multi-valeurs*. Sans confusion, l'expression "attribut (du type) T d'un item" est utilisée pour désigner l'ensemble des instances de l'attribut du type T attachées à l'item.

La structure des valeurs d'un attribut est spécifiée par le descripteur *type de valeurs* qui définit la structure de données des valeurs. Les différents attributs (i.e. les attributs de différents types) ont souvent des différents types de valeurs. On ne peut pas prévoir tous les types de valeurs nécessaires dans un archiveur général. Une solution est de permettre de définir dynamiquement des types de données en fournissant un langage de définition de types de données (par exemple, ASN.1). Cette solution est trop compliquée, et n'est pas non plus nécessaire. En effet, le type de données peut servir à:

- examiner la conformité des données; ceci est généralement fait par les interfaces des utilisateurs ou des FE qui génèrent les attributs,
- permettre de manipuler non seulement la valeur totale, mais aussi les composants de la valeur. Cela n'est pas non plus justifié par rapport à la complexité introduite.

La solution de MAP est à la fois performante et flexible. Elle définit un grand ensemble de structures de valeurs fréquemment utilisées (Date, ObjectDescriptor, etc), en plus des types simples (Entier, Booléen, ...). En particulier, nous permettons aux FE d'avoir leur propre *présentation*, i.e. MAP permet de stocker des données brutes qui sont traitées comme chaînes d'octets. Une liste plus complète des types de valeurs prédéfinis est donnée dans la description des descripteurs et dans [You 89c].

5.1.3.4. Descripteurs

Les descripteurs spécifient les propriétés globales d'un attribut dans un item, par exemple, le nombre de valeurs, ou les propriétés spécifiques à une seule instance (si l'instance est modifiable, si elle est originelle, etc).
Nota: pour un attribut *mono-valeur*, les descripteurs spécifiques aux instances s'appliquent aussi à l'attribut total.
Ils permettent à MAP d'effectuer des traitements appropriés et de fournir des services appropriés. Par exemple, on peut obtenir tous les attributs originels d'un message, mais on ne peut modifier une instance statique.

Les descripteurs suivants sont définis par MAP :

Type d'information

Ce descripteur spécifie le type d'information que l'attribut représente. Il peut être:

- Lien si l'attribut représente des liens,
- Utilisateur si l'attribut représente des relations item-utilisateurs,
- InformationDeDistribution: ce sont des attributs utilisés pour la distribution/répartition d'informations. Ils spécifient les relations entre différentes instances d'un même objet, qui se trouvent dans différents archiveurs,
- Propriété pour les autres sortes d'information.

Ce descripteur est le même pour toutes les instances d'un même attribut dans tous les items. Puisque les *Liens* et les *relations item-utilisateur* sont traités de façon particulière, ce descripteur permet d'indiquer à MAP si une instance de l'attribut doit être traitée comme un lien ou une relation item-utilisateur.

Type/Structure de Valeurs

Il spécifie la structure de données pour les valeurs de l'attribut. Toutes les instances d'un même attribut ont la même structure de valeurs. Les types de valeurs suivants sont définis par MAP:

- Booléen
- Entier
- Chaîne de caractères
- Chaîne d'octets: une valeur de ce type représente des données brutes ou génériques du point de vue de MAP.
- Date UTC
- OBJECT IDENTIFIER: la structure de données pour les identificateurs de classes, etc.
- OID: la structure de données pour les identificateurs internes d'items.
- ObjectDescriptor: la structure de données pour les références aux item. Elle est la réunion d'un OID et/ou un ExternalName, avec une petite complément d'information. Une valeur de ce type fait référence à un item.
- UID: le format de données pour les identificateurs d'utilisateurs.
- UserDescriptor: le format de données pour les références aux utilisateurs.
- CPType: le format de types de parties de contenu. C'est un entier avec la sémantique prédéfinie.
- Selector: le format de données des sélecteurs d'items; ces derniers spécifient les conditions que doivent satisfaire les items à sélectionner.

I-Statique/I-Dynamique

Ce descripteur est spécifique à une instance d'attribut. Une instance d'un attribut (d'un objet) est *i-statique* si elle n'est pas modifiable (remplacer, supprimer) contrairement à une instance *i-dynamique* qui peut être modifiée dynamiquement.

A-Statique/A-Dynamique

Ce descripteur est spécifique à l'ensemble des instances de l'attribut d'un item. Un attribut est a-statqiue s'il n'est pas modifiable (ajouter, remplacer, supprimer une ou plusieurs instances). Les instances d'un attribut a-statiques sont obligatoirement i-statiques. Cependant les instances d'un attribut a-dynamiques peuvent être i-statiques ou i-dynamiques.

Pour un item représentant un message stocké, les attributs/instances statiques peuvent représenter des attributs/instances initialement affectés par son expéditeur, et les attributs/instances dynamiques représenter les attributs additionnels. Les attributs statiques d'un item représentent souvent les caractéristiques essentielles de l'item, et leurs modifications changent complètement l'item. Les identificateurs (les noms interne et externe) sont représentés comme des attributs statiques. Les attributs générés par MAP sont souvent dynamiques, par exemple la date de la dernière consultation.

Multi/Mono-Valeur

Ce descripteur est spécifique à l'attribut d'un item. Il spécifie combien d'instances de l'attribut un item peut avoir, i.e. une seule ou plusieurs. Les instances d'un attribut multi-valeurs d'un item forment un ensemble. Les utilisateurs peuvent insérer ou retirer une valeur de l'ensemble. Des liens sont souvent multi-valeurs.

Originel/Additionnel

Ce descripteur est spécifique à l'instance individuelle. Une instance est originelle, si elle est attaché à l'item par le créateur de l'item, une instance additionnelle d'un attribut peut être ajoutée par l'*éditeur* de l'item. Ce descripteur permet de *marquer* les attributs originels et de les extraire séparément. Imaginons qu'un nouveau rôle "éditeur" soit ajouté dans la conférence MMM. Celui-ci a le droit d'ajouter de nouveaux attributs aux messages soumis par des auteurs. Dans ce cas, il est important de pouvoir distinguer les attributs originels attachés par l'auteur et ceux ajoutés par un éditeur.

Universel/Spécifique

Ce descripteur est spécifique à l'attribut d'un item. Un attribut est défini comme *universel* dans MAP s'il doit être présent dans tous les items de la base d'information, sinon il est *spécifique* à l'item.

Global/Local

Ce descripteur est spécifique à l'instance individuelle. Une instance d'un attribut d'un objet est **globale** si elle est reconnue par toutes les instances de l'objet que représente l'item. Une instance **locale** est significative seulement dans la base. Ce descripteur est généralement utilisé dans des archiveurs distribués pour indiquer si les modifications de l'instance doivent être propagées ou non aux autres archiveurs. Il peut y avoir, pour le même attribut du même item, une instance globale et une instance lobcale. Ce descripteur n'est pas significatif dans le sens où le traitement pour une instance globale ou locale est strictement identique au niveau MAP. Les instances des attributs de distribution sont souvent *locales*. Pour la conférence MMM, il peut être intéressant pour un site d'ajouter des attributs locaux aux messages du site sans qu'ils soient visibles des autres sites.

Recherchable/Non-Recherchable

Ce descripteur est spécifique à l'attribut. Il spécifie si l'attribut peut être utilisé pour la recherche, i.e. parmi les critères de recherche. Seul un sous-ensemble d'attributs prédéfinis par MAP ainsi que les liens et les relations item-utilisateur sont utilisables pour la recherche. On peut cependant rendre un attribut de ce sous-ensemble non-recherchable. Cela permet d'avoir plus de sécurité en excluant les attributs confidentiels de la recherche, et aussi une réalisation plus efficace en n'indexant pas les attributs non-recherchables.

Les descripteurs d'un attribut peuvent être enregistrés dans le DS avec la définition de l'attribut, et/ou être enregistrés dans la base. Le dernier choix est souhaitable à cause de l'indisponibilité de DS et du fait qu'un attribut local peut ne pas être enregistré dans le DS. De plus, certains descripteurs d'un attribut sont différents d'une instance à l'autre; une instance du même attribut peut être *i-statique* dans un item mais *i-dynamique* dans un autre.

D'après le domaine de validité des descripteurs, on peut classer les descripteurs en deux catégories:

(1) **Universel**: pour chaque attribut les descripteurs de cette catégorie sont les mêmes pour tous les items qui contiennent l'attribut. Ils peuvent donc être définis une fois pour toutes dans MAP. Ces descripteurs sont:

- Type d'Information
- Type de Valeurs
- A-Statique/A-Dynamique
- Multi/Mono-Valeur
- Universel/Spécifique
- Recherchable/Non-Recherchable

(2) **Instance Spécifique**: les descripteurs de cette catégorie sont spécifiques à l'instance individuelle de l'attribut. Ils peuvent être différents d'une instance à l'autre. Ce sont:

- I-Statique/I-Dynamique
- Originel/Additionnel
- Global/Local

Les affectations des descripteurs de la catégorie *universel* sont faites une fois pour toutes dans MAP. Ces descripteurs sont alors stockés avec les définitions des attributs. Tandis que les descripteurs de la catégorie *instance spécifique* sont associés aux instances individuelles, et sont affectés aux instances lorsqu'elles sont attachées à des items.

5.1.3.5. Définitions d'Attributs

On doit d'abord définir un attribut avant de l'utiliser dans MAP. Donc MAP contient l'ensemble des définitions de tous les attributs qui peuvent être utilisés dans MAP. La définition d'un attribut se compose:

- de la déclaration du type de l'attribut
- des affectations des descripteurs **universels** qui ne peuvent pas être redéfinis dans les items.
- des valeurs par défaut pour les descripteurs **instances spécifiques**. Ces descripteurs peuvent être redéfinis (en prenant la même valeur ou pas) dans chaque instance.
- d'un petit complément d'information supplémentaire. Elle représente des descripteurs supplémentaires de l'application, comme par exemple, la nature du lien dans le système CRMM (cf. chapitre 2 §3.6).

MAP permet aux FE de manipuler les définitions des attributs. Les FE peuvent *définir un nouvel attribut*, *changer la définition d'un attribut*, ou *lire les définitions d'un* ou *plusieurs attributs*.

Cependant, la modification de la définition d'un attribut n'est pas aléatoire; on doit respecter certaines règles pour avoir une cohérence avec la définition existante. Ces règles sont:

- Les descripteurs de la catégorie *universel* ne peuvent être modifiés.
- Les valeurs par défaut peuvent être modifiées librement. Ce type de modification n'a pas d'effet de bord, et les descripteurs associés aux instances ne sont pas modifiés.

5.1.3.6. Attributs Prédéfinis

Une des différences entre une base de données générale et MAP est que les objets supportés dans MAP sont des objets des applications MHS et GC, et en particulier des messages. Certains attributs communs sont fréquemment utilisés dans ces applications.

Pour des raisons de simplicité et performance, MAP choisit de prédéfinir un ensemble d'attributs tout en permettant de stocker tous les autres types d'attributs. Cet ensemble est basé sur les normes X.400, et RFC-822, ainsi que sur les propositions d'AMIGO pour MHS et GC, et les besoins de stockage de MAP. Les attributs prédéfinis sont traités de façon plus efficace et plus complète. Par exemple, seul un sous-ensemble d'attributs prédéfinis est utilisable pour la recherche.

Un attribut prédéfini signifie que:

- la classe sémantique de l'attribut est prédéfinie; ceci donne en plus un outil de modélisation et de conception. Il faut noter qu'une application peut très bien utiliser un attribut prédéfini pour représenter une propriété différente de celle prévue, comme par exemple utiliser un *mot-clé* pour contenir le nom du projet auquel appartient cet item.
- l'identificateur du type est prédéfini, i.e. un *ObjectIdentifier* lui est affecté.
- un sous-ensemble de descripteurs, en particulier, le type de valeurs et le type d'information représentée sont prédéfinis. Par exemple, l'attribut *expéditeur* est prédéfini représentant la *relation item-utilisateur*. Il est *mono-valeur*, *a-statique* et *recherchable*.

La figure 5-7 montre la structure d'un item qui se compose d'un ensemble d'attributs prédéfinis, d'un ensemble d'attributs spécifiques, et d'un contenu.

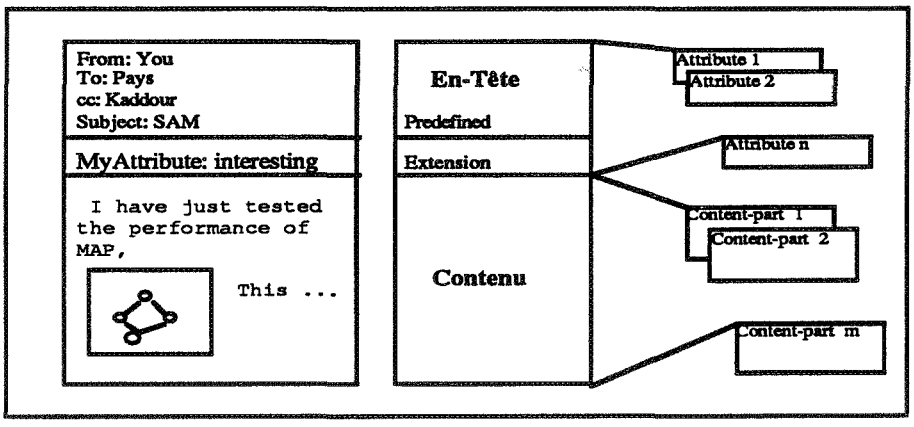


Figure 5-7. Structure d'un Item

On donne ici les classifications des attributs prédéfinis et quelques exemples:

- Attributs liés au stockage: la date d'entrée de l'item, la date de la dernière modification, la date de la dernière consultation, etc. Ces attributs sont *universels*, ils sont obligatoires pour tous les items.
- Attributs représentant les noms: *OID* et *ExternalName*.
- Attributs décrivant le contenu: *ContentOrdered* spécifie si le contenu est ordonné ou non, *ContentType* spécifie si le contenu est explicite ou implicite, etc.
- Attributs de Répartition: *IncompletLevel* donne le niveau d'incomplétude, *MayBeFoundIn* donne la (les) localisation(s) d'autres instances de l'objet, *IsMaster* spécifie si l'item est une instance *maître*, *Master* donne la localisation de l'instance maître, etc.
- Propriétés diverses: *Importance* spécifie le niveau d'importance, *Subject* est une chaîne de caractères contenant le sujet d'un message ou la description d'une collection. Les autres exemples sont *Keywords*, *Priority*, *Comments*.
- Attributs représentant des liens: *InReplyTo*, *CommentOn*, *VersionOf*, *ContinuationOf*, *CrossReference*, etc.
- Attributs représentant des relations item-utilisateur: *Originator*, *Recipients*, *Editors*, *Authors*, etc.

On peut manipuler des attributs d'un item dynamiquement: par exemple, ajouter une instance d'un attribut, supprimer une instance d'un attribut, supprimer un attribut (i.e. supprimer toutes les valeurs/instances de l'attribut), lire un ou plusieurs attributs selon leurs types ou descripteurs.

En particulier, MAP permet d'ajouter n'importe quel attribut à un item (s'il est autorisé par l'application). Par exemple, pour gérer la nouveauté pour un archiveur personnel, on pourra attacher un attribut *NiveauDeNouveauté*, qui prend une valeur parmi

- déjà lu (ou processed)
- déjà résumé (listed)
- pas encore touché

5.1.4. Contenu

Chaque item a un contenu qui représente la vraie information de l'item. Le contenu est conçu pour représenter le corps d'un objet atomique ou composite.

Chaque contenu est composé d'un **ensemble ordonné** ou **non** de parties-de-contenu (Content-Part ou CP), chacune ayant un nom unique dans l'ensemble (obligatoirement!). Un ensemble ordonné forme une séquence, et la position d'une CP dans la séquence ne peut être utilisée que pour lire cette CP (cela garantit que les opérations sont *idempotentes*). Cette propriété du contenu est indiquée par un attribut *ContentOrdered* qui prend une valeur booléenne. Le choix de cette structure est basé sur les messages utilisés dans les systèmes de messageries. Un document complexe ayant besoin d'une relation plus compliquée entre les parties de contenu peut être représenté par plusieurs items liés. De plus, un attribut utilisateur peut être attaché à l'item pour indiquer une structure de contenu autre que celles supportées par MAP.

Un contenu peut être *statique* ou *dynamique,* selon l'indication de l'attribut *ContentState.* On ne peut pas modifier un contenu statique. Remarquons qu'un contenu *ordonné* est souvent *statique*; cela permet l'optimisation dans l'implantation. Les corps de messages sont des exemples typiques.

Une CP est soit un document (texte, image, son, graphe, fax, ...), soit une **référence** à un autre item. Donc, des objets atomiques correspondent aux items dont le contenu est composé uniquement de Document-CP (DCP). Les objets composites sont des items avec une ou plusieurs Référence-CP (RCP).

Le contenu d'un item peut être **explicite** ou **implicite.** Les CP d'un contenu explicite sont associées explicitement à l'item, tandis que les CP d'un contenu implicite sont décidées dynamiquement par une *règle de dérivation.* Les CP d'un contenu implicite sont nécessairement des RCP. La règle de dérivation est un critère de sélection (i.e. un sélecteur) qui spécifie les conditions que doivent satisfaire les items référencés par les RCP. Un attribut *ContentType* avec une valeur booléenne indique le type du contenu d'un item. La règle de dérivation est contenue dans l'attribut *DerivationRule.*

Le nom d'une CP est soit une chaîne de caractères pour un DCP, soit l'OID de l'item référencé pour un RCP.

Une DCP est composée de:

- Un nom qui identifie la DCP dans le contenu.
- Un type qui spécifie le type d'information contenue dans la DCP, par exemple, texte(IA5), image, graphe, ... Les types supportés sont basés sur les types de parties de corps de messages-IP. On définit dn plus un type *brut* (*raw*) qui est simplement une chaîne d'octets.
- Paramètres qui spécifient les propriétés de présentation de l'information, par exemple, le type de codage utilisé pour l'image, ou la version de PostScript™ utilisée. Les paramètres sont traités par MAP comme une chaîne d'octets.
- Données représentant l'information en une chaîne d'octet.

Une RCP est composée de:

- Un nom: l'OID de l'item référencé; cet item est aussi appelé *composant.*
- Un type: il est du type *RCP.*
- Paramètres: données utilisateurs ou applications, par exemple, indication de la partie de contenu "intéressante" dans l'item référencé. Cette partie est vide pour une RCP d'un contenu implicite.

Les utilisateurs peuvent aussi spécifier les types de CP qu'un item peut avoir, en donnant les types autorisés comme valeurs de l'attribut *InformationAutorisée.* On peut ainsi définir des classeurs en donnant la valeur "RCP" à cet attribut.

On peut insérer, retirer d'une CP d'un contenu explicite; remplacer un contenu explicite; lire un, plusieurs ou toutes les CP du contenu d'un item, selon leurs noms, types ou positions

dans le contenu (si ordonné). En fait, la flexibilité de contenus (multi-parties, dynamiques) permet de réaliser des contextes centralisés comme dans les systèmes hypertextes.

5.1.5. Liens entre des Items

Un aspect spécifique des messages est qu'ils sont souvent liés. Les relations sont établies entre des items pendant leurs créations ou dynamiquement après leurs créations. Les liens représentent les relations entre items. Ils permettent de retrouver des items à partir des items connus. Par exemple, la relation Question-Réponse entre une question et une réponse est représentée comme un lien entre la réponse et la question. Elle permet de retrouver la question à partir de la réponse, et vice versa.

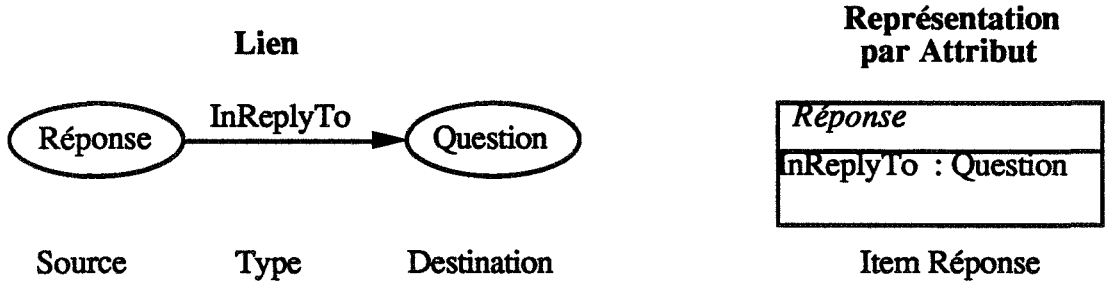


Figure 5-8. Lien

Un lien est **typé** et **orienté**. Il comprend un *type*, un item *source* et un item *destination* (la figure 5-8). Sa direction est de l'item *source* vers l'item *destination*. Le lien est **sortant** de l'item *source* et **entrant** de l'item *destination*. Le lien Question-Réponse est du type *InReplyTo* de la réponse vers la question. Un lien est représenté comme une instance d'un attribut (avec le descripteur *InformationReprésentée* comme *Link*) de l'item source. Le type de l'attribut est celui du lien et la valeur est une *référence* à l'item destination (du type *ObjectDescriptor*). Le fait de représenter un lien comme un attribut de l'item source permet de donner la responsabilité du lien à l'item source. Un utilisateur peut ajouter un lien ou détruire un lien en manipulant simplement l'item source sans tenir compte de l'item destination. Ceci est dû au fait que la création des messages et donc des liens ne peut pas être totalement contrôlée par l'archiveur de messages dans un système de messagerie distribuée. Par exemple, dans la conférence MMM, un auteur du site INRIA crée un message dans le sous-thème "archivage" et lui attache un attribut "CommentOn" faisant référence à un autre message dans le sous-thème "image". Ce message est ensuite distribué au site EMSE qui n'ait pas abonné au sous-thème "image". Cependant le lien existe comme attribut du message dans le site EMSE même si l'item destination n'existe pas dans ce site.

Le nombre et le type des liens entre deux items ne sont pas limités dans MAP. De plus, tous les liens sont utilisables pour la recherche.

Le type **membership** est un type particulier de lien. Il est la relation entre un objet composite et un composant de cet objet (la direction est de l'objet composite vers son composant). Un lien de ce type est représenté, contrairement à tous les autres, comme une RCP de l'objet composite. Tous les items d'une base MAP forment une **hiérarchie** avec les liens *membership*. Parcourir cette hiérarchie permet de suivre l'organisation des items. Un exemple de cette hiérarchie est montré dans la figure 5-9. Une utilisation possible est de représenter un arbre de *folders*.

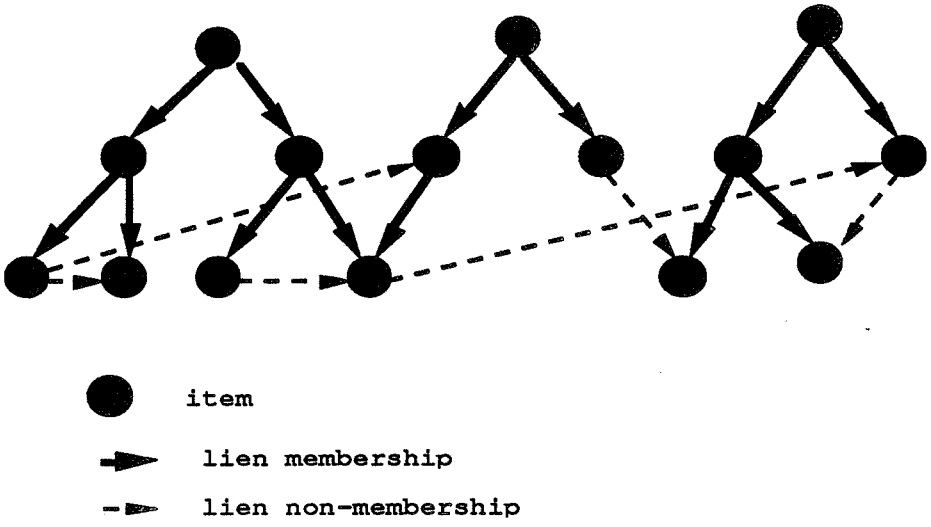


Figure 5-9. Un exemple de la hiérarchie d'items

Un *ObjectDescriptor* est composé du nom de l'item (interne et/ou externe) avec une petit complément d'information. Cette dernière indique les parties exactes des items qui se sont référencées. Il peut également contenir la nature du lien (cf. [Brun 87]). Une autre utilisation de cecomplément d'information attachée est de contenir une explication/description du lien comme dans un système hypertexte (cf.[Conklin 87], [Campbell 88]).

Comme un attribut, on peut insérer, retirer ou supprimer un lien (s'il n'est pas statique). Cependant, l'importance des liens exige des supports supplémentaires. MAP permet d'obtenir facilement tous les liens entrants d'un item aussi bien que les liens sortants de l'item. On peut suivre un lien **récursivement**, par exemple, obtenir tous les items commentant *directement* ou *indirectement* un item donné.Une autre facilité fournie par MAP est d'obtenir les liens entre deux ensembles d'items (ces deux peuvent être identiques), les types des liens sont spécifiés par l'utilisateur. Ainsi on peut obtenir un sous-graphe avec les liens "InReplyTo". Il est possible de ne garder que des liens sortants d'un item sans autres parties de l'item (voir la section objet incomplet). Ces fonctionnalités sont fournies aussi bien pour des liens spécifiques que pour les liens prédéfinis.

Ces facilités sont très intéressantes pour réaliser les fonctions évoluées. Par exemple, un lien *VersionOf* est défini pour faciliter la gestion de versions par les FE. Les versions d'un même objet peuvent former une relation linéaire, ou un arbre, selon l'application. Le sous-graphe des versions peut être obtenu très facilement et efficacement, soit en suivant récursivement le lien VersionOf soit en demandant les liens du type *VersionOf* entre deux ensembles d'items.

Le lecteur familier avec les systèmes *hypertextes* peut remarquer que notre système peut être utilisé comme la partie stockage d'un système hypertexte. Il fournit plus de fonctionnalités de manipulations de liens que les systèmes hypertextes généraux; en particulier, MAP permet la récursivité dans la manipulation de liens. D'un autre coté, cela nous donne sans doute la possibilité de réaliser notre système en utilisant un système hypertexte.

5.1.6. Relations entre Items et Utilisateurs

Un autre aspect spécifique est l'existence de nombreuses liaisons entre des objets d'information et des utilisateurs. Deux sortes de liaisons existent entre les objets d'information et les utilisateurs:

- Les liaisons sémantiques: expéditeur, auteur, etc.
- Les liaisons d'Accès: les droits d'accès relient les utilisateurs et les objets d'information.

Dans cette section, nous nous intéressons à la première catégorie de liaisons.

Les relations item-utilisateur sont typées. Elles sont représentées comme des attributs dont les types sont les types des relations. Le type de valeurs d'un tel attribut est *UserDescriptor*, qui est composé de l'identificateur de l'utilisateur et d'un complément d'information, indiquant, par exemple, si une réponse est demandée par l'utilisateur quand le type de relation est *destinataire* (Figure 5-10).

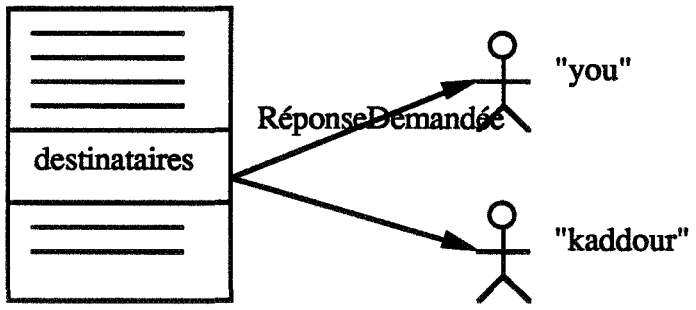


Figure 5-10

MAP supporte tous les types de relations dont un sous-ensemble est prédéfini. A part les manipulations des attributs, MAP fournit des supports spéciaux. Par exemple, on peut connaître tous les items concernant un utilisateur donné (i.e. tous les items qui ont des relations avec l'utilisateur).

On pourra utiliser des relations item-utilisateur pour gérer la nouveauté dans un archiveur multi-utilisateurs. Pour cela, deux attributs locaux sont définis:

- LuPar: il prend les utilisateurs qui ont déjà lu cet item comme valeurs.
- RésuméPar: les utilisateurs qui ont déjà obtenu un résumé de l'item.

Ainsi pour connaître tous les nouveaux items, il suffit de faire la recherche d'après ces deux relations.

5.1.7. Objets Incomplets

Dans une base d'information, on a souvent besoin de stocker et gérer des items *incomplets* qui contiennent seulement une partie des objets qu'ils représentent. Cela est nécessaire pour les raisons suivantes:

- Des objets référencés peuvent être **absents** parce qu'on ne connaît que leurs noms. Dans un environnement de messagerie, la communication n'est pas garantie, i.e. les messages peuvent arriver dans l'ordre inverse de celui de l'envoi; il est même possible de perdre des messages. Dans ce cas, il est possible qu'un commentaire soit arrivé avant le message principal dont le nom externe est connu, grâce au lien contenu dans le commentaire. C'est à dire que seuls les noms de l'item sont connus.

- Raison de la Cohérence: le contenu d'un item peut être périmé, par exemple par dépassement de la date de péremption. Mais il est parfois nécessaire de continuer à garder les liens sortants de l'item qui peuvent être utilisés pour relier les autres items. Par exemple, dans une chaîne de versions linéaire, on doit garder tous les liens *VersionOf* afin de ne pas perdre les liaisons entre les différentes versions discontinues.

- Raison Economique: un *petit* archiveur qui n'a pas beaucoup de ressources (disque, par exemple) ne garde que les petits items fréquemment utilisés. Seules les références des autres items (*gros* ou *inutilisés*) sont gardées, et ces items sont cherchés automatiquement dans un archiveur de référence qui se situe dans une machine plus puissante si nécessaire. Cela garantit l'efficacité en faisant la plupart des accès locaux dans le *petit* archiveur. C'est typiquement la situation de micro-ordinateurs.

- Raison de la Sécurité: Les items incomplets permettent de ne distribuer qu'une partie non-confidentielle d'un objet, par exemple, la partie *résumé* d'un item seul. Pour le reste un droit supplémentaire est demandé.

On définit un **objet incomplet** comme une instance partielle de l'objet dont seule une partie (les noms, et/ou attributs, et/ou certaines parties de contenus) est présente. En théorie, n'importe quelle partie peut être absente, cependant pour les utilisations et manipulations efficaces, on définit plusieurs **niveaux d'incomplétude** d'items.

- Niveau **1**: Un item de ce niveau contient seulement ses identificateurs (i.e. les noms interne et/ou externe) et l'attribut qui indique son niveau d'incomplétude. Les items de ce niveau sont souvent appelés les *item noms*, et sont utilisés comme des items fictifs quand les vrais items référencés sont absents.

- Niveau **2** : Un item du niveau 2 comprend les attributs de distribution en plus de ses noms. L'item de ce niveau permet de localiser une instance plus complète.

- Niveau **3.1**: Un item du niveau 3.1 contient un en-tête complet, mais sans contenu. Ce niveau permet de garder les en-têtes contenant des informations structurées qui sont généralement assez petits en taille.

- Niveau **3.2** : Un item du niveau 3.2 contient tous les attributs *liens* et les *RCP*, en plus de l'information contenue dans un item du niveau 2, i.e. un item de ce niveau garde tous les liens.

- Niveau **4** : Un item du niveau 4 contient toutes les informations d'un item du niveau 3.1, plus toutes les RCP et une *DESCRITPTION* {Type, Nom, Longueur} pour chaque DCP. En général, un item de ce niveau est utilisé pour garder les informations structurées (l'en-tête) et la structure du contenu sans la donnée elle-même.

- Niveau **5** : Un item du niveau 5 comprend toutes les informations du niveau 4 plus certaines DCP complètes, i.e. il manque seulement quelques DCP complètes. On peut utiliser des items de ce niveau pour inclure toutes les informations sauf quelques parties de contenu de taille importante, ou les parties de contenu illisibles pour raison matérielle (par exemple, l'image et le son nécessitent des équipements spéciaux).

Pour unifier le langage, nous ajoutons deux niveaux:

• Niveau **0**: l'item est complètement absent. Tous les items *inconnus* sont à ce niveau.

• Niveau **6**: c'est le niveau *complet*, un item (plus exactement, une instance) de ce niveau est complet.

Les niveaux forment un ordre partiel (0 < 1 < 2 < (3.1, 3.2) < 4 < 5 < 6), montré dans la figure 5-11.

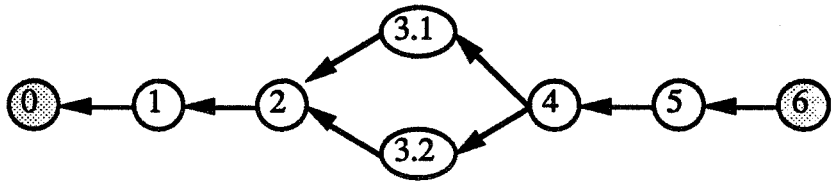


Figure 5-11. L'ordre des Niveaux

Chaque item dans une base a un attribut mono-valeur **niveau** (*IncompletLevel*) qui indique le niveau d'incomplétude de l'item. Le niveau d'incomplétude d'un item spécifie aussi les opérations possibles sur cet item. Pour chaque item, on peut

- diminuer ou augmenter son niveau.
- manipuler la partie présente d'après son niveau d'incomplétude, la partie absente n'est pas manipulable. Par exemple, on peut ajouter une DCP complète dans un item du niveau 5, mais pas dans celui du niveau 4.

On doit remarquer que les objets incomplets sont très utiles dans un environnement avec des archiveurs distribués, car ils permettent des stratégies de distribution et de répartition souples et efficaces.

5.1.8. Représentabilité d'items

Dans cette section, nous allons donner quelques exemples pour montrer comment les objets d'applications peuvent être représentés par les items.

Représenter des objets réels en *items* est très simple. Chaque objet atomique est représenté comme un item sans *RCP*. Chaque objet composite est représenté par plusieurs items, représentant chacun un objet (composite ou composant). Ces items sont liés par des liens *membership* représentés comme *RCP*. Les attributs sont représentées comme l'en-tête de l'item. Les différentes caractéristiques des objets sont représentés par les attributs qui possèdent des descripteurs appropriés. Par exemple, l'attribut *Recherchable* indique si l'objet représenté peut être utilisé pour la recherche; l'attribut *Niveau* indique le niveau d'incomplétude qui sera différent de **6** lorsque l'objet est incomplet; l'attribut *global* indique si l'objet est global ou local; etc. Voici quelques exemples:

• Messages Simples: On représente chaque message simple comme un item, dont les attributs originels sont *statiques* et *originels*. Le corps devient le contenu de l'item dont chaque partie de corps correspond à une DCP, et le contenu ainsi obtenu est *ordonné* et *statique*. Les attributs additionnels sont représentés comme des attributs *dynamiques* et *additionnels* de l'item.

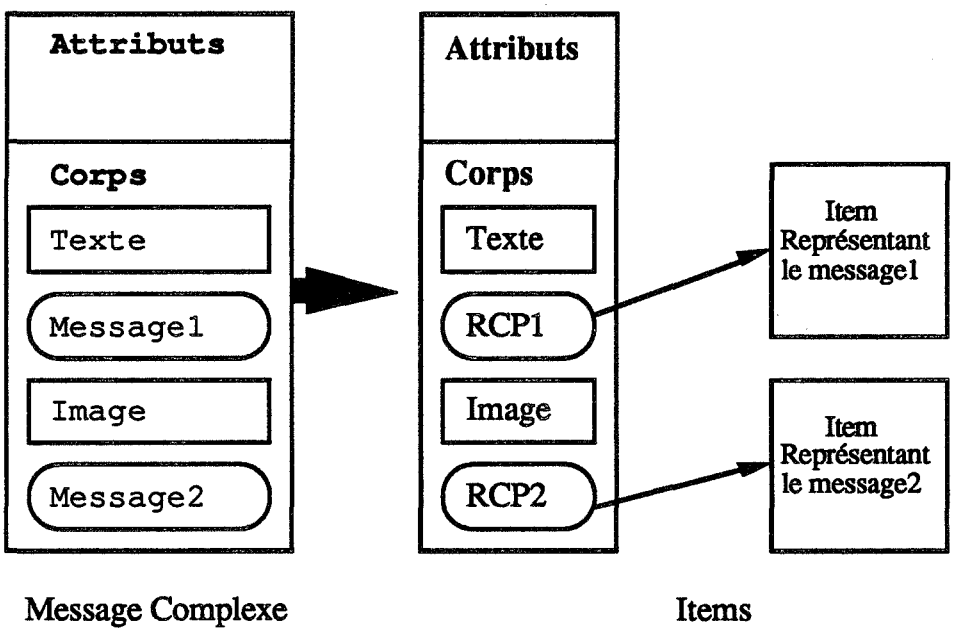


Figure 5-12. Représentation d'un message complexe en items

• Messages Complexes: un message qui contient d'autres messages dans son corps est représenté par plusieurs items, dont un item principal représente le message lui-même et les autres représentent les messages emboîtés. Le contenu de l'item principal est ordonné selon le même ordre que l'originel. Il contient toutes les parties non-messages du corps du message comme *DCP*. Les messages emboîtés sont remplacés par les RCP contenant les références aux items représentant ces messages. (cf. figure 5-12)

• Collections: Une collection explicite est représentée comme un item dont le contenu est explicite et contient uniquement des RCP non-ordonnées. Chaque RCP fait référence à un item qui représente un membre de la collection. Les attributs de la collection sont représentés comme des attributs de l'item, et sont souvent *dynamiques*. Une collection implicite est représentée comme un item implicite dont le *DerivationRule* est un sélecteur équivalent à la règle de dérivation de la collection.

• Post-it-notes: Chaque post-it-note est représenté par un item lié à l'item qui représente le message auquel le post-it-note est collé. Les attributs du post-it-note sont représentés comme des attributs de l'item, et sont souvent dynamiques. Le contenu est simplement un bout de texte contenant les notes. L'item contient un lien soit *RelatedTo* soit *AppliedTo* pointant sur l'item représentant le message. Il est aussi possible de lier le même post-it-note à plusieurs messages (i.e. items) ou même à une collection (une note sur l'ensemble de messages dans la collection).

5.1.9. Problèmes Spécifiques aux Environnements Messageries

Le système de messagerie pose des problèmes bien spécifiques à cause du fait que les messages peuvent être perdus, leur ordre d'arrivée inversée, les délais de communication non garantis, et la distribution des informations incomplète. Cela implique qu'en théorie, il n'est même pas possible de savoir si un message est perdu ou simplement retardé. Dans cette section, nous étudions des problèmes spécifiques ainsi posés à l'archiveur de messages et les solutions prises par MAP.

A cause du délai, et de la non-fiabilité de la communication, ou de la distribution partielle des informations, un item peut être référencé par un autre item en étant absent dans la base. Par exemple, une réponse est arrivée, mais la question est perdue, retardée ou non envoyée à ce même destinataire. Plusieurs solutions sont possibles:

(1) *Cacher l'item* qui fait référence à un item absent en attendant que l'item référencé arrive. Cette solution échoue si l'item référencé n'arrive jamais à cause de la perte ou tout simplement du non envoi vers cet archiveur.

(2) *Créer un objet incomplet de niveau 1*. Il résout non seulement le problème de la visibilité mais aussi permet de *mémoriser* des opérations sur l'item (voir le mécanisme d'historique). L'item incomplet ainsi créé peut ensuite être remplacé par une instance complète. C'est cette solution qui a été choisie. Il n'empêche qu'un FE peut utiliser la première solution s'il est sûr que les conditions extrêmes ne se produisent pas.

En fait, on peut considérer qu'une fois qu'un item est créé dans la base, il existera toujours, ne sera jamais détruit et il n'y aura jamais un autre item qui possèdera le même nom. Cependant un item peut être effacé jusqu'à ce qu'il ne reste plus que son(ses) nom(s) dans la base. C'est lors de l'implantation que l'on décide si l'item est gardé physiquement dans la base, quand il n'est plus référencé par aucun autre item de la base. Ainsi nous obtenons un modèle cohérent.

Il est possible que le même message arrive plusieurs fois. Par exemple, lorsque le destinataire est sur deux différentes listes de distribution, ou lorsqu'un utilisateur reçoit directement le message qui est aussi retransmis par un ami. Ces deux copies peuvent être *légèrement* différentes, par exemple, l'utilisateur reçoit deux copies du même message comme *destinataire principal* et *destinataire en copie muette* respectivement. Dans les deux cas, les attributs "destinataires" (en copie, en copie muette, etc) sont différents (cf. [X.411 84]). MAP ne garde qu'une copie pour chaque item/message (il considère que les deux items sont équivalents s'ils ont le même nom externe). C'est au FE de choisir la copie préférée. La même solution est adoptée quand les deux copies diffèrent dans le codage utilisé pour leurs contenus (cf. [X.400]) à cause de la conversion automatique.

En fait, MAP ne peut décider d'une stratégie unique pour des situations différentes dans différentes applications MHS et GC. C'est donc le rôle des FE de décider et de mettre en oeuvre les stratégies adaptées à l'application.

5.1.10. Manipulation d'Items

Avant de présenter les opérations fournies pour manipuler la base d'information et les items, il nous semble important de rappeler nos principes de conception afin de pouvoir mieux expliquer notre choix:

- Simplicité: seules les primitives sont fournies pour modifier des items.
- Efficacité: certaines opérations de *consultation* sont fournies, même si elles peuvent être réalisées par les primitives.
- Puissance: les opérations sont complètes, i.e. les combinaisons de ces opérations peuvent réaliser toutes les manipulations envisageables.

Les opérations fournies par MAP ont les propriétés suivantes:

- Chaque opération modifie **au plus un** item. On peut voir MAP comme un système orienté-objet qui a une classe d'objets: *items*, avec un ensemble de méthodes qui sont les primitives de MAP manipulant des items individuels.
- toutes les opérations sont **idempotentes**, i.e. les exécutions successives de la même opération ont le même effet qu'une seule exécution sur la base. Cette propriété nous donne la possibilité d'exécuter la même opération plusieurs fois sans risque. Ceci est utile quand on n'est pas sûr qu'une opération a été exécutée à cause du délai et de la non-fiabilité de la communication. Dans un environnement distribué, il est particulièrement intéressant qu'une modification puisse être distribuée ou propagée plusieurs fois dans un même site.
- Pour chaque opération de modification, il existe toujours une opération inverse dont l'exécution annule ("**undo**") l'effet de l'opération. On pourra obtenir l'opération inverse par le mécanisme d'historique. La possibilité d'*annuler* n'importe quelle opération est très souhaitable dans un système informatique.

Il faut noter que dans un archiveur de messages, les modifications sont beaucoup moins fréquentes que les consultations; de plus, la plupart des modifications sont des ajouts de messages, des effacements de messages et des ajouts de liens. Cette propriété est prise en compte par MAP.

5.1.10.1. Liste d'Opérations

Manipulations d'un item individuel:

(1) **Stocker**: Stocker ou remplacer un item; l'item stocké peut être complet ou non.
Quand l'item à stocker existe(i.e. l'item avec le même nom existe), une *option* permet d'indiquer s'il doit être remplacé. Si oui le niveau maximum de l'item à remplacer doit être donné, l'item sera remplacé lorsque son niveau est inférieur ou égal à ce niveau donné. Donc cette opération permet d'augmenter le niveau d'un item.

(2) **Effacer**: Effacer un item à un niveau donné. Comme indiqué avant, on ne peut détruire complètement un item de la base (les noms au moins sont gardés logiquement). L'effacement au niveau **0** a une signification spécifique qui exclut un item d'un contexte. (Voir la section correspondante pour plus de détails)

Manipulations d'un attribut d'un item:

(3) **V_Ajouter**: ajouter une ou plusieurs valeurs/instances d'un attribut. Si l'attribut n'existe pas dans l'item, il sera créé automatiquement. Pour un attribut mono-valeur, la valeur précédente (si elle existe) sera remplacée.

(4) **V_Retirer**: retirer une, plusieurs ou toutes les valeurs d'un attribut.
Remarque: Puisque les liens sont des attributs, ces deux opérations permettent d'ajouter et supprimer un ou plusieurs liens.

Manipulations du contenu ou d'une partie du contenu d'un item:

(5) **CP_Ajouter**: ajouter une partie de contenu. La position de la CP à ajouter doit aussi être spécifiée si le contenu est ordonné. Si une CP avec le même nom existe, une erreur est produite.

(6) **CP_Retirer**: supprimer une partie de contenu. La CP à supprimer est identifiée par son nom.

(7) **C_Remplacer**: remplacer le contenu par un nouveau.

Consultation et Recherche:

(8) **Rechercher**: rechercher un ensemble d'items selon un critère de choix que les items doivent satisfaire (voir §9 pour une description détaillée). Le résultat est une liste d'OID des items trouvés.

(9) **Résumer**: obtenir les sommaires pour un ensemble d'items satisfaisant un critère. Le sommaire d'un item contient un sous-ensemble d'attributs de l'item indiqué comme paramètre de l'opération (par exemple, le niveau d'incomplétude, le sujet, la date de réception, l'importance, etc) plus une *DESCRIPTION* du contenu sous forme {nom, type, longueur} pour chaque CP.

(10) **Lire**: lire un objet ou une partie, et/ou les liens entrants. La partie peut être un sous-ensemble d'attributs sélectionnés d'après leurs types ou descripteurs, et/ou une ou plusieurs parties de contenu identifiées par leurs noms ou leurs types. En particulier, on pourra ne lire que les identificateurs de l'objet qui nous permet de faire la correspondance entre un identificateur interne et le nom externe, et réciproquement. On peut obtenir tous les liens d'un item, ou tous les attributs originels d'un item. On peut aussi ne lire que les parties de contenu *textes* si l'on ne dispose que d'un terminal alphanumérique. La possibilité de lire des liens entrants est très importante: elle nous permet de suivre, en sens inverse, les liens, c'est à dire d'obtenir les réponses à partir d'une question en suivant les liens *InReplyTo* en sens inverse.

(11) **L_Get**: Obtenir des liens sélectionnés (i.e. qui ont des types spécifiés) entre deux ensembles d'items. Cette opération nous permet d'extraire un sous-graphe. Par exemple, on peut obtenir rapidement le graphe des messages dans le sous-thème "archivage" avec les liens "InReplyTo". Cela nous permet de visualiser les questions et leurs réponses par une interface graphique.

(12) **U_Get**: Obtenir les relations entre un ensemble d'items et un utilisateur. Ceci permet à un utilisateur de connaître ses relations avec un ensemble d'items, s'il est *auteur, éditeur, expéditeur, destinataire,* etc.

Remarque: Il n'y a pas d'opération de **destruction** explicite d'un item. Chaque item est considéré comme éternel dans la base, une fois qu'il est stocké. Cependant, l'item peut se réduire au niveau **1** (seuls les noms sont gardés). MAP peut aussi physiquement détruire ces noms quand ils ne sont plus utiles, i.e. lorsque l'item du niveau **1** n'est plus référencé par aucun autre item.

5.2. Classes d'Items

Bien que les items aient tous la même structure, selon les objets réels qu'ils représentent, ils sont divisés en **classes** ([Malone 87], [Lai 88]). Les items d'une classe sont souvent appelés **instances** de la classe. Ils obéissent à la même règle vis-à-vis de leur structure et des informations qu'ils contiennent, et reçoivent un même groupe d'opérations. Les exemples de classes d'items sont *collection* dont les instances sont des collections d'items, *messages* dont les instances représentent les messages.

Une classe regroupe un ensemble d'items. Elle spécifie les caractéristiques ou les contraintes suivantes pour les instances de la classe:

- Un ensemble d'attributs qui sont *obligatoires* pour les instances de la classe. Par exemple, les instances de la classe *message* ont obligatoirement un attribut *destinataires*.
- Pour un sous-ensemble d'attributs définis dans la base, *raffiner* les caractéristiques de chaque attribut, à savoir:
  - La condition que la (les) instance(s) de l'attribut doit (doivent) satisfaire, par exemple, l'*importance* doit être *high*, ou un *mot-clé* doit être *Unix*.
  - les nouvelles valeurs par défaut des descripteurs si besoin. Ces descripteurs sont
    - Global/Local
    - Originel/Additionnel
    - I-Statique/Dynamique
  - La (les) instances(s) par défaut de l'attribut.
- Un ensemble d'actions associées à la classe spécifique aux instances de la classe. Cependant, MAP les considère comme des chaînes d'octets, i.e. il ne fournit qu'un moyen de stockage. C'est aux FE de les utiliser.

Remarque: puisque les caractéristiques du *contenu* sont représentées par les attributs, les conditions sur les attributs spécifient les caractéristiques du contenu. Par exemple, le contenu d'une *collection* est ContentState=Statique ContentOrdered=Non-Ordonné InformationAutorisée=RCP.

Un item d'une classe doit avoir tous les attributs obligatoires attachés, et les attributs raffinés doivent satisfaire les conditions imposées. Les valeurs par défaut de descripteurs et d'attributs sont utilisées par MAP pendant la création/stockage de l'item dans MAP.

5.2.1. Définition de classes

Cela implique que la définition d'une classe se compose des éléments suivants:

- Identificateur de la classe, il est un *ObjectIdentifier*.
- Déclaration des attributs *obligatoires*.
- Déclaration des raffinements pour un ensemble de zéro, un ou plusieurs attributs, obligatoires ou optionnels. Pour chaque attribut dans l'ensemble il faut spécifier:
  - le type de l'attribut
  - les valeurs par défaut d'un ou plusieurs descripteurs.
  - les condition que les valeurs de l'attribut doivent satisfaire.
  - les valeurs par défaut de l'attribut.

  Ces trois derniers champs sont tous optionnels, mais au moins un parmi eux doit être présent.
- Actions associées. Chaque action se compose d'un type et d'une valeur. Le type est un OBJECT IDENTIFIER. La valeur est une chaîne d'octets.

5.2.2. Sous-Classe

Un seul type de relations est supporté entre les classes, i.e. *SousClasseDe*. Une classe est une sous-classe d'une autre classe, si :

- elle déclare un sur-ensemble des attributs obligatoires, **OU**
- elle impose une condition plus stricte au moins sur un attribut.

La caractéristique essentielle de la relation est que toutes les instances de la sous-classe sont aussi les instances *(indirectes)* de la super-classe. On remarque que la sous-classe *hérite* de tous les attributs obligatoires de sa super-classe, ainsi que des actions.

Toutes les classes dans MAP forment une **hiérarchie** avec une source commune. Chaque classe peut avoir une ou plusieurs super-classes, sauf la **racine** de la hiérarchie qui n'a pas de super-classe. La racine (la classe *items*) est prédéfinie par MAP qui déclare tous les attributs *universels* de MAP comme *obligatoires*. Les **héritages** entre les sous-/super-classes suivent les règles suivantes:

- Une sous-classe hérite de tous les attributs *obligatoires* de toutes ses super-classes.
- La condition d'un attribut est la combinaison **ET** de toutes les conditions imposées par les super-classes **ET** celle de cette classe.
- MAP ne gère pas d'héritage de valeurs par défaut ni d'actions. Cependant il permet, pour un type donné, d'obtenir toutes les super-classes qui déclarent une action de ce type. C'est l'application (les FE) qui choisit l'action appropriée, i.e. qui gère l'héritage multiple d'actions.

On prédéfinit trois autres classes dans MAP (figure 5-13), elles sont:

- Items: c'est la racine de l'hiérarchie, tous les items sont instances (directes ou indirectes) de cette classe.
- Collections: les instances de cette classe sont des collections.
- IPMessages: les instances de cette classe sont des IPMessages, les attributs obligatoires des IPMessages sont définis, par exemple, destinataires, etc.

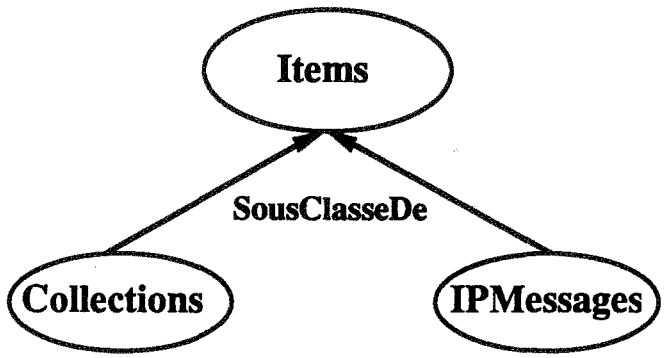


Figure 5-13. Les Classes Prédéfinis

En résumé la définition d'une classe provient des éléments hérités des super-classes et des nouveautés apportées par:

- les nouveaux attributs *obligatoires*.

- les nouvelles valeurs par défaut pour certains attributs et/ou les descripteurs par défaut.
- les nouvelles conditions sur certains attributs.
- les nouvelles actions.

Avec la déclaration de super-classes, ces éléments constituent la définition d'une classe.

Chaque item ne peut être instance directe d'une seule classe, il est cependant instance indirecte de toutes les super-classes de la classe. La classe auquelle un item appartient est indiquée par attribut *ObjectType* qui est *obligatoire* pour tous les items. Cet attribut peut être dynamique, on peut "déplacer" un item d'une classe à une autre, à la condition qu'il respecte la définition de la dernière classe. Pendant la recherche, tous les items directs ou indirects de la classe spécifiée dans le sélecteur sont vérifiés.

5.2.3. Manipulation de Classes

On peut définir une nouvelle classe, supprimer une classe si elle n'a pas d'instance. Cependant il n'est pas possible de modifier la définition d'une classe sauf les valeurs par défaut et les descripteurs par défaut. Mais on peut changer la classe d'un item. Donc il suffit de définir une nouvelle classe et de remettre les instances de la classe existante dans la nouvelle (en modifiant leurs attributs *ObjectType*), pour modifier la définition d'une classe existante.

If faut noter que le concept de *classe* peut être réalisé par les FE de l'application. C'est la solution prise dans POMS (cf. Chapitre 8). POMS a été conçu avant l'introduction de la notion de classe dans MAP. En effet, nos expériences acquises durant la conception de POMS nous ont conduit à intégrer les classes dans MAP.

6. Modèle d'Utilisateurs

Un autre type d'objets est utilisateur. Bien que les utilisateurs soient gérés par le système de directory, i.e. stockés dans le Directory, MAP doit avoir son propre modèle utilisateur cohérent avec ceux d'utilisés par les applications MHS et GC.

Un utilisateur *interroge* une base de messages soit par accès aux objets d'information (lire, stocker, ...), soit par des relations sémantiques avec les objets d'information (auteur, expéditeur, destinataire, éditeur, ...). Une identification est nécessaire dans les deux cas. En plus dans le premier cas, les autres attributs de l'utilisateur doivent être considérés pour contrôler les accès.

Logiquement, la gestion d'utilisateurs et de leurs attributs est sous la responsabilité du DS. Nous définissons seulement ici notre modèle d'utilisateurs qui sera géré par le DS global (si disponible) ou par un DS local en coopérant avec les FE.

Dans une application MHS et GC, un *groupe* d'utilisateurs coopèrent dans le cadre d'une *activité*. Les fonctionnalités des utilisateurs sont groupées par les *rôles*. A chaque instant donné, chaque utilisateur joue zéro, un, ou plusieurs rôles dans chaque activité auquelle il participe. Par exemple, les utilisateurs d'une conférence d'ordinateurs ont les rôles suivants (cf. [Brun 87]):

- *Administrateur* qui s'occupe de la gestion globale de la conférence.
- *Editeur* qui accepte des contributions des auteurs et les ajoute dans la conférence.
- *Ecrivain* qui ajoute sa propre contribution dans la conférence.
- *Auteur* qui a le droit de créer une contribution, mais doit la passer à un éditeur afin de l'ajouter dans la conférence.
- *Lecteur* qui a le droit de lire les contributions dans la conférence.

En plus, les utilisateurs sont souvent organisés en groupes et rôles dans un environnement bureautique, par exemple, *directeur*, *sous-directeur*, *secrétaire*, *employés*, etc. Le rôle joué par un utilisateur décide de sa capacité d'échange d'informations (cf. [Tsichritzis 86]).

Donc MAP considère que chaque utilisateur a un identificateur unique, il participe à une ou plusieurs activités (dans un ou plusieurs groupes) et joue zéro, un ou plusieurs rôles dans chaque groupe, à chaque moment donné. C'est à dire qu'un attribut *rôle* est associé à un utilisateur pour chaque *activité* auquelle il participe.

Quand un utilisateur accède MAP, il le fait dans le cadre d'une activité et joue un rôle précis dans cette activité (cf. [Danielson 86]). Donc, un utilisateur accédant à la base se présente comme un triplet <nom, activité, rôle> qui est utilisé pour spécifier ses relations avec les items et contrôler son accès.

Remarque: ce modèle peut être utilisé pour simuler le modèle users-groups du système UNIX. Dans ce cas, l'activité est *AccèsUnix*, les rôles correspondent aux groupes d'UNIX.

Les structures de données d'Identificateurs d'Utilisateur (**UID**), d'Identificateurs de Rôle (**RID**) et d'Identificateurs de l'Activité (**AID**) sont spécifiées par MAP, leurs affectations sont à la charge des FE de l'application.

Il faut noter qu'il peut aussi avoir des relations entre groupes et rôles, par exemple, un groupe peut être sous-groupe d'un autre, un rôle peut être *sous-ordonné* d'un autre rôle (directeur-secrétaire), etc. Leurs spécifications ne sont pas claires dans des applications MHS et GC à cause de la complexité. Voir [You 87b] pour des utilisations dans un environnement bureautique pour contrôler les accès aux documents.

7. Contrôle d'Accès

L'importance du contrôle d'accès dans un environnement multi-utilisateurs est évidente, en fait, les différences entre les utilisateurs résident sur leurs droits d'accès à la base d'information.

La tâche du contrôle d'accès dans un archiveur consiste à:

- Fournir un mécanisme aux administrateurs ou utilisateurs privilégiés de spécifier les droits d'accès des utilisateurs sur la base d'information.
- Authentifier les utilisateurs.
- Vérifier le droit d'accès d'un utilisateur aux informations qu'il veut accéder en utilisant la spécification faite par les administrateurs.

7.1. Principe

Puisque MAP n'a pas de connaissance spécifique de l'application, le contrôle d'accès est effectué par les **FE** et **MAP** en deux niveaux.

- Les FE s'occupent de l'authentification des utilisateurs, et utilisent le mécanisme du contrôle d'accès fourni par MAP. (**Nota**: il peut également passer à coté du mécanisme de MAP)
- MAP fournit un mécanisme de base pour les FE, il permet également aux FE de stocker leurs informations de contrôle d'accès avec objets d'information comme leurs attributs.

7.2. Mécanisme de Base de MAP

En général, deux méthodes sont possibles pour spécifier les relations d'accès entre les utilisateurs et les objets d'information:

- Orienté-Utilisateur: pour chaque utilisateur, on spécifie les objets d'information qui peuvent accéder (et/ou qui ne peuvent pas) et ses droits d'accès sur les objets. Cette méthode n'est pas adaptée à MAP puisque les utilisateurs ne sont pas gérés par MAP.
- Orienté-Objet d'information: pour chaque objet d'information, on spécifie les utilisateurs et leurs droits d'accès. C'est cette approche que MAP utilise.

Le mécanisme de MAP consiste d'abord à introduire un ensemble de **Contrôleur d'Accès** qui spécifie les droits d'accès aux utilisateurs, ensuite associer à chaque item un **Contrôleur d'Accès**. L'association des contrôleurs et des items sont faites via les contextes, que nous allons présenter dans la section suivante. Chaque fois qu'un utilisateur accède à la base, soit le contrôleur associé par le contexte est utilisé (**Nota**: chaque opération doit être effectuée dans un contexte), soit le FE spécifie un contrôleur à utiliser.

Un contrôleur d'accès spécifie les droits des utilisateurs, i.e. il se compose de paires <utilisateurs, droits d'accès>. Un contrôleur doit pouvoir:

- Contrôler les accès pour un item individuel.
- Contrôle les accès pour un ensemble d'items de la même manière pour tous les items de l'ensemble.

Dans ce but, un contrôleur se compose d'un identificateur (ACID pour Access Controler IDentifier) et d'un ACL (pour Access Control List). L'ACID identifie le contrôleur dans l'ensemble de contrôleurs de MAP et l'ACL spécifie les droits d'accès des utilisateurs. Associer l'ACID à un ensemble d'items permet de les contrôler de la même façon, et en même temps de pouvoir modifier l'ACL. Pour plus de clarté, un ACID est un numéro de séquence donné par MAP. MAP gère un ensemble de contrôleurs (ou une base de contrôleurs).

Les utilisateurs privilégiés peuvent créer un Contrôleur, obtenir des informations sur un Contrôler, modifier/changer l'ACL associé à un Contrôleur particulier, ou supprimer un Contrôleur.

7.3. ACL

Un ACL comprend une séquence de paires
<DroitsD'Accès, DescripteurD'EnsembleD'Utilisateurs>
dont le *DroitsD'Accès* spécifie un ensemble de droits d'accès, le *DescripteurD'EnsembleD'Utilisateurs* identifie un ensemble d'utilisateurs qui ont les droits ou qui sont exclus de ces droits. On peut savoir pour chaque utilisateur identifié par <UID AID RID> s'il appartient à l'ensemble identifié par un DescripteurD'EnsembleD'Utilisateurs.

Pour chaque opération qui a besoin d'être contrôlée par MAP, MAP examine les droits de l'utilisateur qui effectue l'opération, auprès d'un ACL qui dépend du contexte où l'opération est effectuée. Les paires dans l'ACL sont evoluées une par une jusqu'à ce que les droits soient accordés ou exclus. Si les droits ne sont ni accordés ni exclus, ils sont considérés comme exclus.

7.3.1. Droits d'Accès

Comme son nom l'indique, un droit d'accès spécifie le droit d'accéder à un objet, le droit peut être "lire", "rechercher", "stocker", "effacer", etc. Cependant MAP ne spécifie pas la sémantique d'un droit d'accès, c'est à dire que MAP ne fait pas la correspondance entre une opération et le(s) droit(s) d'accès nécessaire(s). C'est la responsabilité des FE de les mettre en correspondance.

Donc, quand un FE passe une opération à MAP, il spécifie l'utilisateur qui demande cette exécution, si MAP doit contrôler l'accès, si oui, le(s) droit(s) d'accès que l'utilisateur doit avoir. On remarque que le FE peut ne pas subir le contrôle de MAP en spécifiant simplement l'option *PasseContrôle* avec l'opération à exécuter.

De ne pas pré-définir les sémantiques des droits d'accès individuels donne une plus grande flexibilité aux FE. Par exemple, pour la même opération, le droit d'accès nécessaire peut être différent si elle fait partie des opérations complexes spécifiques à l'application. Pour un même item, un utilisateur peut modifier certains attributs, mais pas les autres si ces derniers s'agissent

des attributs d'administration. Cela nécessite d'associer différents droits d'accès au même type d'opérations de MAP.

Comme les autres données affectées par les FE, MAP définit le format de données de DroitsD'Accès (AR pour Access Right) qui est simplement un entier .

7.3.2. Descripteur d'Ensemble d'Utilisateur

Les *DescripteurD'EnsembleD'Utilisateurs* (*USD* pour UserSetDescriptor) suivent le modèle d'utilisateurs de MAP pour identifier un ensemble d'utilisateurs.

Un **USD** se compose de quatre parties:

(1) *Flag*: il prend une valeur booléenne spécifiant si les utilisateurs identifiés sont affectés/accordés ou exclus des droits d'accès spécifiés dans la paire. Il indique si la paire est *positive* ou *négative*.

(2) *Activity_Part*: C'est un AID identifiant l'activité auquelle des utilisateurs participent.

(3) *User_Part*: cette partie est une liste d'UID identifiant les utilisateurs portant ces identificateurs. Si cette partie est absente, tous les utilisateurs de l'activité sont identifiés.

(4) *Role_Part*: c'est une listes de RID identifiant les rôles de l'activité. Si elle est absente, la partie identifie tous les rôles.

Les utilisateurs identifiés sont ceux qui participent à l'activité identifiée par Activity_part, et identifiés par User_Part et jouent un des rôles identifiés par Role_Part. Une ACL ainsi spécifiée est fonctionnellement complète, c'est une combinaison **ou-et** entre les paires positives. De plus les paires négatives rendent la spécification plus simple et facile.

Remarque: Chaque USD concerne une seule activité. Une ACL peut concerner plusieurs activités et une seule activité peut avoir plusieurs paires dans l'ACL.

Par exemple, pour spécifier "tous les *lecteur* de l'activité *Conf1* ont le droit de *lire*", l'ACL est:
Paire1:

- AR: `Lire`
- Flag: `Positif`
- Activity_Part: `Conf1`
- User_Part: `Tous`
- Role_Part: `Lecteur`

Pour spécifier que "tous les membres de l'activité *enseignement*" ont le droit de *stocker* sauf les *étudiants*, l'ACL est:
Paire1:

- AR: `Stocker`
- Flag: `Négatif`
- Activity_Part: `Enseignement`
- User_Part: `Tous`
- Role_Part: `Etudiant`

Paire2:

- AR: `Stocker`
- Flag: `Positif`
- Activity_Part: `Enseignement`
- User_Part: `Tous`
- Role_Part: `Tous`

Remarque: l'ordre est important!

Pour spécifier qu'un utilisateur *MonsieurToto* a tous les droits, l'ACL est:
Paire1:

- AR: `Tous`
- Flag: `Positif`
- Activity_Part: `Tous`
- User_Part: `MonsieurToto`
- Role_Part: `Tous`

Remarque: cet exemple montre la puissance de l'expression de l'ACL. Cependant, dans une application MHS et GC, cela signifie la mauvaise conception de l'application, puisque les fonctionnalités d'un utilisateur sont spécifiées par ses rôles, mais pas en tant qu'un utilisateur individuel.

7.4. Procédure du Contrôle d'Accès de MAP

Quand un FE passe une opération à MAP pour l'exécuter, il passe les informations suivantes pour le contrôle d'Accès:
- Identification de l'utilisateur qui demande cette opération, i.e. <UID, AID, RID>.
- Si un contrôle doit être effectué par MAP.
- Si oui, le(s) droit(s) d'accès nécessaires pour l'exécution de l'opération.
- L'identificateur d'un Contrôleur si le FE n'utilise pas celui (indirectement) associé à l'item.

Quand MAP reçoit cette demande d'exécution d'opération, la procédure du contrôle se déroule comme suite:
(1) Il décide si le contrôle d'accès est demandé par le FE, si non la procédure du contrôle d'accès est positive, l'exécution continue.
(2) Si oui, MAP localise le Contrôleur utilisé: s'il n'est pas spécifié explicitement dans la demande d'opération, le Contrôleur associé à l'item est utilisé. (voir la section Contexte pour les détails)
(3) MAP vérifie les droits d'accès auprès de l'ACL du Contrôleur comme décrit ci-dessus. Le résultat de la vérification nous permet de savoir si cette opération peut être exécutée ou non.

7.5. Manipulation de Contrôleurs

MAP gère une base de contrôleurs qui peuvent être manipulés par les utilisateurs privilégiés, ces manipulations ne sont pas contrôlées par MAP, mais à la charge des FE (Nota: au point de vue de la spécification du service abstrait, les utilisateurs manipulent les contrôleurs via la *porte Administrateur* qui est différente de celle qui permet de manipuler la base d'information). Un utilisateur peut:

- Créer un Contrôleur. Le nom du nouveau contrôleur est donné comme résultat.

- Modifier un Contrôleur, i.e. remplacer l'ACL du contrôleur par une nouvelle.
- Supprimer un Contrôleur. Le contrôleur ne peut être supprimé s'il est associé à des objets dans MAP.
- Lire un Contrôleur: obtenir l'ACL d'un contrôleur.
- Lister tous les contrôleurs: lister les ACID de tous les contrôleurs dans la base.

MAP permet en plus aux FE de pouvoir vérifier les droits d'accès pour un utilisateur sans exécuter vraiment l'opération.

- Vérifier: les arguments sont l'utilisateur (le triplet), les droits d'accès et le contrôleur avec lequel les droits d'accès de l'utilisateur sont à vérifier. Cette possibilité est très intéressante pour tester les contrôleurs et pour assurer qu'un ensemble de manipulations peut être accepté, avant de les exécuter.

8. Contextes

MAP gère une base de contextes. Un contexte peut être vu comme une fenêtre sur la base d'information à travers laquelle les utilisateurs accèdent à une partie de la base, c'est à dire qu'un contexte est **un environnement de travail** qui associe un ensemble d'utilisateurs à un ensemble d'objets d'information (cf. Figure 5-14). On dit aussi que le contexte *contient* les objets visibles. Chaque item appartient au maximum à un contexte directement.

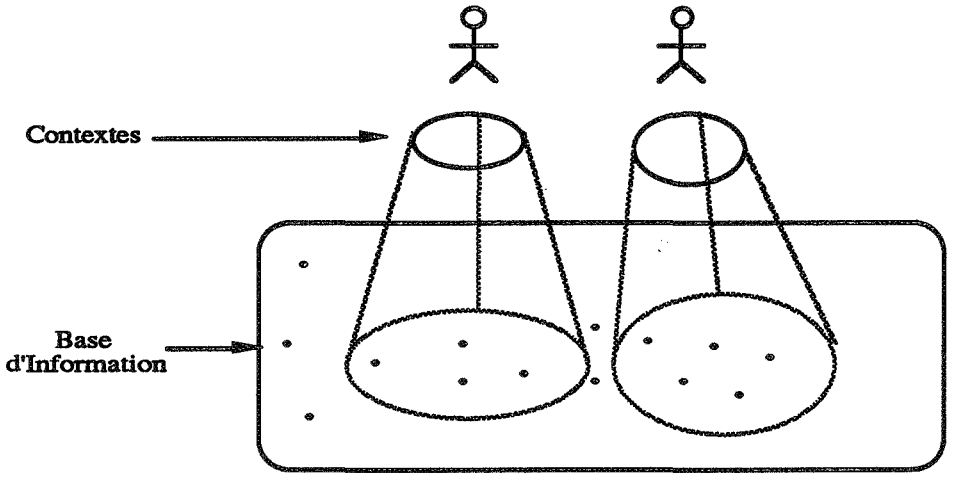


Figure 5-14. Contextes comme Fenêtres

Un utilisateur peut avoir des droits d'accès différents suivant les contextes. Un contexte est aussi **une unité d'administration** et **de gestion**. Les contextes peuvent être utilisés pour diviser la base d'information en plusieurs unités d'administration relativement indépendantes (par rapport aux objets composites qui contiennent aussi un ensemble d'objets d'information). Ceci peut permettre une administration plus facile, et en particulier le partage de la base par plusieurs applications (e.g., plusieurs boîtes aux lettres et/ou conférences, comme l'archiveur central de COM).

Un contexte peut être utilisé comme **une unité de distribution** ou **de répartition**, on peut ainsi distribuer ou répartir une partie de la base d'information dans les autres archiveurs et fournir des outils pour synchroniser les contextes distribués.

De plus, les contextes fournissent un moyen pour organiser des objets d'information autre que des objets composites, qui peuvent donner plus d'efficacité dans une réalisation bien pensée. Par exemple, on pourra indexer les items par contexte puisque chaque accès individuel se fait via un et un seul contexte; cela permet de mieux organiser l'espace mémoire (en particulier , la mémoire centrale) et accélérer l'exécution des opérations, e.g., la recherche.

8.1. Caractéristiques et Définition de Contextes

Chaque contexte contient un ensemble d'items de la base d'information, et les informations nécessaires pour accéder et manipuler ces items. Chaque accès d'un utilisateur à la base d'information doit *obligatoirement* passer par un contexte, i.e. chaque opération doit être effectuée dans un contexte.

Comme un objet d'information, chaque contexte a un identificateur interne et/ou un nom externe qui permettent de l'identifier, et un ensemble de champs (pour ne pas confondre avec les attributs attachées à des items). Chaque contexte contient une description informelle. En tant qu'un environnement de travail et unité d'administration, chaque contexte spécifie un ensemble d'attributs *utilisables* ou *reconnus* dans le contexte; seuls ces attributs peuvent être lus et manipulés dans le contexte. Comme mentionné dans la section du contrôle d'accès, à chaque contexte est associé un *Contrôleur* d'accès qui est utilisé pour contrôler les accès via le contexte. Un contexte spécifie aussi un ensemble d'opérations qui peuvent être exécutées dans le contexte.

En général, les contextes sont également enregistrés dans le Directory ou gérés par les FE. Donc seuls les champs significatifs pour MAP sont enregistrés dans la base avec des contextes, les autres champs (attributs de distribution, d'administration, ...) peuvent être enregistrés dans le Directory ou gérés par des FE. Voici la liste des champs associés à un contexte:

- Identificateur Interne: c'est un entier généré par MAP, il est unique dans MAP.
- Nom Externe: une chaîne de caractères donnée par l'utilisateur.
- Description du Contexte: un segment de texte décrivant brièvement le contexte.
- Super-Contextes et Sous-Contextes: nous présenterons ces deux attributs dans la section 8.4.
- Attributs d'items utilisables/reconnus dans le contexte.
- Contrôleur: le Contrôleur associé pour contrôler les accès via le contexte.
- Opérations permises: les opérations manipulant la base d'information qui peuvent être effectuées dans ce contexte. Il faut noter que cet attribut est différent du contrôle d'accès qui spécifie les droits des utilisateurs. Cet attribut indique la *capacité* et *vue fonctionnelle* du contexte. Il est aussi utilisé pour optimiser l'implantation.
- Données spécifiques: une chaîne d'octets qui permet aux FE d'attacher leurs propres données à ce contexte.

Remarque: la notion d'*administrateur* n'existe pas explicitement dans MAP. En fait, un administrateur *local* ou *global* est simplement un utilisateur avec certains privilèges. C'est à la

charge des FE d'authentifier l'administrateur (s'il existe dans l'application) et de lui affecter les droits nécessaires.

L'association d'un item à un contexte est faite par l'exécution de l'opération *Stocker* qui fonctionne comme décrit dans la section 5.1.10.1, et qui inclut implicitement l'item à stocker, au contexte dans lequel l'opération est exécutée. Un item peut être exclu d'un contexte en l'*effaçant* au niveau **0**. Attention: l'objet lui-même n'est pas effacé dans la base.

Des deux points précédents, on peut déduire qu'il est possible qu'un item (du niveau 1 ou plus) n'appartienne à un instant donné à aucun contexte, à savoir lorsqu'il est exclu du contexte auquel il appartenait. Un tel item peut être ensuite re-inclus dans un contexte. Un item fictif n'appartient à aucun contexte quand il est créé automatiquement par MAP. Il faut noter que pour re-inclure un item, il est nécessaire de connaître son nom; ceci implique qu'il est possible qu'un item soit *oublié* éternellement et ainsi ne jamais être supprimé de la base. Une facilité de MAP permet d'obtenir tous les items *oubliés*, i.e. qui n'appartiennent à aucun contexte et n'ont pas de lien entrant.

8.1.1 Attributs Reconnus dans un Contexte

Chaque contexte qui ouvre une fenêtre sur la base d'information spécifie un sous-ensemble d'attributs qui sont significatifs dans le contexte. Les items dans le contexte peuvent avoir d'autres types d'attributs attachés, mais aucune manipulation de ces attributs n'est possible, même pas la lecture.

Donc la vue d'un item dans le contexte est:
- tous les attributs de l'item qui sont reconnus/utilisables par le contexte.
- le contenu de l'item.

Il est possible d'avoir des parties *invisibles* dans un contexte pour un item donné, ces attributs invisibles dans un contexte peuvent être stockés via le contexte, et peuvent ensuite être déclarés dans le contexte. Cela permet de changer dynamiquement des visions dans un contexte.

8.1.2. Contexte comme un Environnement de travail

Un contexte est avant tout un environnement de travail. Un utilisateur voulant accéder ou manipuler des **items** doit obligatoirement le faire via un contexte. Chaque contexte a un **Contrôleur** par défaut. Le Contrôleur par défaut d'un contexte est conceptuellement associé à chaque item du contexte. Différents contextes peuvent avoir différents Contrôleurs par défaut. L'environnement de travail se manifeste aussi par l'ensemble des champs d'un contexte:

- seul *un sous-ensemble d'opérations* sont possibles dans le contexte. En particulier, on pourra ne pas permettre la recherche dans un contexte, mais seulement dans un des sous-contextes (voir les textes suivants).

- le champ de vision de l'opération est l'ensemble des items du contexte. Par exemple, l'opération *Rechercher* effectuée dans un contexte ne peut chercher que parmi l'ensemble des items du contexte.

Remarque: un contexte est un environnement de travail pour des objets d'information, mais pas pour d'autres ressources de la base comme les Contrôleurs, l'Historique, etc.

8.1.3. Contextes en tant qu'Unités de Distribution

Un contexte peut être utilisé comme une unité de distribution. Les manipulations des éléments d'un contexte ne peuvent s'effectuer que dans le contexte. Le mécanisme d'historique permet aussi de traiter un contexte comme une unité de distribution, il garde une historique logique pour chaque contexte, en tenant compte de leurs sous-contextes. Le mécanisme d'historique permet de plus d'extraire les seules opérations modifiant des données globales (par exemple, des objets globaux et des attributs globaux) dans l'historique d'un contexte. Ainsi on peut distribuer un contexte (i.e. ses éléments) dans plusieurs archiveurs, et utiliser le mécanisme d'historique pour synchroniser le contexte dans ces archiveurs distribués.

8.2. Relations entre les Contextes

Les contextes ne sont pas indépendants:
(1) Un contexte peut être **sous-contexte** d'un autre contexte. Tous les éléments (items) d'un sous-contexte sont aussi ceux du **super-contexte.**
(2) Les éléments appartenant aux différents contextes peuvent se relier par des liens typés, c'est à dire qu'il peut avoir des liens *cross-contextes.*

Un contexte peut avoir zéro, un ou plusieurs super-contextes, et zéro, un ou plusieurs sous-contextes. Ainsi les contextes forment une hiérarchie (Figure 5-15), la racine s'appelle *top-contexte* qui est le seul contexte n'ayant pas de super-contexte. Les éléments d'un contexte peuvent être divisés en deux catégories:

- les **éléments directs** qui appartiennent directement au contexte,
- les **éléments indirects** qui sont éléments directs ou indirects de ses sous-contextes.

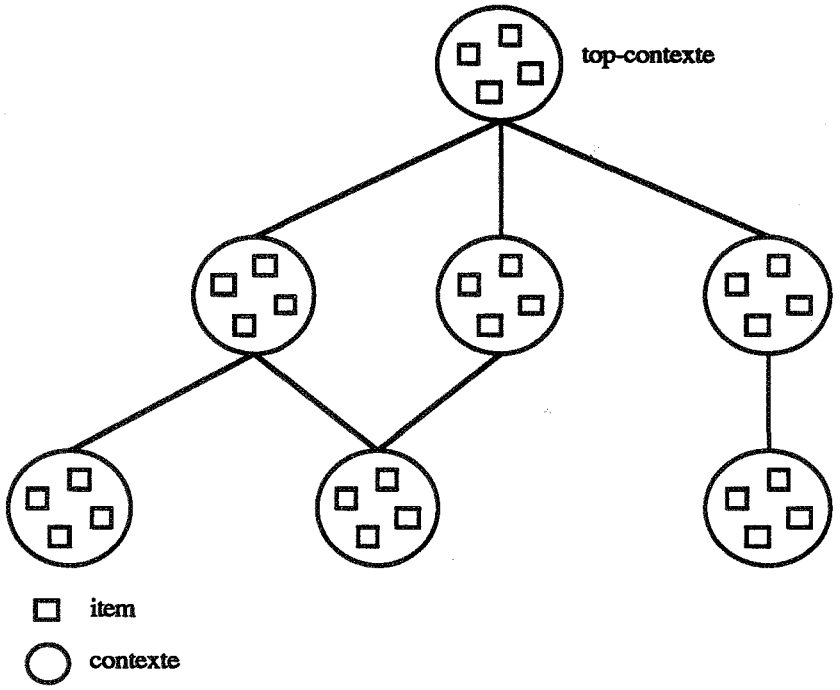


Figure 5-15. La Hiérarchie de Contextes

Dans un contexte, on peut *lire*, *rechercher* et *résumer* tous les éléments du contexte, qu'ils soient directs ou indirects; mais on ne peut pas modifier les éléments indirects. Cela implique qu'un item ne peut être manipulé que dans le contexte où il est un élément direct. Pour faciliter les utilisations, une opération spécifique *context-switch* (basculer un contexte) est fournie par MAP, cette opération permet de changer *dynamiquement* le contexte de travail.

D'un point de vue administratif, un *super-administrateur* doit s'occuper de la cohérence entre contextes. L'*administration locale* de chaque contexte (définition des attributs, changement du Contrôleur, etc) est effectuée par un administrateur local sous la contrainte du super-administrateur. La relation d'appartenance d'un fils à son père dans l'arbre sera désignée par *super-/sous-contexte.*

8.2.1. Héritage entre Contextes

Un sous-contexte contient un sous-ensemble d'items du super-contexte et **hérite** par défaut des attributs du super-contexte. Le but des sous-contextes est de fournir un moyen logique de diviser un super-contexte et donc de diviser toute la base. Ces sous-contextes peuvent ensuite servir comme environnement de travail et/ou unité de distribution. Par exemple, on pourra diviser une conférence en sous-conférences et ne distribuer qu'une seule sous-conférence ou permettre à quelqu'un de travailler (lire, par exemple) dans une sous-conférence seulement.

L'héritage suit les règles suivantes:

1. Un item ajouté dans un sous-contexte devient automatiquement élément indirect de tous les super-contextes du contexte.

2. Tous les attributs reconnus dans un contexte sont aussi reconnus dans ses sous-contextes. Donc les attributs reconnus dans un contexte sont la réunion de ceux reconnus dans tous ses super-contextes, et ceux déclarés dans le contexte lui-même.

3. Chaque attribut reconnu dans un contexte est soit *déclaré* directement dans le contexte, soit *hérité* des super-contextes. L'ajout ou le retrait d'un attribut reconnu dans un contexte ajoute ou supprime l'attribut hérité dans tous ses sous-contextes.

8.2.2. Top-Contexte

Chaque MAP a un top-contexte qui est la racine de tous les autres contextes. Il est créé pendant l'initialisation du MAP et ses attributs sont initialisés comme suit:

- Nom interne: 0.
- Nom externe: optionnellement donné par l'utilisateur qui initialise MAP.
- Description: donnée par l'utilisateur qui initialise MAP.
- Super-contextes: Vide (Inexistant)
- Sous-contextes: Vide
- Attributs Utilisables: tous les attributs définis comme *universels* de MAP.
- Contrôleur par défaut: Vide

• Opérations permises: toutes

Ce contexte ne peut être supprimé!

8.3. Manipulation de Contextes

Les manipulations de contextes sont souvent effectuées par des utilisateurs privilégiés. Voici la liste des opérations:

• Créer un nouveau contexte en spécifiant les super-contextes et les champs du contexte à créer, qui peuvent simplement être hérités du super-contexte. Le résultat est le nom interne du contexte, généré par MAP.

Remarque: le Top-Contexte est créé pendant l'initialisation de MAP.

• Ajouter une relation "SousContexteDe" entre deux contextes. L'héritage est fait comme l'effet de l'exécution de l'opération. Il ne doit pas y avoir de boucle entre contextes, on doit toujours garder la structure hiérarchique.

• Couper le lien "SousContexteDe" entre deux contextes. Lorsqu'un contexte n'a plus de super-contexte, il est *supprimé*. Dans ce cas, MAP d'abord coupe ses liens avec ses sous-contextes et exclut ses éléments directs. Ce processus peut être récursif.

• Déclarer un nouvel attribut dynamiquement: dans le cas d'un contexte non-feuille, cette opération propage la déclaration à tous les sous-contextes directs et indirects.

• Supprimer un attribut non-hérité. L'attribut supprimé n'est plus visible dans le contexte, mais les instances de l'attribut ne sont pas physiquement enlevées des items.

• Changer le Contrôleur par défaut. Lorsque le nouveau Contrôleur est vide, deux cas sont possibles:
- soit un contrôleur est spécifié explicitement pendant l'exécution et il sera utilisé pour contrôler l'opération,
- soit plus aucun contrôle n'est effectué.

• Changer l'ensemble d'opération permises.

• Remplacer les données spécifiques.

A cause de l'introduction de contextes, les sémantiques de certaines opérations de la manipulation de la base d'information ont légèrement changées, et une nouvelle opération est introduite:

- *Stocker* un item associe implicitement l'item à ajouter au contexte où l'opération s'effectue. Une erreur est produite si l'item est déjà élément direct d'un autre contexte.
- *Effacer* un item à niveau 0 dissocie l'item de son contexte, et donc de tous les contextes. Remarquons que l'item doit être l'élément direct du contexte où on effectue l'opération.
- *Rechercher* sans sélecteur dans un contexte donne tous les éléments (directs et/ou indirects) du contexte, MAP indique pour chaque élément indirect son contexte direct où l'on peut modifier cet item.

- *Context-Switch* est une nouvelle opération qui bascule l'environnement de travail de la session actuelle, à un autre contexte indiqué dans un argument. Cela permet, par exemple, de modifier un item qui se situe dans un autre contexte que le contexte de travail (après y avoir trouvé par la recherche), sans créer une autre session. L'analogie à l'opération "cd" (change directory) permet sans doute de mieux comprendre cette opération.

9. Recherche d'Information

La fonctionnalité essentielle d'un archiveur de messages est de pouvoir *lire* des messages/objets d'information stockés. Sans la fonction *rechercher*, un utilisateur doit savoir les noms des messages qu'il *veut* lire. Cela est presque impossible quand la quantité de messages est assez importante. La fonction de **recherche d'information** dans un archiveur permet d'abord à un utilisateur de trouver les références des objets qu'il veut, i.e. qui satisfont un *critère de recherche*. Il faut noter que c'est aussi cette fonction qui est souvent omise ou insuffisante dans les archiveurs existants.

9.1. Besoins et Choix

Sans exception, MAP doit aussi fournir les fonctionnalités pour la recherche d'information.

Les utilisateurs doivent d'abord pouvoir *grouper* leurs items dans des *collections/classeurs* d'après leurs propres critères de classification, et ensuite les retrouver en lisant simplement le contenu des classeurs.

Il est nécessaire de pouvoir retrouver des items d'après leurs *propriétés*, au moins les importantes, par exemple, le sujet, les mots-clé, etc. Ce genre de recherches est souvent appelé **requêtes analytiques.**

Il est indispensable de rechercher des items d'après leurs *relations avec les utilisateurs* dans un archiveur de messages. Par exemple, retrouver tous les messages envoyés par un utilisateur, i.e. rechercher tous les messages qui ont des relations *expéditeur* avec l'utilisateur.

L'existence de nombreuses relations entre messages fournit un moyen naturel de retrouver des messages à partir de ceux connus. Donc on doit pouvoir faire une recherche en *suivant les liens* entre items, cela peut être récursif; par exemple, trouver tous les items concernant (*RelateTo*) un item donné, directement ou *indirectement*. On appelle ce genre de recherches **recherche navigationnelle généralisée**, ou tout simplement **recherche navigationnelle.**

Le contenu d'un item est une autre source d'après laquelle on recherche des items, par exemple, trouver les messages qui contiennent des textes *lisibles* sur un terminal alphanumérique, i.e. d'après les types des parties de contenus. En plus, il est parfois intéressant de pouvoir rechercher des messages qui contiennent certains mots ou phrases, i.e. **recherche de plein texte.**

Parfois, les utilisateurs ne savent pas exactement ce qu'ils préfèrent lire, mais juste *feuilletent* (brows) les messages. Donc il est intéressant de pouvoir lire une partie d'un item sélectionné, en suivant des liens.

Le choix de MAP est de fournir les facilités suivantes pour la recherche:

- Fournir des items *composites* comme classeurs des messages (items), cela est présenté dans les sections précédentes.
- Lire une partie quelconque d'un item, c'est la fonctionnalité de l'opération *lire*.
- Obtenir les *liens* ou *relations item-utilisateur* qui sont aussi fournies par les opérations *L_Get* et *U_Get*.
- Rechercher d'après (théoriquement) n'importe quelle propriété des items, càd *tous les attributs* sont théoriquement utilisables pour la recherche. Cependant MAP distingue deux classes d'attributs. Les attributs *prédéfinis* sont assurés d'être utilisés efficacement (en les indexant, par exemple), les attributs spécifiques sont utilisables, mais sans performance. Pour la dernière catégorie, les FE doivent considérer que MAP *pique* simplement chaque item et voit si la condition est satisfaite. Ce service n'est pas fournit dans la version actuelle de MAP. Il sera donc préférable que les FE gèrent eux-mêmes des indexes pour ces attributs. On doit noter que seuls les attributs définis comme *recherchables* peuvent être utilisés.
- Rechercher d'après *tous les liens*.
- Rechercher d'après *toutes les relations* avec utilisateurs.
- Rechercher d'après les types de CP du contenu. La recherche *plein texte* n'est pas permise dans la version actuelle, à cause de sa complexité et du fait que cette facilité n'est pas très demandée dans un environnement MHS et GC. En fait, les mots-clé sont souvent suffisants pour les besoins de la recherche textuelle.

9.2. Opération Rechercher

Le mécanisme principal de recherche d'information de MAP est l'opération *rechercher*. Cette opération prend deux arguments:

- un *champs de vision* qui spécifie un ensemble d'items à partir duquel la recherche s'effectue.
- un *sélecteur* qui spécifie le critère de choix que les items à chercher doivent satisfaire.

Le résultat de l'opération est un ensemble de noms et localisations des (zéro, un ou plusieurs) items trouvés. En utilisant cette opération avec les arguments appropriés, un FE peut effectuer la requête analytique, la recherche navigationnelle, la recherche d'après les relations item-utilisateur et la recherche d'après les types de contenu. Cette opération est conçue pour avoir une bonne performance qui doit au moins être acceptable pour les utilisateurs *en-ligne*.

9.2.1. Champs de Vision

Le *champs de vision* spécifie un ensemble d'items où l'opération effectue sa recherche. L'idée initiale est de permettre non seulement d'identifier un ensemble d'items explicitement par leurs

noms, mais aussi implicitement en indiquant la collection (un item composite) auquelle ils appartiennent.

Le champs de vision d'une opération *rechercher* est soit explicitement un ensembles d'items, soit un item composite avec une option indiquant que le champs de vision est l'ensemble de tous les composants directs de l'item ou tous les composants directs et indirects. Cela nous permet de chercher des items dans un classeur au lieu du contexte total. Par défaut, le champs de vision d'une opération est l'ensemble de tous les éléments du contexte où l'opération est exécutée.

9.2.2. Sélecteur

Un *sélecteur* se compose d'un indicateur de classe et d'une combinaison logique (AND/ET, OR/OU, NOT/NON) des **filtres**, qui peut être récursive. L'indicateur de classe spécifie la classe dont les items à chercher sont instances directes ou indirectes. Les quatre types de filtres sont définis:

- **Filtres Analytiques** permettent de tester l'existence d'un attribut recherchable et comparer le(s) valeur(s) d'un attribut recherchable.
- **Filtres Liens** spécifient des conditions complexes sur les liens entre items, en plus de simple comparaison des valeurs (puisque la plupart des liens sont aussi représentés comme des attributs *recherchables*, les simples comparaisons de liens peuvent aussi être spécifiées par des *filtres analytiques).*
- **Filtres Utilisateurs** spécifient des conditions complexes sur les relations item-utilisateur, en plus de simple comparaison des valeurs.
- **Filtres Contenus** qui spécifient les conditions sur les types de CP.

9.2.2.1. Filtres Analytiques

Chaque FA (pour Filtre Analytique) est associé à un attribut recherchable, il spécifie une condition sur l'attribut. L'évaluation de chaque filtre auprès d'un item donne une valeur *VRAI* ou *FAUX*. Un FA est composé de trois éléments:

- Un type qui spécifie la relation entre le(s) valeur(s) de l'attribut et les *paramètres* donnés.
- Un flag qui indique la valeur du filtre en cas d'absence de l'attribut, puisqu'il est possible que l'attribut soit présent dans certains items mais absent dans d'autres.
- Un ensemble de paramètres. Leur relation avec le(s) valeur(s) est spécifiée par le type, le nombre de paramètres dépendant du type.

On donne ici les types de FA, ils ne sont pas valides pour tous les types d'attributs, e.g., il est impossible de comparer si un lien est *supérieur* ou *inférieur* à un autre lien.

- *equal*: le filtre est VRAI si une valeur de l'attribut est égale au paramètre donné. Le nombre de paramètres est 1.
- *inequal*: le fitre est VRAI si aucune valeur de l'attribut est égale au paramètre donné. Le nombre de paramètres est 1.
- *subset*: le filtre est vrai si l'ensemble de valeurs de l'attribut (une ou plusieurs) est un sous-ensemble des paramètres.

- *supset*: le filtre est vrai si l'ensemble de valeurs de l'attribut (une ou plusieurs) est un sur-ensemble des paramètres.
- *greater*: Ce type de filtres n'est valide que pour les attributs mono-valeur. Le filtre est VRAI si la valeur de l'attribut est *supérieure* au paramètre donné. Le nombre de paramètres est 1.
- *greater-or-equal*: Ce type de filtres n'est valide que pour les attributs mono-valeur. Le filtre est VRAI si la valeur de l'attribut est *supérieure ou égale* au paramètre donné. Le nombre de paramètres est 1.
- *less*: Ce type de filtres n'est valide que pour les attributs mono-valeur. Le filtre est VRAI si la valeur de l'attribut est *inférieure* au paramètre donné. Le nombre de paramètres est 1.
- *less-or-equal*: Ce type de filtres n'est valide que pour les attributs mono-valeur. Le filtre est VRAI si la valeur de l'attribut est *inférieure ou égale* au paramètre donné. Le nombre de paramètres est 1.
- *between*: Ce type de filtres n'est valide que pour les attributs mono-valeur. Le filtre est VRAI si la valeur de l'attribut est *entre* deux paramètres donnés, i.e. *supérieure* ou *égale* au premier et *inférieure* ou *égale* au deuxième. Le nombre de paramètres est 2.
- *like*: Ce type de filtres n'est valide que pour les attributs ayant le type de valeur *chaîne de caractères*. Le filtre est VRAI si une valeur de l'attribut "*match*" avec le paramètre, i.e. le paramètre est une *sous-chaîne* de la valeur. L'extension possible est de permettre d'avoir des *vrais matchs*, avec des expressions régulières.

L'opération rechercher avec les filtres analytiques réalise la requête analytique. Un exemple est de trouver `tous les messages dans "Collections d'Annoncements de Réunions" avec le sujet comme "Problèmes d'Enseignement de 89"`. Il faut noter que la classe dont un item est une instance directe est représentée comme la valeur d'un attribut de l'item recherchable (*ObjectType*), il est donc possible de spécifier un FA sur l'attribut "*ObjectType*". Mais dans ce cas, MAP ne tient pas compte de la sémantique de la classe, ni de la hiérarchie de classes, il effectue simplement la comparaison de valeurs.

9.2.2.2. Filtres Contenus

Un filtre contenu est simplement un ensemble de types de CP, le filtre est VRAI pour un item si l'item contient au moins une CP dont le type est parmi l'ensemble spécifié.
Par exemple, on peut utiliser le filtre contenu pour trouver `tous les items contenant au moins un "graphe"`.

9.2.2.3. Filtres Liens

Un filtre lien (FL) permet aux utilisateurs de spécifier les items désirés par leurs liens à un item donné. La forme générale d'un FL est

Is {(L1...Ln)|*}[+] {B|U} Id,

où *Is* est l'item source et *Id* est l'item destination, un des deux est un item connu, identifié par son nom, l'autre est implicitement l'item à trouver; *L1, ..., Ln* sont des types de liens et * signifie tous les types; + désigne la récursivité; *B/U* spécifie si la direction de liens est considérée (Uni-direction) ou pas (Bi-Direction); *[]* signifie optionnel, *{}* signifie le choix entre les possibilités inclus entre { et }. La sémantique du filtre est que *Is est lié au Id directement ou indirectement (si*

*+) par les liens sortants (si U); ou sortants et entrants (si B) des types L1, ..., Ln, ou de n'importe quel type (si *).* Cela permet aux utilisateurs de trouver tous les messages directement ou indirectement liés à un message initial, à savoir, une conversation.

Pour mieux comprendre le filtre lien, nous donnons ici quelques exemples de ses utilisations, nous supposons que les items sont liés comme dans la figure 5-16:

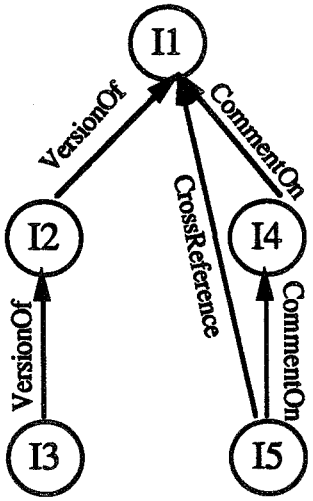


Figure 5-16

• `? (CommentOn | CrossReference) U I1`
trouve tous les items qui répondent ou ont une référence croisée à l'item nommé *I1*. Le résultat est (*I4*, *I5*).

• `? (VersionOf) + B I2`
spécifie toutes les versions plus anciennes ou plus récentes de l'item *I2*. Le résultat est (*I1*, *I3*).

• `? (*) + B I2`
trouve tous les items liés à l'item nommé *I2*. Le résultat est (*I1*, *I3*, *I4*, *I5*).

9.2.2.4. Filtres Utilisateurs

Un FU permet de trouver les items qui ont des relations avec un utilisateur donné. Sa forme générale est

(T1...Tn|*) User

où *T1,...,Tn* sont les types des relations item-utilisateur, *User* est un identificateur de l'utilisateur avec lequel les items à trouver ont au moins une relation du type T1, ..., Tn, ou n'importe quel type (si *). Par exemple (cf. Figure 5-17):

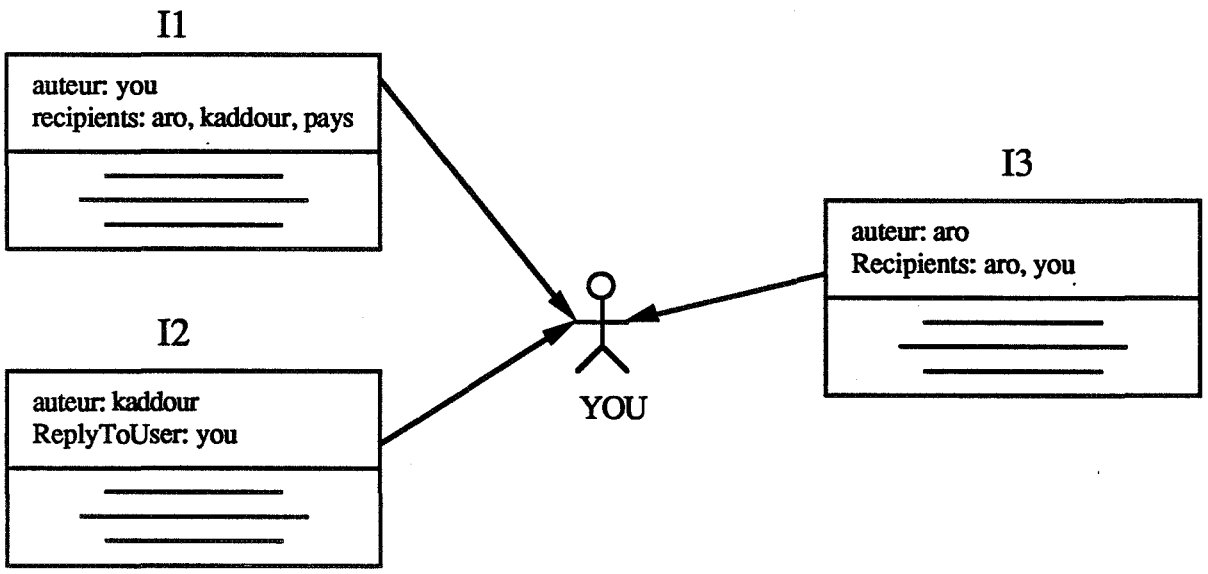


Figure 5-17

• `(ReplyToUser | recipients) YOU`
spécifie les items qui est une réponse à l'utilisateur *YOU* ou a pour destinataire *YOU*. Le résultat est (*I2, I3*).

• `(*) YOU`
trouvera tous les items qui concernent *YOU*. Le résultat est (*I1, I2, I3*).

Pour finir notre description de l'opération rechercher, on donne un exemple plus complexe: `Trouver les items qui sont entrés dans l'archiveur` *`entre`* `le 01/09/89 et le 31/09/89,` **`ET`** `avec l'importance "high"` **`OU`** `"normal"` **`ET`** `lié directement ou indirectement à un item nommé "EMSE+20"` **`ET`** `contient un CP "texte"` *`dans`* `le sous-thème "archivage" de la conférence "MMM".`

9.3. Conclusion

Dans cette section, nous avons présenté le mécanisme de recherche de MAP. Ce mécanisme consiste principalement en l'opération *rechercher*, il permet les différentes méthodes de recherche adaptées aux applications MHS et GC.
Pour terminer, nous indiquons ici les futures directions de recherche dans le domaine de la recherche d'information dans un archiveur de messages:

- Permettre la recherche d'après des attributs définis par les FE spécifiques aux applications et en même temps assurer la bonne performance du mécanisme.
- Permettre un "pattern-matching" plus sophistiqué, à la place de simple "sous-chaîne" dans la version actuelle.
- Permettre la recherche "plein texte" d'après des contenus de messages, y compris des images, des graphes, ou mêmes des sons.

10. Mécanisme d'Historique

MAP fournit un mécanisme d'historique pour mémoriser et gérer l'historique de la base d'information. L'historique contient toutes les opérations effectuées sur la base d'information et les différents états de la base. Elle peut être utilisée pour:

- Réserver des informations sur ce qui s'est passé. Il permet même de conserver les opérations qui ne peuvent être exécutées au moment de leur lancement, à cause du fait que l'item à manipuler n'est pas complet ou de la violation des droits d'accès. Les FE peuvent re-exécuter ces opérations après que l'item ait été complété.

- Récupérer et Sauvegarder. Les anciens états des items sont conservés directement par des versions ou indirectement déduits à partir de l'état des items et des opérations ayant été exécutées.

- La gestion et l'administration de la base. Les administrateurs peuvent savoir ce qui s'est passé et qui a exécuté quelle opération dans la base.

- La distribution et la synchronisation d'informations. Les opérations conservées dans l'historique peuvent être appliquées/propagées vers les autres bases dupliquées.

Le mécanisme d'historique mémorise toutes les informations sur le passé des objets d'information de la base, leurs états, les changements et les tentatives de changement. On appelle l'ensemble des informations l'**historique** de la base.

Le mécanisme d'historique permet d'obtenir des informations historiques et de les utiliser. En particulier, les FE peuvent obtenir toutes les opérations exécutées dans un contexte et/ou ses sous-contextes. Ils peuvent aussi n'extraire que les opérations qui manipulent les données globales (des items et des attributs globaux).

10.1. Représentation de l'Historique

Pour enregistrer des changements ou des évolutions de la base, on dispose de deux possibilités:

- Par *versions*: enregistrer chaque état, i.e. toutes les versions de tous les items aux différents moments. Cette solution est techniquement simple mais très coûteuse puisqu'elle utilise beaucoup de mémoire pour conserver les versions.

- Par *événements*: on conserve les opérations et les informations concernant leur exécution, par exemple, l'utilisateur qui exécute l'opération, les données que l'opération a changées, etc. Une conception appropriée permet de déduire tous les états. Cette solution est plus économique, elle est fonctionnellement supérieure à la solution précédente. Son inconvénient est que la suppression d'un événement perd la possibilité de *revenir en arrière.*

Notre choix est une combinaison des deux possibilités, basée principalement sur les *événements* (cf. figure 5-18).

- Par défaut: un événement est enregistré automatiquement chaque fois une opération est exécutée (avec succès ou non).
- Par la demande explicite, on enregistre la version actuelle d'un item.

Ainsi toutes les manipulations (même celles qui sont exécutées sans succès et celles qui ne modifient pas l'état de la base) sont enregistrées automatiquement, de plus les utilisateurs peuvent demander explicitement de conserver une version d'un item à un instant donné.

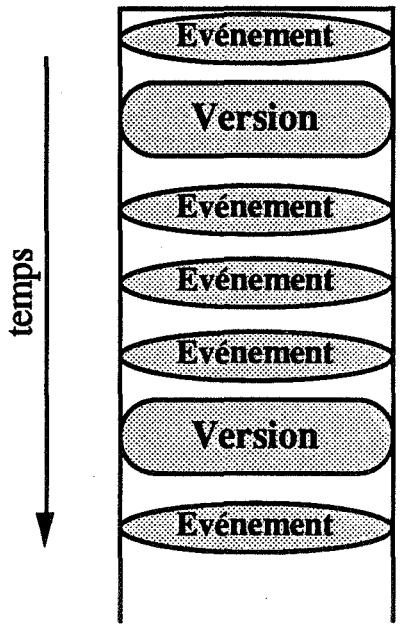


Figure 5-18. Historique

Pour donner plus de flexibilité aux FE, à chaque opération venant d'un FE est associé un indicateur qui spécifie si l'événement de cette opération doit être enregistré. Ainsi les FE peuvent passer complètement le mécanisme de l'historique s'ils en ont besoin, ils peuvent aussi n'enregistrer que les opérations qui modifient la base.

10.2. Evénements

Chaque événement correspond à une opération demandée par un FE, un événement est composé de:

• *Données locales*: ce sont les données locales à la base concernant l'exécution de l'opération.

- EID: l'identificateur de l'événement qui est affecté par MAP, i.e. par le mécanisme d'historique. L'EID est un entier qui représente le numéro de séquence de l'événement.
- Date-d'Exécution: la date d'exécution de l'opération.
- Contexte: le contexte dans lequel l'opération est exécutée.
- Local: si l'opération manipule des données locales ou globales. Ce champs n'est significatif que pour des opérations de modification.

• *Données de l'Utilisateur et de Contrôle d'accès* : ce sont les données concernant l'utilisateur qui crée ou demande cette opération (Nota: il le fait via un FE).

- Utilisateur: le triplet <UID, AID, RID> de l'utilisateur lance cette opération.
- Droits-Demandés: les droits nécessaires pour exécuter l'opération.
- Contrôleur: le Contrôleur utilisé pour contrôler cet accès, si le FE ne désire pas utiliser celui associé via contexte.

• *Données d'Opération*:

- OPID: l'identificateur de l'opération, il est donné par le FE. C'est une chaîne de caractères.
- Date-de-Création: la date de création de l'opération. Dans un environnement messagerie, l'opération peut être créée et ensuite envoyée par un message, ceci implique que le décalage existe entre la date de création et la date d'exécution.
- Type-d'Opération: le type d'opération, e.g., *stocker, lire, ...*
- Item-Manipulé: l'OID de l'item manipulé s'il s'agit de la manipulation d'un item individuel.
- Arguments: les arguments complets de l'opération.

• *Données de Résultat et de Sauvegarde*: les données qui concernent le résultat et l'ancien état de l'item.

- Etat-d'exécution: il indique si l'opération est exécutée avec succès.
- Sauvegarde: les données nécessaires pour revenir en arrière en cas de succès. Par exemple, l'ancienne valeur de l'attribut modifié, l'ancienne CP, etc.
- Raison-d'échec: la raison pour laquelle l'opération n'est pas exécutée avec succès, elle n'est présente qu'en cas d'échec.

• *Données Spécifiques*: données spécifiques associées à l'opération. Par exemple, l'archiveur où l'opération est initialement exécutée en cas d'archiveurs distribués. MAP permet de les stocker sans les manipuler, i.e. il les considère comme une chaîne d'octets.

Une demande d'opération doit présenter toutes les informations nécessaires sauf les données locales et les données de résultat et de sauvegarde.

10.3. Versions

Une *version* sauvegarde l'état d'un item à un instant donné, elle est créée par une demande explicite. Une version se compose de:

- *EID*: l'identificateur de la version, il prend un numéro de séquence d'événement.
- *Date-De-Création*: la date de création de la version, la version représente l'état de l'item de cet instant.
- *OID*: l'OID de l'item dont c'est une version.
- *item*: l'état de l'item , les attributs et le contenu.

10.4. Historique d'un Contexte

Puisque chaque item ne peut être modifié dans son contexte direct et que chaque opération s'exécute obligatoirement dans un contexte, on peut définir l'historique d'un contexte comme

l'ensemble d'événements des opérations exécutées dans le contexte. L'historique de la base est donc la somme des historiques de tous les contextes et de toutes les versions enregistrées.

L'idée ici est très simple. A partir d'un état identique, deux contextes dans deux archiveurs différents restent identiques après avoir exécuté la même séquence d'opérations, puisqu'un élément direct d'un contexte ne peut être modifié que dans ce contexte. Cela nous permet de synchroniser deux contextes dans deux différents archiveurs en utilisant les historiques des contextes, de même pour leurs sous-contextes.

MAP permet de plus aux FE d'obtenir tous les événements exécutés dans un contextes et dans ses sous-contextes. Cela facilite comme décrit ci-dessus la synchronisation et la distribution de contextes.

10.5. Service d'Historique

Le mécanisme d'Historique de MAP fournit un ensemble d'opérations pour manipuler l'historique, voici la liste:

- Demande d'enregistrement d'une version d'un item. Cette opération enregistre une version d'un item dans l'historique.
- Suppression des événements ou des versions identifiés par une liste d'EID.
- Lecture d'un événement identifié par son EID. On peut ne lire qu'une partie d'un événement.
- Obtention de l'état d'un item à un moment donné: MAP *calcule* automatiquement la version si les informations nécessaires sont disponibles. (i.e. si les événements ne sont pas détruits ou si une version appropriée existe)
- Sélection d'un ensemble d'événements d'après:
  - Identificateur de l'Opération
  - Item manipulé
  - Type de l'opération effectuée
  - Interval de temps pendant lequel l'opération est exécutée.
  - Effective: seuls les événements des opérations *effectives*. Une opération est *effective* à un instant donné si son effet sur l'item manipulé n'est pas écrasé par d'autres opérations.
  - Les derniers n événements

L'utilisateur peut aussi spécifier:

- le contexte dans lequel ont été exécutées les opérations, et si MAP doit compter aussi les opérations exécutées dans les sous-contextes du contexte.
- seuls les événements globaux (contenant des opérations qui manipulent des données globales) ou seuls les événements locaux (manipulent des données locales), ou les deux.

Nota: UNDO, REDO et REVENIR EN ARRIERE peuvent être réalisés très facilement en exécutant respectivement l'opération inverse, la même opération ou en remplaçant l'item par une

version précédente. On peut obtenir les informations nécessaires de ces opérations par le mécanisme d'historique. Donc ces fonctions ne sont pas fournies explicitement par MAP, les FE peuvent les réaliser en coopérant avec le mécanisme d'historique. Il faut noter que les modifications dynamiques d'un item sont extrêmement rares dans les applications se basant sur la messagerie, cela implique que UNDO REDO ... sont assez peu utilisés et donc justifie notre choix.

L'historique peut servir à beaucoup d'utilisations comme ce qu'on a présenté au début. Son utilisation en archiveurs distribués est particulièrement intéressante pour nous. Dans ce cas, il est utilisé pour synchroniser des contextes et items dupliqués dans plusieurs bases d'information. Puisque des changements sont représentés par des événements dans chaque base, et que ces changements sont disponibles via l'historique en la consultant simplement, une base peut propager ses changements soit en les diffusant par la base elle-même, ou en se mettant à disposition des autres bases pour qu'elles puissent consulter son historique. D'ailleurs, les événements eux-mêmes peuvent être considérés comme des objets ayant chacun un nom unique composé du nom de l'archiveur et son EID, ceci facilite de distribuer les opérations dans les différents archiveurs.

11. Modèle Fonctionnel et Service Abstrait de MAP

La figure 5-19 montre le modèle fonctionnel de MAP, le service est fourni via quatre portes, i.e. *Base*, *Contexte*, *Historique* et *Administration*. Un FE communique avec MAP via une ou plusieurs de ces portes.

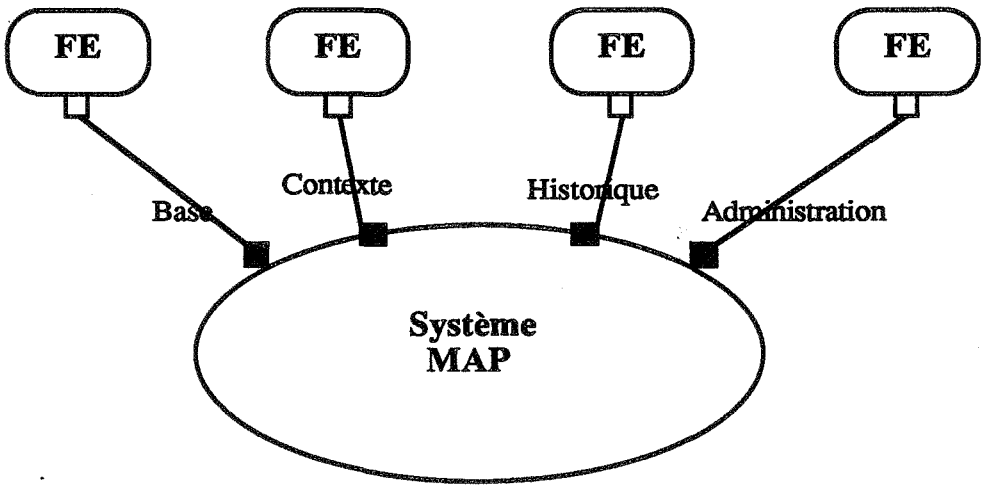


Figure 5-19. Le modèle fonctionnel de MAP

La porte **Base** fournit les opérations communes (BIND, UNBIND, ERROR) plus les opérations pour manipuler la base d'information. Les BIND et UNBIND permettent de se connecter et déconnecter avec un client à travers la même porte), l'ERROR retourne les erreurs en cas d'exception. On donne ici la liste des opérations pour résumer:

- Stocker
- Effacer

- V_Ajouter
- V_Retirer
- CP_Ajouter
- CP_Retirer
- C_Remplacer
- Rechercher
- Résumer
- Lire
- L_Get
- U_Get
- Context_Switch

Un FE doit indiquer le contexte de travail via lequel la base sera manipulée pendant l'établissement de la connexion (BIND). Le contexte de travail peut être dynamiquement changé par des exécutions de l'opération Contexte_Switch.

La porte **Contexte** fournit les opérations communes (BIND, UNBIND, ERROR) plus les opérations pour manipuler la base de contextes, on donne ici la liste des opérations:

- Créer un nouveau contexte
- Ajouter le lien SousContexteDe entre deux contextes.
- Couper le lien SousContexteDe entre deux contextes.
- Déclarer un nouvel attribut
- Supprimer un attribut
- Changer le Contrôleur
- Changer l'ensemble d'opérations autorisées.
- Remplacer les données utilisateur.

La porte **Historique** fournit les opérations communes (BIND, UNBIND, ERROR) plus les opérations pour accéder l'historique, on donne ici la liste des opérations:

- Demander l'enregistrement d'une version d'un item
- Supprimer les événements ou versions identifiés par une liste d'EID
- Lire un événement identifié par son EID
- Obtenir l'état d'un item à un instant donné
- Sélectionner un ensemble d'événements

La porte **Administration** fournit des opérations pour manipuler les définitions des attributs, les définitions des classes d'items, les contrôleurs d'accès; et quelques opérations d'administration. On donne ici la liste des opérations:

- Définir un nouvel attribut
- Changer la définition d'un attribut
- Lire les définitions des attributs

- Définir une nouvelle classe
- Supprimer une classe
- Lire une classe

- Suivre un type d'action à partir d'une classe
- Créer un Contrôleur
- Modifier un Contrôleur
- Supprimer un Contrôleur
- Lire un Contrôleur
- Lister tous les contrôleurs
- Vérifier
- Lister tous les contextes auquels un item donné appartient directement ou indirectement.
- Lister tous les items oubliés.

Chapitre 6
Réalisation d'un Prototype

1. Introduction

Ce chapitre présente la réalisation d'un prototype de MAP. Cette réalisation a pour but de valider les concepts de MAP et d'étudier la méthodologie de la réalisation.

Bien que MAP soit un outil *général* pour supporter différentes applications MHS et GC, i.e. pour réaliser les archiveurs de ces applications, une réalisation efficace dépend des applications visées. En particulier, les propriétés particulières des applications à supporter permettent de simplifier l'implantation. Par exemple, pour certains stockages comme les boîtes aux lettres personnelles, le contenu d'un item est généralement statique sauf ceux représentant les collections de messages (avec le contenu dynamique et non ordonné), cela nous permet d'optimiser l'implantation. Un autre exemple est le nombre maximum et le nombre moyen des items dans la base d'information. La réalisation présentée est plutôt générale, la taille de la base d'information à supporter est quelques dizaines de milliers d'items.

2. Outils Existants

Dans cette section, nous étudions brièvement les outils existants susceptibles d'être intéressants pour la réalisation de MAP, peuvent-ils être utilisés directement pour stocker et manipuler les objets de MAP ? ou pour gérer une partie de MAP ?

Les outils que nous étudions sont les *Bases de Données Relationnelles*, les *Bases de Données Orientées-objet*, et les *Systèmes Hypertextes*.

2.1. Bases de Données Relationnelles

Les bases de données relationnelles gèrent des *tables* d'*articles*, chacun composé d'un certain nombre de *champs* spécifiés dans la définition de la table. On peut stocker, manipuler et rechercher (d'après les champs) des articles dans les tables. Les exemples des systèmes commerciaux sont Oracle™, DBase™, Informix™, etc. Les limites des BDR empêchent de les utiliser pour stocker et manipuler des items:

- Les BDR ne peuvent stocker des données immenses ni variables, comme le contenu d'un item.
- Certaines BDR n'acceptent pas les données brutes qui peuvent être présentes dans les items.
- La plupart des BDR commerciales ne supportent pas de champs multi-valeurs dont MAP a besoin.
- Les BDR ne supportent pas directement la recherche récursive, cela rend les recherches de ce type trop inefficaces pour pouvoir être acceptées par les utilisateurs.

L'avantage des BDR est qu'il existe des produits commerciaux . Ils organisent d'une façon optimale des données et la recherche est souvent rapide. Donc on peut utiliser les BDR pour stocker et indexer des attributs prédéfinis de MAP.

2.2. Bases de Données Orientées-Objet

Les bases de données orientées-objet (BDOO) [Bancilhon 88a] sont un nouveau venu dans le monde des bases de données. Elles sont plus récentes que les BDR, en particulier, il n'existe pas des produits commerciaux largement acceptés par les utilisateurs, et il n'y a pas de langage de manipulation et de recherche de données standard comme SQL pour les BDR.

Les idées des BDOO viennent des *objets persistants* des langages de programmation orientés-objet (par exemple, Smalltalk™ [Glodberg 84]). Le modèle de données des BOOD est orienté-objet. Les données sont représentées comme des *objets*, et les objets sont caractérisés par leurs *classes*. Une classe définit le type de données de ses *instances* (i.e. les objets appartenant à la classe), et les procédures (i.e. *méthodes*) pour manipuler les instances. Une classe peut avoir des sous-classes et/ou super-classe(s), une sous-classe *hérite* des propriétés de la super-classe, en particulier les méthodes. A chaque instant, chaque objet a un ensemble d'attributs et un ensemble de méthodes permettant de le manipuler. Les attributs peuvent avoir comme valeurs des références aux autres objets, donc les objets peuvent être liés. La communication avec un objet se fait par l'envoi d'un *message*: l'objet "appelle" alors une méthode choisie selon le sélecteur contenu dans le message. Les BDOO permettent de définir dynamiquement de nouvelles classes, de stocker les objets/instances des classes, et de manipuler les instances en appelant les méthodes définies dans les classes.

Le modèle orienté-objet permet de modéliser les données plus complexes que les simples "tables" des BDR, donc permet de stocker de façon plus naturelle des objets du monde réel, ayant entre eux des relations diverses. Il permet de plus d'associer des procédures spécifiques à ces objets.

Donc les BDOO paraissent très intéressantes pour la réalisation de MAP puisque MAP contient des objets classés et liés. Elles permettent ensuite de réaliser très facilement les fonctionnalités spécifiques de FE, en créant simplement des sous-classes et en ajoutant des nouvelles méthodes. Cependant, on doit d'abord résoudre certains problèmes avant de les utiliser confortablement.

- Les BDOO doivent pouvoir stocker des données immenses et non-structurées, ce qui n'est pas le cas pour la plupart des BDOO existantes.
- Les BDOO doivent supporter des objets *incomplets* où certains attributs sont absents, elles doivent aussi supporter des attributs optionnels. Cela n'est pas encore le cas, à notre connaissance.
- Les BDOO doivent fournir des possibilités de recherche sophistiquée.
- Un langage standard de définition et de manipulation de données doit être défini afin de pouvoir porter facilement les applications.

On doit noter qu'à cause de l'indisponibilité des produits de BDOO, nos informations sont principalement basées sur les articles de recherches ([Maier 86], [Kim 88], [Bancilhon 88b]), qui sont généralement en avance par rapport aux produits commerciaux.

2.3. Systèmes Hypertextes

Le principe de **hypertexte** est très simple ([Conklin 87]): les fenêtres sur l'écran sont associées aux objets dans une base de données, et les liens sémantiques entre ces objets sont visualisés sur l'écran (graphiquement) et sont décrits dans la base de données.

Un système hypertexte stocke des objets (appelés souvent *nodes*) et les *liens* entre eux. Il fournit une interface graphique et uniforme pour manipuler les objets et les liens stockés. Chaque objet est un "texte", un graphe, une image, un son, etc. Les utilisateurs d'un système hypertexte peuvent stocker les objets et les lier aux autres; et suivre les liens pour retrouver et lire les objets existants. Le système hypertexte permet donc de représenter et d'organiser les "textes" en un graphe, et fournit les outils pour lire et ajouter des "textes" dans le graphe. Les exemples des systèmes hypertextes sont HyperCard™, NoteCard™, etc.

Dans certains systèmes (e.g., HAM[Campbell]), les *contextes* sont introduits pour regrouper les objets concernant le même thème. Il peut y avoir des liens entre les objets des différents contextes.

L'idée d'*hypertexte* a beaucoup influencer la conception de MAP. Cependant, bien qu'il soit possible de représenter/stocker les contenus des items et les liens entre items en utilisant un système hypertexte, les attributs ne sont généralement pas supportés par le système.

Du point de vue fonctionnel, MAP possède les caractéristiques des trois types de systèmes que nous avons décris ci-dessus, aucun outil ne peut lui seul satisfaire le besoin de MAP.

3. Environnement de la Réalisation

Ce projet a été réalisé dans le département Informatique Appliquée de l'Ecole des Mines de St-Etienne. Les environnements, matériel et logiciel, sont:

- Matériel: Machines Unix (HP9000/318, SUN3/xx, etc) reliées par un réseau Ethernet.
- Logiciel: les logiciels et outils de base d'Unix, la bibliothèque de codage et décodage AbsSynt™ de la société Sync qui permet de coder et décoder des données ASN.1.

Chaque instance de MAP (qui gère une base d'information) se situe dans une des machines qui utilise le système de fichiers de la machine locale, elle peut aussi utiliser les fichiers distants via NFS™. L'instance de MAP se présente sous forme d'un processus *serveur*. Le(s) FE du MAP peu(ven)t être sur la même machine ou sur d'autres machines du réseau local. Un FE communique avec MAP suivant le protocole *Psam* qui est codé en un format interne ou ASN.1. Dans le cas où le FE et MAP se situent dans des machines hétérogènes, les outils de communication classiques d'Unix TCP/IP sont utilisés pour établir la connexion, envoyer et recevoir des données du protocole. La Figure 6-1 donne la configuration.

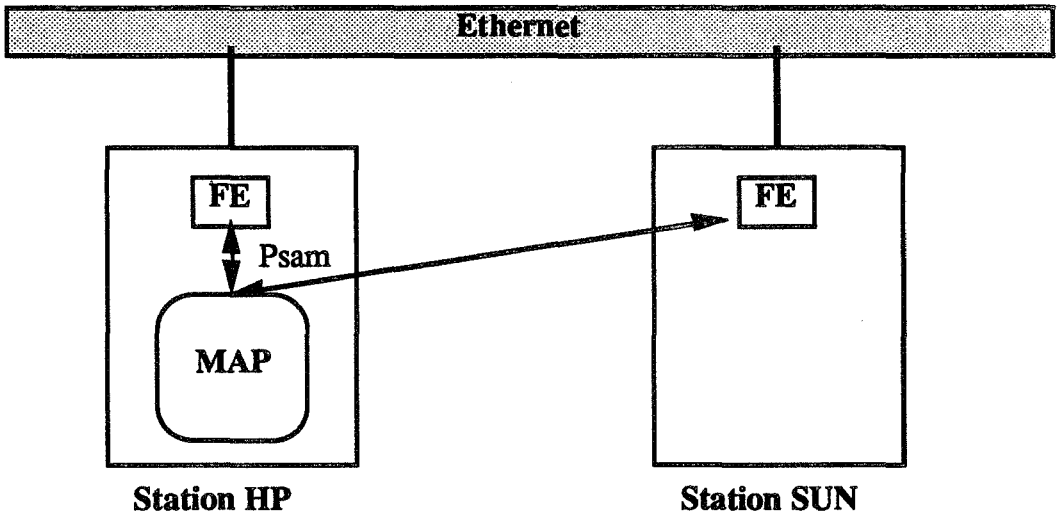


Figure 6-1. L'environnement de MAP

4. Décomposition du Système en Modules

Dans cette section, nous présentons dans ses grandes lignes la décomposition du système en modules, les fonctionnalités et l'organisation de chaque module.

La décomposition en modules suit le principe de la conception orientée-objet (cf. [Mayer 88]), chaque module correspond à une type/classe d'objets. La figure 6-2 illustre la décomposition de MAP (le module au bout d'une flèche utilise les fonctions du module pointé par la flèche):

- **MCom** (Module de Communication et d'Interface) est utilisé comme intermédiaire entre les FE et les autres modules, il gére la communication avec les FE.
- **MDéf** (Module de Définitions des Attributs) gère la base de définitions d'attributs de MAP.
- **MCon** (Module de Contrôleurs) s'occupe de la base de Contrôleurs d'accès.
- **MCont** (Module de Contextes) gère l'ensemble des contextes de MAP.
- **MHis** (Module de l'Historique) est le module qui s'occupe de l'historique de MAP.
- **MCla** (Module de Classes) s'occupe de la hiérarchie des définitions des classes et des relations entre classes.
- **MInf** (Module de la base d'Information) gère la base d'information de MAP.

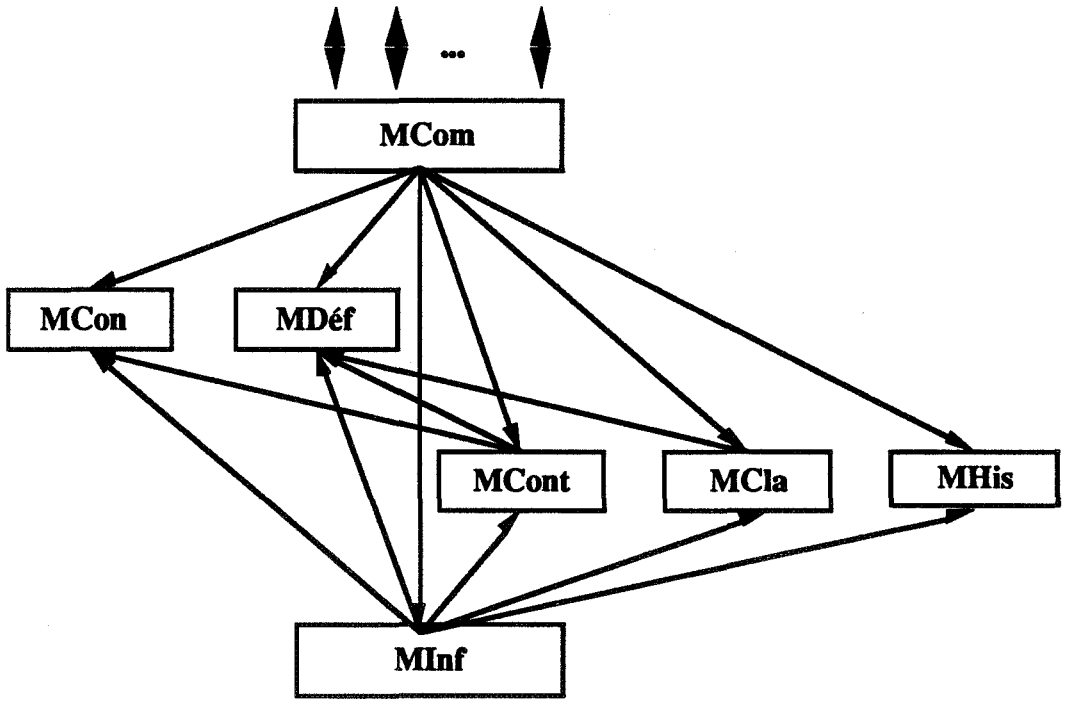


Figure 6-2. Les Modules de MAP

4.1. Module de Communication et d'Interface

MCom est le module qui réalise l'interface avec les FE et fournit le service suivant le protocole *Psam*. Ses fonctions sont:

- Gestion d'une ou plusieurs liaisons avec les FE via les *Unix Sockets* ou d'autres moyens. Suivant un paramètre de l'établissement de communication, il décide le format/codage des données échangées.
- Multiplexage des requêtes reçues via les liaisons au module approprié, en appelant la fonction appropriée *séquentiellement*. Ensuite envoi de la réponse. On doit noter que chaque requête correspond à une seule fonction fournie par un module.
- Codage et Décodage des données du protocole échangées, depuis ASN.1 au format interne, et vice versa. Le paramètre de l'établissement de connexion précise le format de données pendant l'échange qui est soit ASN.1 (dans ce cas, il faut coder et décoder), soit le format interne (qui n'a pas besoin de codage et de décodage, mais simplement de séquencer les données structurées).

Les autres modules stockent et gèrent une classe de données. Chaque module fournit des fonctions primitives pour manipuler les données stockées, et les fonctions correspondantes aux requêtes des FE. Un module peut utiliser les fonctions des autres modules. Leurs relations d'utilisation sont montrées dans la figure 6-2.

4.2. Module des Définitions d'Attributs

Le module **MDéf** stocke et gère les définitions d'attributs de MAP, il fournit les fonctions pour *MCom, MCla, MInf* et *MCont.* Les fonctionnalités sont:

- Ajouter la définition d'un nouvel attribut.
- Lire une ou plusieurs définitions.
- Modifier la définition d'un attribut.

Après l'initialisation de MAP, le module contient les définitions de tous les attributs prédéfinis.
La réalisation du module est direct, nous ne précisons pas ici les détails.

4.3. Modules de Contextes

Le module **MCon** est la classe des Contrôleurs qui stocke et gère la base de Contrôleurs de MAP. Il fournit les fonctions pour *MCom, MInf* et *MCont.* Les fonctionnalités sont:

- Stocker un nouveau Contrôleur.
- Supprimer un Contrôleur.
- Lire un ou plusieurs (tous) Contrôleurs.
- Changer l'ACL d'un Contrôleur existant.
- Vérifier les droits d'accès d'un utilisateur auprès d'un Contrôleur.

L'organisation des données et l'implantation des fonctions sont relativement simples et directes, les détails sont omis dans cette thèse.

4.3. Module de l'Historique

Le module MHis gère l'historique de MAP, i.e. l'ensemble des événements et des versions. Les modules *MCom* et *MInf* accèdent ce module.

4.4. Module de Contrôleurs

Le module **MCont** gère l'ensemble des contextes de MAP et leurs relations, il fournit les primitives pour manipuler le contexte individuel et les relations. Le module MCont est accédé par *MCom* et *MInf.*

4.4.1. Organisation des Données

L'organisation *logique* des contextes de MAP est montrée dans la figure 6-3. Les contextes forment une hiérarchie, chaque noeud de la hiérarchie représente un contexte. Un noeud peut être *ouvert* ou *fermé.* Un noeud fermé contient l'information minimale et se compose de :

- Tous les champs de contextes sauf celui représentant les attributs utilisables/reconnus dans le contexte.
- Les attributs utilisables dans le contexte qui y sont déclarés directement.
- Les items directs du contexte.

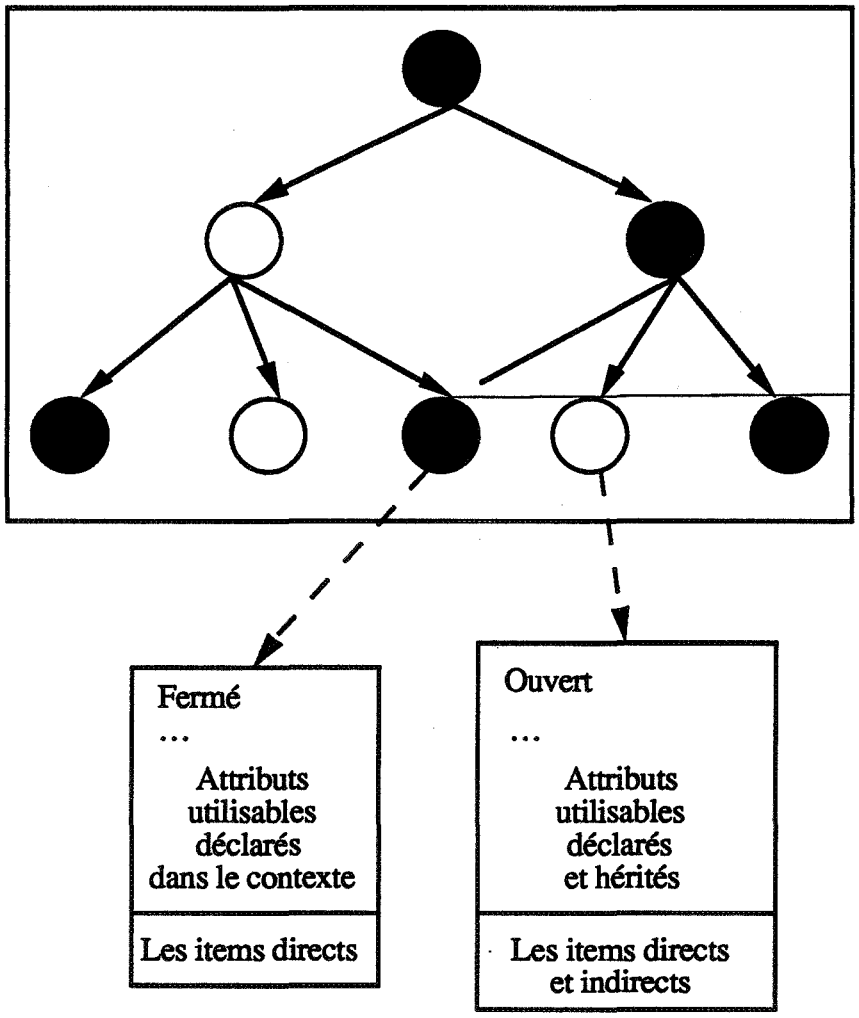


Figure 6-3. Organisation Logique

Un noeud *ouvert* contient des informations complètes concernant le contexte avec l'héritage, il permet d'accéder à toutes les informations d'un contexte rapidement, il se compose de :

- Tous les champs du contexte sauf celui contenant les attributs utilisables dans le contexte.
- Les attributs utilisables dans le contexte qui y sont déclarés directement, et ceux déclarés chez ses super-contextes directs ou indirects.
- Les items directs et indirects du contexte.

A chaque instant, il y a un sous-ensemble des contextes *ouverts*. Un contexte est *ouvert* quand un FE établit une connexion à la porte *Base* pour travailler dans le contexte ou après un basculement de contexte (par l'opération Context-Switch). Le contexte ouvert permet d'accélérer les accès aux informations du contexte. Le nombre maximum de contextes ouverts est un paramètre de l'implantation dépendant de la mémoire centrale disponible.

L'ajout d'un nouvel attribut utilisable dans un contexte propage l'ajout de l'attribut *hérité* dans tous les sous-contextes ouverts, s'il n'y est pas déjà; les sous-contextes *fermés* ne sont pas touchés.

La suppression d'un attribut utilisable dans un contexte supprime l'attribut de tous les sous-contextes ouverts qui ont hérité de cet attribut utilisable, on n'a toujours pas besoin de modifier les sous-contextes *fermés*.

L'association d'un nouvel item dans un contexte propage l'ajout uniquement aux super-contextes ouverts dont l'item est indirect.

La figure 6-4 montre notre organisation *physique* de données du module **MCont**; l'idée est de garder complètement les contextes ouverts dans la mémoire centrale; seuls le nom et l'adresse physique des contextes fermés, et les relations entre les contextes, sont stockés dans la mémoire centrale.

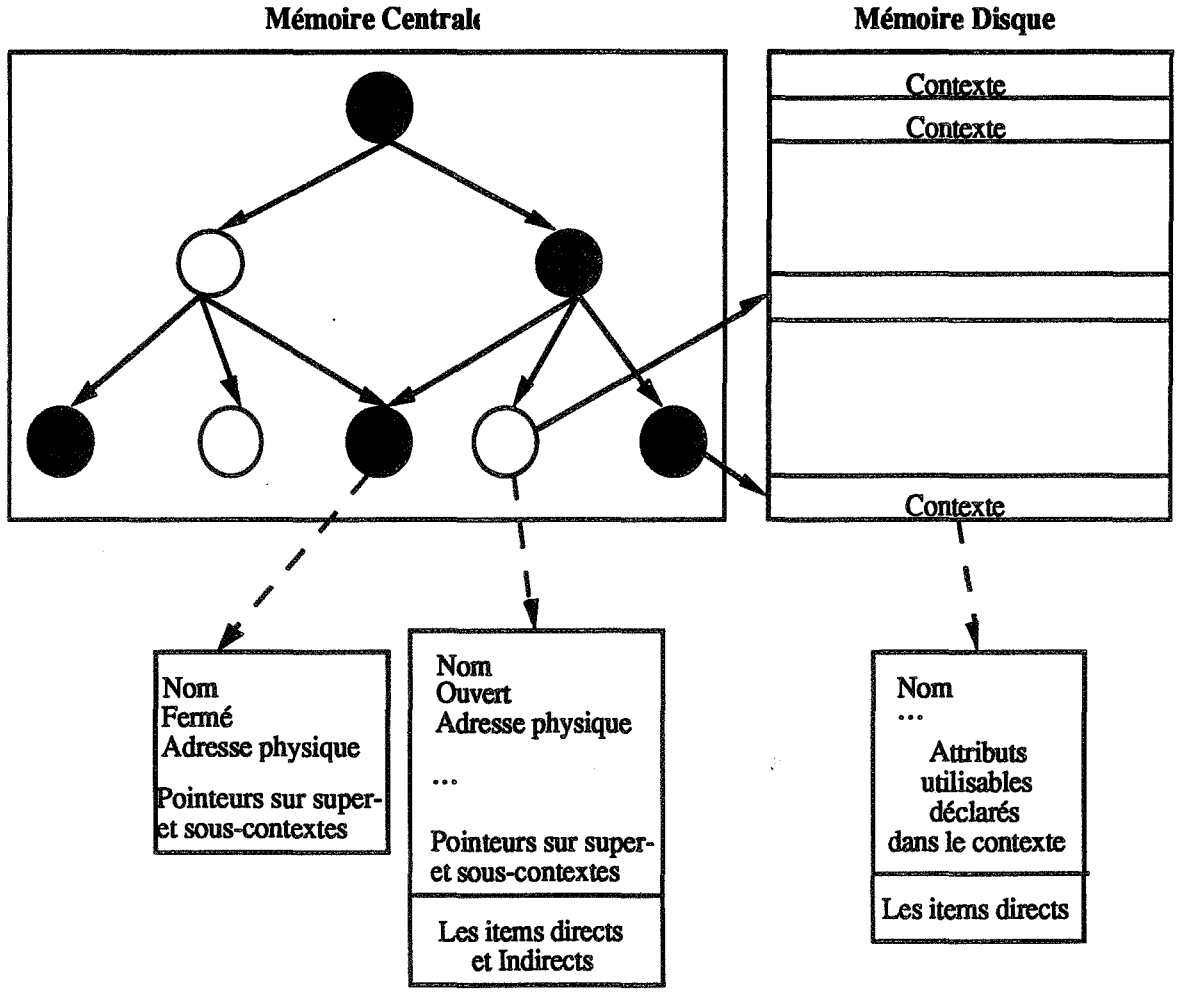


Figure 6-4. Organisation Physique

4.4.2. Fonctions

Les fonctions fournies par le module **MCont** sont:

- *Créer* un nouveau contexte.
- *Lier* un contexte comme sous-contexte d'un autre.
- *Couper* le lien entre un contexte et son super-contexte.

- *Lire* les attributs du contexte.
- *Lire* les attributs utilisables dans un contexte avec héritage.
- *Lire* les items directs et/ou indirects d'un contexte.
- *Modifier* un attribut d'un contexte.
- *Associer* un item à un contexte.
- *Dissocier* un item d'un contexte.
- *Ouvrir* un contexte.
- *Fermer* un contexte.

Remarque: quand le nombre de contextes est limité par la mémoire centrale, on pourra envisager la technique de la *mémoire virtuelle.*

4.5. Module de Classes

MCla est le module qui stocke et gère les définitions des classes et les relations entre ces classes, i.e. l'héritage. En fait, les fonctionnalités de classes sont réalisées par deux modules: le module en question et un sous-module du **MInf.** Le module **MCla** s'occupe uniquement des classes et de leurs relations, le sous-module du **MInf** gère les relations entre les items et les classes: est-ce qu'un item est une instance légale d'une classe? quelles sont les instances (directes et/ou indirectes) d'une classe donnée? etc. **MCla** fournit les outils/fonctions nécessaires au sous-module.

4.6. Module de la base d'Information

MInf est le module principal de MAP, il gère la base d'information et utilise les autres modules pour stocker et obtenir des informations (i.e. appelle les fonctions des autres modules). Ce module fournit toutes les fonctions pour réaliser les opérations de la porte *Base.*

Le module MInf est composé de plusieurs sous-modules (cf. figure 6-6.):

- **MInt** (Module d'Interface): c'est le seul module visible de l'extérieur, il fournit les fonctions du module *MInf.* Il utilise les fonctions des autres sous-modules pour réaliser ses fonctions.

- **MSto** (Module de Stockage): ce module gère l'archive physique des items et permet de les modifier.

- **MIns** (Module d'Instances): ce sous-module s'occupe des relations entre les items et les classes d'items, il accède au module **MCla** pour obtenir les informations nécessaires pour effectuer ces opérations.

- Les autres modules (Module de liens, ...) gèrent des index des items pour la recherche. En particulier, **MNom** (Module de Noms) gère la correspondance entre les noms externes et les noms internes, et gère la numérotation des items, i.e. la génération des noms internes des items. **MLie** (Module de Liens) gère tous les liens (y compris les *membership*) et permet la recherche et la lecture. **MUti** (Module d'Utilisateur) gère les index sur les relations item-utilisateur. De plus, il y a un ensemble de modules qu'on

appelle généralement les modules d'**Index**, ils gèrent chacun des index pour un sous-ensemble des attributs prédéfinis. Ces modules peuvent être réalisés par les bases de données relationnelles. Cependant, à cause de la lourdeur de l'utilisation et de l'indisponibilité de BDR, on utilise simplement les techniques classiques sans BDR. Il faut noter qu'il sera facile de remplacer la réalisation d'un module, puisque chaque module ne manipule qu'un type de données, qui ne sont manipulées directement par aucun autre module.

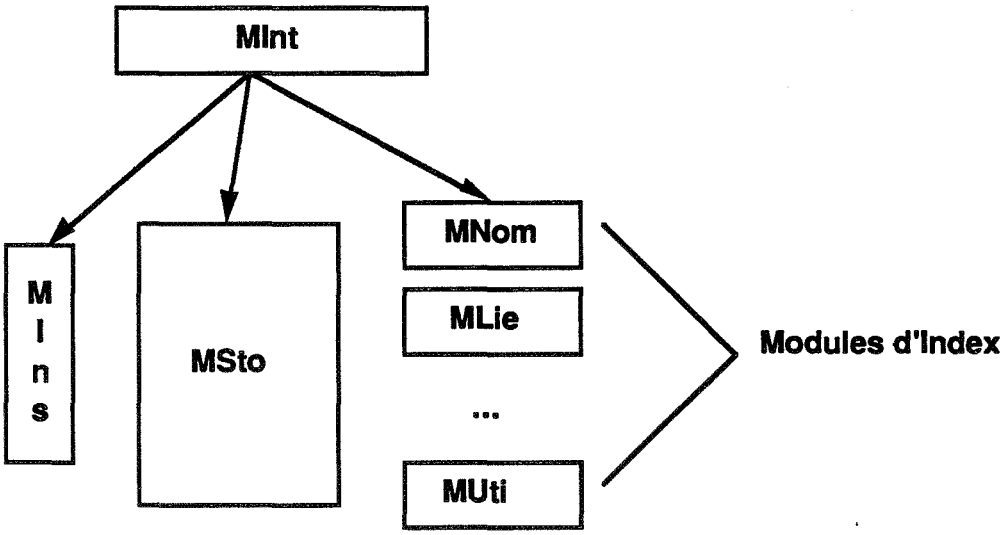


Figure 6-6. Les Sous-Modules de MInf

En effet, on peut envisager une autre décomposition de telle sorte que les items ne soient pas stockés entièrement dans un seul module, mais également dans les modules d'index. L'avantage de cette autre solution est qu'il n'y a pas de redondance et que les modifications sont plus simples et plus rapides. Cependant, l'inconvénient est que les lectures sont moins rapides puisqu'il faut chercher dans tous les index pour former un item complet. Du fait que les lectures sont beaucoup plus fréquentes que les modifications pour les archiveurs de messages, et que la durée de la lecture est beaucoup plus critique que les autres, on conclut que la solution choisie est meilleure.

Nous détaillerons dans la section suivante le module *MSto*.

4.6.1. Module de Stochage Physique

MSto organise et stocke les items (la figure 6-7).

Un index (les noms internes -> les adresses physiques des items) permet au **MSto** de trouver rapidement un item. Chaque item est décomposé en 5 parties:

- les attributs prédéfinis: ils sont peu modifiés, et ont souvent une taille fixée pour un item donné.
- les attributs spécifiques: ils dépendent de l'application, et peuvent être modifiés plus fréquemment que les attributs prédéfinis.

- les liens: ils contiennent tous les liens sortants de l'attribut, et sont modifiés plus fréquemment que les autres parties.
- les utilisateurs: ils contiennent toutes les relations item-utilisateur.
- le contenu: il est généralement statique, sauf pour les collections (dynamique, non-ordonné et le seul type de CP possible est RCP). Le contenu d'une collection est traité de même façon que les liens.

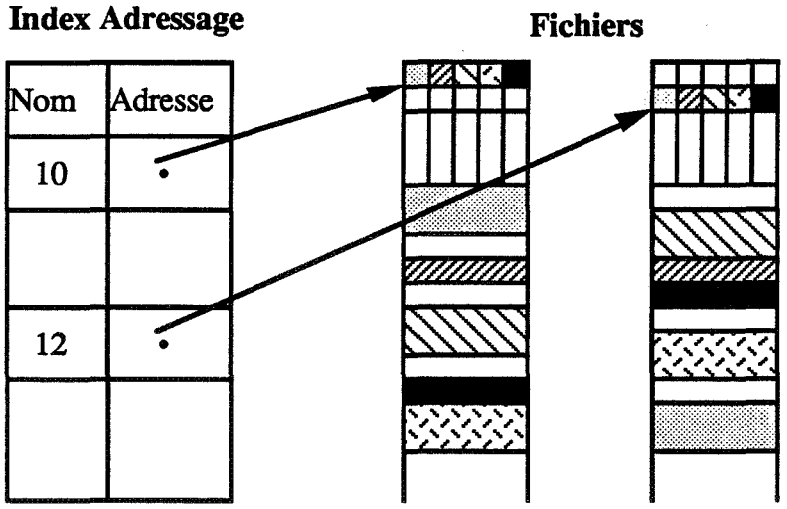


Figure 6-7. Stockages Physiques

Chaque partie est une unité de stockage qui peut être complètement statique, par exemple, la partie "contenu" est souvent statique pour des messages. Cela nous permet de mieux traiter les modifications. L'adresse d'un item est indirecte, elle pointe sur une article de 5 champs, chacun étant composé de l'adresse absolue de la partie représentée et de la longueur de la partie. Les espaces disques sont représentés par un ensemble de fichiers, chacun contient au maximum 255 items; et comme 255 fichiers sont possibles, cela nous donne la capacité de stockage: 255x255=65025 items, sous la contrainte de la mémoire réelle disponible de la machine. Chaque item est stocké dans un seul fichier dont chaque partie est contiguë dans l'espace du fichier. Quand une partie dépasse la taille réservée à cause d'une modification, elle est mise à la fin du fichier. Un compactage est fourni par le module qui doit être lancé périodiquement par l'administrateur.

5. Conclusion

Les aspects de la fiabilité et de la récupération après des incidents (*crashs*) suivent les techniques classiques de bases de données (cf. [Kohler 81], [Davidson 85], [Svobodova 84]). Cela consiste à maintenir les différents journaux et à utiliser des mémoires redondantes. Nous omettons ici les descriptions détaillées.

Une version complète de MAP a été réalisée et testée sur HP9000/318, HP9000/825 et SUN3/160. Elle comprend actuellement 23 modules et 18 000 lignes en C, l'exécutable fait 1,3 méga octets sur HP9000/318. Une nouvelle version plus compacte est en cours de réalisation. A cause du manque des FE (dont certains sont en construction), nous n'avons pas pu la tester dans un environnement réel d'utilisation.

Notre réalisation utilise la méthode orientée-objet pour décomposer le système, ceci nous donne beaucoup d'avantages au point de vue du génie logiciel. Par exemple, le remplacement de l'implantation d'un module ne change pas les autres qui l'utilise. Il est aussi très facile d'insérer de nouveaux modules fonctionnels qui réalisent des fonctionnalités spécifiques au-dessus des modules existants.

Une autre façon de représenter MAP qui nous paraît très intéressante, est de fournir une bibliothèque (comme les utilitaires dbm/ndbm de Unix) de fonctions qui donne l'interface de programmation pour les FE. Cette méthode est plus efficace dans le cas où les FE et MAP se situent dans la même machine, de plus, il est plus facile de réutiliser les modules de MAP par les FE. Mais dans ce cas-là, la gestion d'accès concurrents est à la charge des FE s'ils sont réalisés comme processus indépendants. Remarquons que dans la réalisation actuelle, les accès concurrents sont traités par le module **MCom** en exécutant simplement les requêtes une par une.

Troisième Partie

Exemples

Introduction

Chapitre 7

Répartition: le modèle et un exemple

Chapitre 8

Une Application dans l'Environnement Bureautique
POMS : Personal Office Message Store

Chapitre 9

Un Système de Vote en Mode Messagerie

Introduction

SAM peut être adapté à chaque application par la construction d'un ou plusieurs Front-Ends particuliers. Cela consiste à représenter les objets de l'application par des objets de MAP, et utiliser des mécanismes fournis par MAP pour réaliser des fonctions dont a besoin l'application. En particulier :

- Représenter les objets d'information de l'application par des items.
- Regrouper des objets d'information en unités de distribution et d'administration par des contextes.
- Utiliser des fonctions de "Base" pour rechercher et manipuler des objets d'information.
- Utiliser le mécanisme de contrôle d'accès pour contrôler des accès aux objets d'information.
- Utiliser le mécanisme d'historique et d'autres supports de distribution pour distribuer et synchroniser des informations.

Dans cette partie, nous allons présenter quelques exemples des utilisations de MAP, qui permettent de montrer la puissance et la flexibilité de notre conception. Le chapitre 7 étudie un exemple de conférence répartie pour montrer la distribution et la synchronisation d'information. Le chapitre 8 concerne la conception et la réalisation d'un archiveur personnel des messages dans un environnement bureautique et montre comment organiser et représenter des messages en utilisant MAP. Le chapitre 9 donne un exemple de la communication de groupe et son utilisation de MAP.

Chapitre 7
Répartition

Ce chapitre présente nos études de la distribution d'informations, et montre comment MAP supporte la distribution d'informations en donnant un exemple d'une conférence répartie, basée sur CRMM (Brun 87]).

1. Introduction

Les applications MHS et GC sont souvent distribuées, les agents et les communicateurs d'une application MHS et GC sont répartis, de même pour les informations, en particulier les messages. On a donc souvent besoin de **répartir** ou **distribuer** des informations dans plusieurs archiveurs de messages. La répartition des informations a les avantages suivants:

- Un coût de communication plus faible. La répartition des informations diminue le coût de communication de longue distance qui est souvent énorme par rapport au coût de communication locale et de ressource d'ordinateur. Par exemple, dupliquer des messages dans des endroits géographiquement distants permet aux utilisateurs d'effectuer des accès locaux.
- Une meilleure performance. L'accès local est généralement plus rapide que l'accès distant.
- Une plus grande sécurité. Dupliquer des informations (messages) réduit le risque de perte.
- Plus économique. Il est possible que chaque site ne garde que les informations intéressantes, cela économise les mémoires utilisées vis-à-vis du site.

Les informations réparties ont des relations différentes entre eux, ces relations permettent de choisir les stratégies ou méthodes de distribution/répartition d'informations:

- **Duplication**: les objets d'information peuvent être dupliquées dans plusieurs archiveurs. Un problème ici est de gérer la cohérence entre les instances d'un objet d'information dupliqué, lorsqu'une instance est modifiée.
- **Référence**: Dans un endroit, on ne garde que les références de certains objets d'information: elles indiquent les localisations (archiveurs) où on peut trouver ces objets d'information.
- **Chaining** et **Multi-casting**: les informations sont distribuées dans plusieurs archiveurs sans que certains d'entre eux sachent les localisations d'informations absentes. Dans ce cas, pour trouver des informations absentes dans un archiveur, il faut soit demander à tous les autres s'ils possèdent ces informations (*multi-casting*); soit demander à un voisin choisi qui peut, à son tour, demander à un autre (*Chaining*). L'algorithme de *chaining* doit éviter la boucle infinie.

La distribution des informations peut être un mélange de ces trois stratégies, par exemple, dupliquer certains objets d'information intéressants, garder des références pour des objets de grande taille, et utiliser les méthodes de *chaining* ou *multi-casting* pour trouver d'autres objets.

Après avoir distribuer les informations dans un ensemble d'archiveurs, il faut ensuite **synchroniser** ces archiveurs afin de garantir la cohérence entre eux, i.e. entre leurs bases

d'information, lorsque les informations sont modifiées dans un sous-ensemble d'archiveurs. Selon l'application et ses stratégies de distribution, les différentes stratégies de synchronisation peuvent intervenir. Les stratégies de synchronisation décident quand et comment synchroniser les archiveurs.

Quelques aspects spéciaux d'archiveurs de messages qui peuvent avoir des influences sur les stratégies de distribution et de synchronisation d'informations sont:

- La partie globale d'un objet d'information (en particulier, message) est souvent statique, des modifications sont très peu fréquentes. Seuls les ajouts de nouveaux objets sont fréquents parmi les opérations de modifications. En plus, les modifications globales sont souvent contrôlées par un mécanisme centralisé (par exemple, établissement des nouveaux liens dans CRMM). Cela nous permet d'avoir des stratégies *lourdes* pour traiter les modifications sans dégradation importante de performance. Il implique aussi qu'il est possible de former un objet dynamiquement à partir de son état original et ses modifications sans trop perdre en performance.
- Le besoin de cohérence est relativement faible dans un environnement messagerie à cause de la nature de la transmission d'information. Le délai de transmission est assez important et très différent selon les communicateurs. De plus un message peut être perdu pendant sa transmission. Il est donc normal qu'à un moment donné, deux archiveurs dupliqués possèdent des informations différentes ou incohérentes. Un autre point concernant la cohérence est que différents archiveurs sont souvent administrés par des unités différentes qui peuvent eux-mêmes être incohérentes. En conclusion, il est tout à fait acceptable que les "mêmes" informations dans différents archiveurs distribués soient dans des état *légèrement* incohérents.

1.1. Supports Fournis par MAP

Avant de commencer à présenter le modèle et l'exemple, nous rappelons ici les supports fournis par MAP vis-à-vis de la répartition et de la distribution d'informations.

(1) Nom externe: le nom externe d'un item permet d'identifier l'item indépendamment de l'archiveur où il est stocké. Il facilite ainsi la distribution de l'objet et son identification.

(2) Contextes: des contextes fournissent des unités de distributions. On peut distribuer des contextes dans différents archiveurs, et gèrer la cohérences entre ces contextes distribués.

(3) Différents niveaux d'objets: des objets incomplets permettent de ne garder qu'une partie d'un objet global dans un archiveur et la référence à un autre archiveur, où on peut trouver une instance complète. Un objet incomplet de niveau 2 permet de garder des **références** aux instances complètes (Cf. Chapitre 5 §5.1.7).

(4) Attributs de distribution: les attributs de distribution mémorisent des paramètres pour différentes méthodes de distribution, la liste d'attributs est donnée comme suite:

- IncompletLevel
- IsMaster: indique si l'instance est maître.

- Master: sinon quel est le maître. Ces deux attributs sont utilisés par la stratégie de distribution *maître-esclave.*
- Copies: les localisations des autres instances du même objet, complets et/ou incomplets.
- UpToDate: indique si l'instance est à jour. Ceci permet une stratégie de synchronisation *à la demande.*

(5) Mécanisme d'historique: il conserve des opérations et des versions, et ainsi facilite la propagation de modifications (en événements ou en versions), ou re-exécutions de modifications avec de nouvelles modifications insérées. Par exemple, une modification lancée avant une autre peut arriver après la deuxième qui est déjà exécutée, l'arrivée de la première peut causer des re-exécutions des opérations sur l'objet. En particulier, le mécanisme permet de savoir si un item a été changé depuis un moment donné, et les modifications effectuées depuis le moment.

2. Modèle de Répartition

On donne ici un exemple des modèles de distribution possible au-dessus de MAP, le modèle n'est pas déclaré comme universel ou général. Cependant, nous pensons que c'est le modèle le plus adapté à un environnement messagerie distribué décentralisé dont les utilisateurs se trouvent dans un réseau de longue distance et dont le moyen de transfert est l'échange de messages.

Le modèle permet de répartir une base d'information contenant un ensemble d'objets d'information dans plusieurs archiveurs de façon hiérarchique, chaque archiveur contient un sous-ensemble d'objets d'information. Le modèle décrit aussi des stratégies de synchronisation possibles pour synchroniser ou gérer la cohérence entre ces archiveurs. Ce modèle a les caractéristiques suivants:

- Le modèle est hiérarchique, les archiveurs forment un arbre strict. Chaque noeud représente un archiveur.
- La racine contient la base d'information de l'application. Chaque noeud dans l'arbre contient une partie de la base d'information: un sous-ensemble des objets d'information de la base. Un noeud (sauf la racine) contient un sous-ensemble d'objets d'information contenus dans le noeud père. Si chaque noeud est un archiveur SAM, chaque noeud contient une sous-hiérarchie de la hiérarchie des contextes de son noeud père.
- On appelle le père le *site de référence* de ses fils.
- La structure hiérarchique est très bien adaptée à la distribution d'information dans un environnement messagerie décentralisé. L'avantage de cette structure est qu'un noeud est totalement responsable de la distribution d'information à ses fils. Un noeud n'a donc pas besoin de connaître ni son grand-père ni ses petits-fils. D'un point de vue administratif, cette structure permet une administration simple et flexible. Un site peut décider de distribuer des informations à ses fils avec des stratégies qu'il choisit, sans déranger son père, ni le reste du monde.

La figure 7-1 montre le modèle hiérarchique.

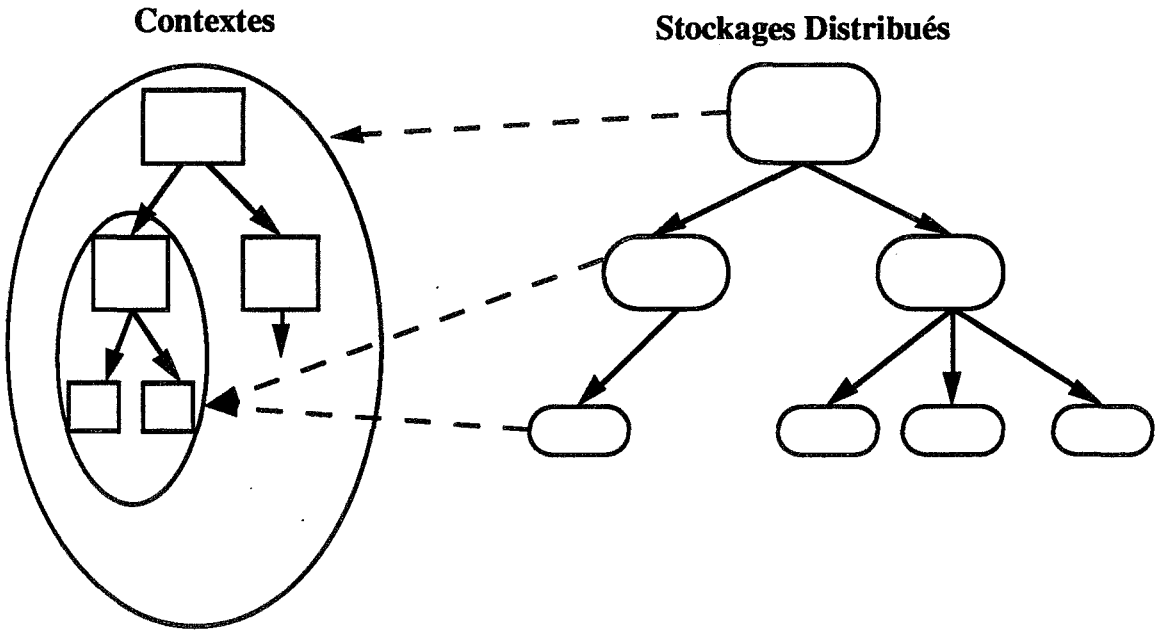


Figure 7-1. Le modèle Hiérarchique de Répartition

2.1. Relations entre des Archiveurs

Une de ces deux relations existe entre un noeud père et un noeud fils:

- *Maître-Esclave*: des informations sont distribuées du maître (le père) à l'esclave (le fils), les modifications dans l'esclave sont considérées comme locales par le maître. Souvent l'esclave n'accepte pas de modification locale.
- *Serveur-Client*: des informations sont distribuées du père (serveur) vers le fils (client), le client peut accepter des modifications locales et demander au serveur d'effectuer ces modifications. Cependant c'est le serveur qui décide si elles sont globalement valables. Les effets de ces modifications peuvent être immédiatement visibles ou non selon l'application.

Par exemple, pour un site non-membre d'une conférence, on peut imaginer un accord entre un site membre et le site non-membre tel que le site membre distribue tous les informations/modifications au site non-membre sans que ce dernier puisse influencer l'état du site membre. Dans ce cas, la relation entre le site membre et le site non-membre est *Maître-Esclave*. Un autre exemple est de distribuer des informations depuis un site de référence, et de permet au site fils d'effectuer un sous-ensemble de modifications comme dans CRMM.

2.2. Stratégies de Synchronisation

La **synchronisation** d'informations dans un ensemble d'archiveurs consiste à propager des informations entre les archiveurs afin de garder leur cohérence. La synchronisation est faite par la *propagation d'informations de changement* (*modifications*) d'un archiveur vers un autre, i.e. entre deux noeuds père et fils. La communication entre des noeuds frères est une décision locale pour

les applications spécifiques. Plus précisément, le père envoie ou propage des *modifications* vers le fils pour mettre le fils à jour, tandis que le fils envoie les *demandes de modifications* vers le père.

Les stratégies de synchronisation peuvent être différentes selon les facteurs suivants:

Quand: le moment de propager un ou des *modifications*.

- Immédiatement: quand la modification est faite, la*modification* correspondante est immédiatement propagée. Cette stratégie permet d'avoir une cohérence forte, au sens que les archiveurs sont mis à jour le plus rapidement possible. L'inconvénient est que les modifications envoyées sont nombreuses, de plus on ne peut pas choisir le meilleur moment (e.g., pendant la nuit lorsque la machine est moins chargée).
- Périodiquement: une modification individuelle n'est pas propagée immédiatement. Les modifications sont propagées périodiquement, par exemple tous les jours à minuit, ou toutes les heures, etc. Bien que cette stratégie gagne en flexibilité, elle perd un peu en vitesse de distribution.
- Au besoin: Les modifications sont propagées quand l'administrateur décide.
- A la demande: Les modifications ne sont propagées que lors de la demande du receveur.

Il est possible d'avoir une stratégie mixte, e.g., propager les *ajouts* immédiatement, mais d'autres modifications périodiquement; ou propager les ajouts immédiatement, les noms des objets modifiés périodiquement, et enfin d'autres modifications à la demande lorsque le destinataire en a besoin.

Initiateur: Qui initialise la propagation, la source (où les modifications ont eu lieu) où le destinataire (où les modifications sont à propager)?

- Source: cette solution donne l'autonomie à la source, combinant avec la stratégie *immédiate* il permet d'une synchronisation rapide. Cependant il peut y avoir des conséquences de sur-charger la source. Cette stratégie est bien adaptée à l'administration hiérarchisée pour la propagation du *maître/serveur* vers l'*esclave/client*.
- Destinataire: cette stratégie donne plus d'autonomie au destinataire, et permet une optimisation de la propagation en propageant seulement les modifications désirées, e.g., celles qui modifient un objet d'information demandé par l'utilisateur.

Comme pour le facteur *Quand*, il est possible d'avoir une stratégie mixte telle que certaines modifications (e.g., les ajouts) soient initialisées par *maître/serveur* (la source), et certaines autres soient initialisées par *esclave/client* (le destinataire) pour la propagation de modifications du maître/serveur vers l'esclave/client.

Mode de Communication:

- Avec Connexion: Ce type de communication permet la synchronisation rapide, mais il nécessite que les deux partenaires de communication (deux machines, par exemple) soient actifs. Il assure souvent des communications fiables. Le problème principal est qu'il n'est pas toujours possible d'établir une connexion directe entre deux partenaires, surtout dans un environnement messagerie.

• Sans Connexion en Mode Messagerie: Ce type de communication est moins sûr, mais il ne requiert pas la présence simultanéedes deux partenaires, de plus il est toujours disponible dans l'environnement messagerie.

Représentation de Modifications: sous quel format est transférée une modification?

• Opération/Evénement: cette représentation est plus économique quand le nombre de modifications est relativement faible. Il permet de plus de *coordonner* les modifications des différentes sources.

• Version: chaque modification est transférée par l'envoi de l'objet d'information qu'elle modifie. L'avantage est qu'il est simple de mettre à jour les modifications dans le site destinataire. Mais l'inconvénient est que la quantité d'informations transmises est grande, et que cette méthode ne permet pas de coordonner des modifications de différentes sources.

• Application Spécifique: chaque application peut avoir sa propre représentation, par exemple, par attributs pour des modifications autres que des ajouts d'objets d'information.

Les choix de ces facteurs sont souvent liés. Par exemple, pour avoir la synchronisation immédiate du maître/serveur vers l'esclave/client, la stratégie est:

• La propagation est Immédiatement,
• Initialisé par la source, i.e. Maître/Serveur,
• Avec connexion (si possible),
• Transmises en Opérations.

Nota: pour une application spécifique, certaines optimisations sont possibles, grâce aux propriétés particulières de l'application. Nous allons voir ce point dans l'exemple suivant.

3. Un Exemple: Conférence Répartie

Dans cet exemple, nous allons présenter la réalisation d'une conférence répartie en utilisant MAP, le modèle de la conférence répartie est basé sur CRMM (cf. [Brun 87]) sauf que la répartition est une extension de celle de CRMM. Il n'est pas possible ni nécessaire de présenter le système en détails dans cette thèse, seulement les aspects qui nous intéressent, en particulier l'aspect répartition. Le lecteur peut se référer à [Brun 87] pour une description complète de CRMM. Nous ne présenterons pas non plus la réalisation en détails de l'archiveur (i.e. les FE), mais insisterons sur l'utilisation de MAP.

3.1. Présentation de la Conférence Répartie Basée sur CRMM

CRMM est une conférence en mode messagerie, dont le moyen de communication est l'échange de *messages*.

Chaque conférence CRMM contient, à chaque instant , un ensemble d'objets d'information appelés *fiches*. Chaque fiche se compose d'un identifiant, d'un en-tête, des relations avec d'autres fiches et un corps. L'en-tête contient un ensemble d'attributs, par exemple, le nom de l'auteur, le

nom de l'éditeur, la durée de validité, les mots-clés, etc. Les relations avec d'autres fiches sont représentées comme *liens* typés. Chaque lien, en plus de son type, est associé à un descripteur appelé *nature* qui indique les sémantiques du lien concernant l'ordre de lecture et de destruction.

Une liste d'opérations est fournie aux utilisateurs pour manipuler les fiches:

- Création d'une fiche.
- Ajout d'une fiche.
- Lecture d'une fiche.
- Sélection d'un ensemble de fiches d'après un critère.
- Effacement d'une fiche de telle façon que seuls les liens et le nom soient gardés.
- Destruction d'une fiche.
- Ajout de liens.
- Suppression de liens.

Les utilisateurs (membres) d'une conférence n'ont pas tous le même rôle. Pour un utilisateur donné, seul un sous-ensemble d'opérations est utilisable. Le classement d'utilisateurs d'après leurs fonctionnalités est fait par *rôles*.

Les rôles de CRMM sont (figure 7-2):

- lecteur
- auteur
- écrivain
- éditeur
- administrateur
- service de conférence

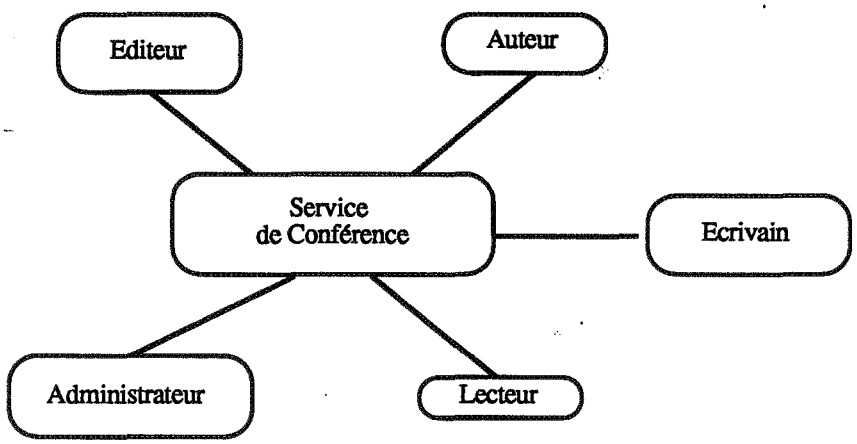


Figure 7-2. Rôles de CRMM (Source [Brun 87])

Un **service de conférence** gère la base de fiches, il gère aussi les accès des utilisateurs à la base. Un utilisateur joue un rôle précis pendant chaque accès, ce rôle est utilisé pour contrôler cet accès.

Si les rôles *Lecteur*, *Auteur*, *Ecrivain* et *Administrateur* sont généralement joués par des utilisateurs humains, le service de conférence est généralement tenu par un programme d'ordinateur dont le noyau est un archiveur de messages.

Le service de conférence est fourni par un ensemble de sites coopérants, chacun gère une partie de la base de fiches (figure 7-3). Chaque utilisateur communique avec un site local pour accéder au service de conférence. Chaque site gère une liste d'utilisateurs locaux, et a un administrateur local s'occupant des tâches administratives du site.

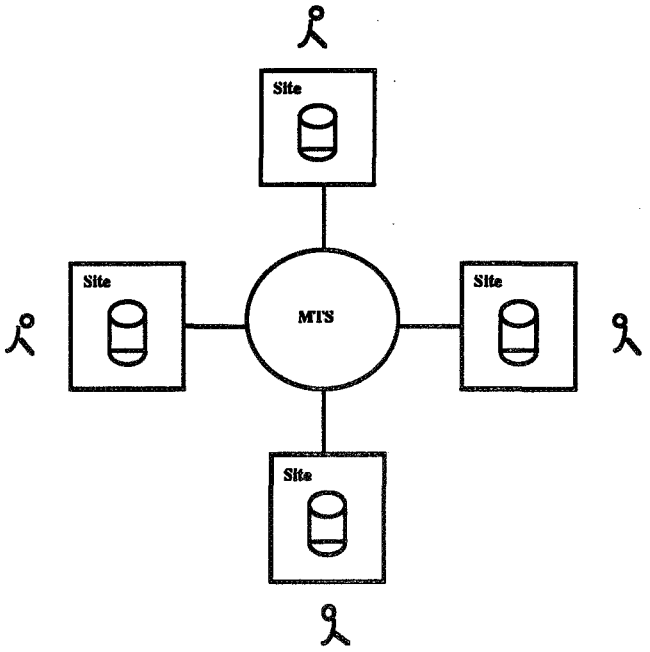


Figure 7-3. Les Sites Coopérants

3.1.1. Sous-conférences et Répartition

Cette section présente notre extension à CRMM sur la répartition de la base de fiches. La nouvelle conférence est appelée **CRMM+.**

L'idée est de découper une conférence en plusieurs parties, et de distribuer hiérarchiquement ces parties. Un site n'a besoin de savoir que de quel site il reçoit de nouvelles fiches, et il a le droit ensuite de distribuer la partie dans d'autres sites.

Une conférence CRMM+ est divisée en zéro, une ou plusieurs sous-conférences qui peuvent ensuite être divisées en sous-sous-conférences, et ainsi de suite. Chaque sous-conférence contient un sous-ensembles de fiches de sa super-conférence. A chaque instant donné chaque site contient une sous-conférence.
Remarque: pour simplifier la description, la conférence est sa propre sous-conférence, donc un site peut contenir la conférence totale.

Tous les sites d'une conférence forment un *arbre*, dont le site racine contient la base complète de la conférence. Chaque noeud, autre que la racine, contient une sous-conférence.

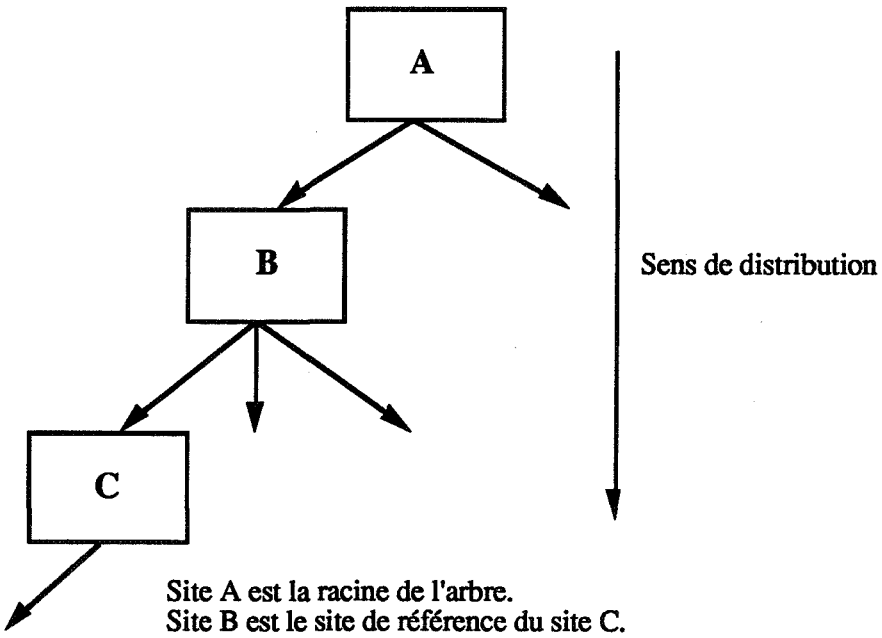


Figure 7-4. Distribution

Chaque site (sauf le site racine) a un site *serveur* ou *de référence*, où il obtient les contributions venant d'autres sites, et par lequel il propage les contributions et les modifications faites dans le site lui-même et ses sites clients. Le site de référence est le site père dans l'arbre (Figure 7-4). Donc, chaque site n'a besoin de connaître que son site serveur et/ou ses sites clients (s'il sert les autres sites). Cela simplifie énormément la tâche d'administration.

Un utilisateur d'un site non-racine peut lire des fiches (s'il est lecteur), ajouter une fiche (s'il est écrivain ou éditeur), ou détruire une fiche (s'il est auteur de la fiche à supprimer). Seul l'administrateur global (l'administrateur de la racine) a le droit d'effectuer les autres modifications comme ajouter un lien, détruire une fiche, etc.

3.1.2. Stratégie de Synchronisation

La synchronisation pour chaque site consiste à evoyer

- à ses sites clients (s'il n'est pas un site *feuille*) des *modifications* effectuées dans le site et venues de son site *serveur* (ou de *référence*).
- à son site serveur (s'il n'est pas *racine*) des *modifications* effectuées dans le site et venues de ses sites *clients*, (figure 7-5).

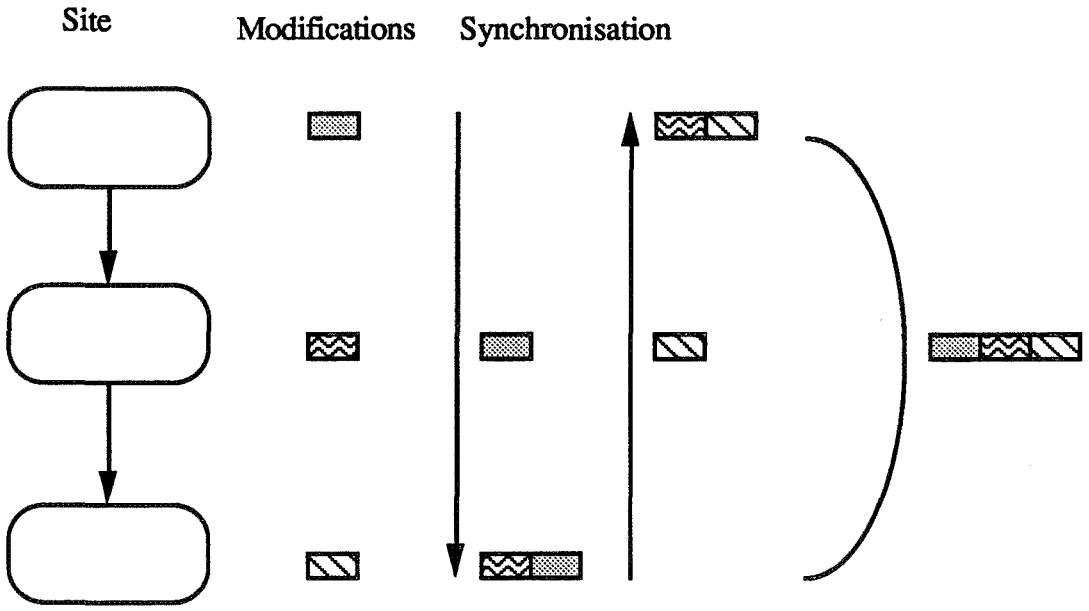


Figure 7-5. La Synchronisation

La stratégie de distribution doit être spécifique en fonction du site qui distribue et du site qui reçoit la distribution. La stratégie de la distribution dynamique ou de la synchronisation peut être différente pour chaque paire de *père-fils*. Les paramètres suivants doivent être décidés par deux sites *père-fils*:

- **Quand**: quand propage le fils des modifications vers le père?
  quand propage le père des modifications vers le fils?
  Le choix peut être:
  - périodique (chaque samedi à minuit, par exemple).
  - immédiatement
  - à la demande du receveur.
  - décider par l'administrateur.

- **Comment**:
  - en version: ceci est utile pour recopier un sous-ensemble de la base.
  - en opération: c'est le mode normal d'échange de modifications.

3.1.3. Administration et Opérations Locales

Chaque site a un administrateur local qui effectue le travail administratif de la sous-conférence qu'il contient. Il existe un administrateur global qui est l'administrateur du site racine qui gère globalement la conférence.

Les tâches d'un administrateur local consiste à:

- Affecter des rôles aux utilisateurs locaux, il ne peut affecter que des rôles autorisés par l'administrateur de son serveur. Il ne peut que permettre à ses clients d'affecter un sous-ensemble des rôles autorisés par son serveur.

- Gérer la distribution: ajouter un site client et décider de la stratégie de synchronisation avec son serveur et ses clients.
- Nettoyer la base locale: effacer les parties inutiles de fiches. **Attention**: ces opérations sont locales!
- Effectuer des opérations locales: par exemple, ajouter un attribut local, etc.

Les tâches de l'administrateur global sont:

- Affecter des rôles aux utilisateurs locaux.
- Décider les rôles que ses clients peuvent affecter.
- Gérer la distribution: ajouter un site client et décider de la stratégie de synchronisation de ses clients
- Gérer la base: effacer les parties inutiles de fiches ou détruire des fiches.

Attention: ce nettoyage a un effet global, il sera propagé vers tous les sites.

- Ajouter ou supprimer des sous-conférences.
- Gérer des liens: ajouter et supprimer des liens.
- Effectuer des opérations locales: les opérations locales au site racine, e.g., ajouter un commentaire local, etc.

3.1.4. Identification

La méthode d'identification est décrite comme suit:

- Chaque site a un nom (et une adresse) global, i.e. O/RName. Il possède également un *alias* (*surnom* en français) qui est une chaîne de caractères relativement courte.
- Chaque sous-conférence a un nom global, i.e. un O/RName. Un alias est aussi associé avec chaque sous-conférence.
- Chaque fiche a un nom externe global qui est composé de l'alias du site où elle a été créée et d'un numéro de séquence unique parmi les fiches créées dans le site.
- Chaque utilisateur a un nom global (i.e. un O/RName).

3.1.5. Situations Anormales

Il est possible que certaines *modifications* soient perdues en route ou que les modifications sur une fiche arrivent avant la création de la fiche.

(1) Modifications perdues: comme ce qu'on a décrit avant, il est acceptable d'avoir des incohérences dans les conditions extrêmes. L'administrateur peut ensuite prendre la version correcte à partir de son site de référence. Si le site est déjà racine, dans ce cas, il possède la version correcte.

(2) Les opérations arrivent avant la création de l'objet, ou bien une fiche est référencée dans un site avant qu'il ne l'ait reçue. Dans ce cas, un objet incomplet (ou une fiche *fictive*) est créé automatiquement. Les modifications sont rattachées à la fiche fictive, et elles seront exécutées dès que la "*vraie*" fiche arrivera.

3.2. Réalisation Basée sur MAP

Dans cette section, nous présentons rapidement la réalisation des archiveurs de CRMM+.

Chaque site de conférence a un archiveur qui est basé sur MAP (figure 7-6).

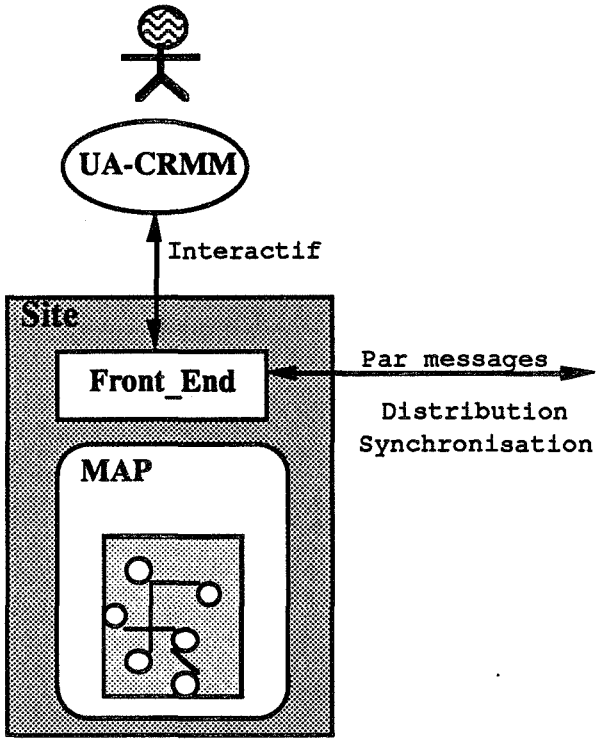


Figure 7-6. Un site CRMM

On peut identifier deux sortes de tâches pour un site:

- Gérer et stocker des fiches, gérer la distribution, et la synchronisation des fiches. La base d'information de MAP stocke les fiches.
- Gérer une ou plusieurs interfaces avec les clients, qui sont des agents utilisateurs de la conférence. Par exemple, une interface en mode interactif, une autre en mode messagerie.

Les tâches spécifiques aux utilisateurs (i.e. contrôle de nouveauté) sont gérées par les agents utilisateurs.

Pour ce qui nous concerne, nous allons étudier comment utiliser MAP pour gérer des fiches et distribuer/synchroniser des informations.

3.2.1. Objets et Leur Organisation

Chaque site est construit au-dessus d'un MAP qui stocke et gère une base d'information, la base d'information contient les fiches du site. Une fiche est représentée par un item atomique, i.e. un item sans RCP (Reference Content Part). Les fiches sont le seul type d'objets d'information à stocker dans la base de MAP. Les facilités d'organisation, comme classeurs, ..., ne sont pas fournies dans CRMM. Les attributs et les liens sont représentés par les attributs de l'item

correspondant. La base de définitions d'attributs de MAP contient les définitions de tous les attributs de CRMM. Le descripteur "*nature*" d'un attribut représentant un lien est stocké comme un descripteur spécifique avec la définition de l'attribut. En plus des attributs de CRMM, une fiche peut avoir aussi des attributs locaux dans CRMM+, e.g. un commentaire. Le nom global d'une fiche est représenté par le nom externe de la fiche, la correspondance est gérée par MAP. Les sous-conférences sont représentées directement par des contextes de MAP. Tous les attributs globaux (définis par CRMM et dynamiquement par l'administrateur global) sont reconnus par toutes les sous-conférences.

Les opérations sur des objets (fiches, conférences, etc) sont réalisées directement grâce aux facilités de manipulation de MAP. Par exemple, l'*ajout* d'une fiche dans une sous-conférence est fait par l'*ajout* de l'item correspondant dans le contexte correspondant.

3.2.2. Utilisateurs, Rôles et Contrôle d'Accès

Chaque site CRMM(+) a un ensemble d'utilisateurs locaux qui accèdent au site à partir d'un agent utilisateur. Dans CRMM+ comme dans CRMM. Chaque utilisateur est globalement identifié par son nom qui est la combinaison du nom de site où il est un utilisateur local et d'un identificateur local. Donc le FE doit affecter un **UID** à chaque utilisateur local (en utilisant un DS s'il est disponible) pour pouvoir utiliser MAP. Le FE gère la correspondance (via DS ou non) et affecte des rôles à chaque utilisateur local. Dans le cas où un DS global n'existe pas, le FE est obligé de faire la correspondance entre le nom global d'un utilisateur et son UID, ceci est souvent simple puisque le nom du site dans CRMM est désigné par un alias.

Le contrôle d'accès est en fait prédéfini par le modèle de conférence, il suffit de définir un seul Contrôleur pour toute la base. Ce contrôleur est:

- Droit d'Accès: Lire
- U_Part: Vide
- A_Part: un identificateur fixé pour désigner la conférence.
- R_Part: Lecteur, Auteur, Ecrivain, Editeur, Administrateur

- Droit d'Accès: Ajout
- U_Part: Vide
- A_Part: un identificateur fixé de la conférence.
- R_Part: Ecrivain, Editeur, Administrateur

...

3.2.3. Répartition et Synchronisation

Chaque contexte correspond à une sous-conférence, chaque *site client* duplique un contexte de son *site serveur*. Le FE du site gère les stratégies de distribution et de synchronisation: pour chaque site voisin (le site père ou les sites fils), le contexte dupliqué, les stratégies de distribution et de synchronisation.

Les *modifications* sont échangés par messages. Chaque message contient:
- soit une seule opération: les noms des fiches et des utilisateurs dans l'argument de l'opération doivent être globaux.
- un ensemble d'opérations.
- une fiche complète, ou une version.

Le mécanisme d'historique est utilisé pour mémoriser des opérations à changer. Le FE n'a besoin de mémoriser que les EIDs des événements des opérations à propager. C'est à dire que le FE associe à chaque site (père ou fils) un ensemble de EIDs à propager. Quand le FE reçoit des *modifications* d'un autre site, il les exécute (i.e. passe au MAP et/ou garde certaines informations). Lorsqu'il s'agit d'un ajout, il regarde dans l'historique de la fiche s'il existe des opérations en attente, si oui il les exécute après l'ajout. Les modifications sont ensuite préparées pour être propagées dans le même sens (i.e. vers les fils si elles sont venues du père, vers le père si elles sont venues d'un des fils). Quand le FE reçoit une opération venant d'un agent utilisateur, il l'exécute, ensuite il la prépare pour propager vers le père et les fils. Dépendant des stratégies, ces *modifications* peuvent être envoyées immédiatement ou mémorisées par leurs EIDs.

3.2.4. Références et Chaining

Dans CRMM+, il est possible de spécifier d'autres sortes de relations de répartition entre fiches, i.e. **Référence** et **Chaining**.

(1) Quand une fiche est *incomplète* dans un site non-racine, le FE demande automatiquement une version complète à son site de référence (ou plus complète que l'existante) lorsue la fiche est consultée par un utilisateur.

(2) Si l'administrateur local d'un site non-racine a l'impression qu'une fiche n'est pas à jour à cause de la perte d'*modifications*, il peut donner un ordre à l'archiveur du site de demander au *site de référence* une version complète de la fiche. Cela pourra se faire pour un ensemble de fiches.

(3) Un *site client* demande une version complète, le site référence n'en dispose pas, il doit donc la demander à son site de référence.

Remarque: Un site serveur peut envoyer la version complète, immédiatement ou non. Cela dépend de l'accord entre deux sites (les stratégies de distribution) et de la disponibilité de la version complète.

3.2.5. Administration

Chaque site a un administrateur local qui s'occupe des tâches administratives du site. L'administrateur local du site racine est l'administrateur global de la conférence, il se charge des tâches administratives de la conférence globale en plus des tâches administratives locales du site racine. Le FE doit fournir des opérations d'administration à l'administrateur local comme décrit dans le modèle de la conférence, les opérations manipulant des fiches sont passées directement au MAP et propagées si elles ne sont pas locales.

L'ajout d'une sous-conférence, i.e. initialiser et ajouter un nouveau contexte ne peut être exécuté que par l'administrateur global, et sera propagé immédiatement vers tous les *sites*

concernés descendant de l'arbre. Un site est concerné s'il contient la sous-conférence ou la super-conférence de la sous-conférence.

L'administrateur global peut ajouter dynamiquement la définition d'un attribut global, la définition est propagée vers tous les sites dans le sens de distribution.

4. Discussions

La distribution et la synchronisation sont essentiellement *centralisées* au sens où une site de *référence/maître* existe et s'occupe de la distribution et de la synchronisation de ses *esclaves*. Une alternative est de permettre la communication directe entre les esclaves, pour échanger certaines *modifications*, par exemple, les ajouts d'objets (fiches). Cela améliore la performance. Mais la gestion devient un peu plus compliquée dans ce cas.

La méthode *centralisée* exige qu'un site *central* conserve tous les objets d'information (fiches) de l'application (conférence). Cela peut poser du problème aux applications *décentralisées* qui ont besoin d'une gestion décentralisée. De plus, le site *central* peut être insuffisant dans le cas où la quantité d'information est immense, par exemple, une conférence mondiale pour PC. Une meilleure solution consiste à avoir plusieurs sites de *référence*, chacun gère un sous-ensemble d'objets (un sous-contexte ou une sous-conférence).

Vis-à-vis des relations entre sous-contextes qui sont *dupliquées* dans plusieurs archiveurs, il peut aussi y avoir d'autres modèles possibles pour la distribution et la synchronisation:

Décentralisé: les contextes (dupliqués) ont des rôles équivalents dans ce modèle. Chaque contexte connaît tous les autres, les *modifications* dans un contexte sont propagées vers tous les autres. Un critère est nécessaire pour chaque contexte afin de décider l'ordre des *modifications* provenant de différents contextes. Ce critère doit en plus obtenir les mêmes résultats dans les différents contextes afin de garantir la cohérence. Un critère possible est la *date* de lancement de la *modification*. Les *modifications* sont reséquencées d'après leur date de création et ré-exécutées si nécessaire. Cependant dans un environnement distribué, la notion de *temps* global n'existe pas. Un administrateur a le contrôle total du temps local de sa machine, une"bêtise" de sa part peut perturper d'autres sites. Un autre inconvénient est que la gestion décentralisée est plus difficile que la gestion centralisée. Il est possible de perdre d'informations si tous les sites/contextes décident de ne conserver que de petits objets.

Centralisé: parmi l'ensemble des contextes, il y a un *maître* qui reçoit toutes les *modifications* de ses *esclaves* et les re-propage vers les autres. L'inconvénient ici est indiqué ci-dessus. De plus, le maître risque d'être surchargé. Une méthode d'élection doit être mise en place en cas de panne du maître. L'avantage principal est que l'administration et la gestion sont simples.

Semi-Décentralisé: Dans ce modèle, tous les contextes peuvent être *maîtres* et *esclaves*. Un contexte peut être *maître* pour un sous-ensemble des objets/instances *maîtres*, et *esclaves* pour les autres objets. Chaque objet a une instance maître qui se trouve dans un des contextes (par exemple, où il a été créé), toutes les *modifications* concernant cet objet sont d'abord propagées vers le contexte contenant l'instance *maître* et sont ensuite propagées vers les autres contextes contenant une instance *esclave*. L'avantage de ce modèle est qu'il combine la méthode centralisée et celle décentralisée, ainsi il évite certains inconvénients de ces deux méthodes. Cependant le

modèle *semi-décentralisé* n'a ni l'avantage complet du modèle *centralisé* ni celui du module *décentralisé.*

Il est possible de combiner ces méthodes en regroupant l'ensemble des contextes dupliqués en plusieurs sous-ensembles, chaque sous-ensemble peut avoir son propre modèle de distribution et de synchronisation. Un contexte *gateway* peut se trouver sur plusieurs sous-ensembles et utilise plusieurs stratégies de distribution différentes suivant les sous-ensembles auquels il appartient, les informations sont donc distribuées et synchronisées entre ces différents sous-ensembles à travers de ce gateway.
En conclusion, les choix du modèle et des stratégies de distribution et de synchronisation dépendent de l'application et son environnement: un choix peut être adapté à une application dans un environnement, mais pas à d'autres dans des environnements différents.

Chapitre 8
Applications dans l'Environnement Bureautique
POMS: Personal Office Message Store

Dans ce chapitre, nous présentons la conception d'un archiveur de messages personnel dans un environnement de bureau, et sa réalisation basée sur MAP. Ce travail a été publié dans [You 89c] qui contient une description plus compacte, le lecteur peut trouver la spécification du système dans [You 89d].

1. Introduction

Les employés dans un environnement bureautique envoient, reçoivent et manipulent des messages quotidiennement. Ils ont besoin des différents types d'archiveurs de messages suivants:

- *Boîtes Aux Lettres Personnelles* qui contiennent tous les messages reçus du distributeur de messages standard (e.g., MTS de X.400).
- *Boîtes Partagées* qui contiennent les messages partagés entre un groupe de personnes.
- *Bulletins Boards* et *Conférences* qui permettent aux utilisateurs de partager des messages d'une manière structurée et organisée.
- *Archives* qui gardent les messages pour une période relativement longue. Les messages sont souvent organisés en *folders*, *cabinets*, *classeurs*, etc. Une facilité de *recherche* est souvent fournie qui permet de retrouver les messages archivés d'après leurs attributs, comme *auteurs*, *sujets*, *expéditeurs*, etc.

Cependant, il n'y a pas d'archiveur disponible dans l'environnement X.400 qui fournisse des facilités suffisantes. Le but de **POMS (Personel Office Message Store)** est de stocker et gérer les messages personnels dans un environnement bureautique utilisant la messagerie X.400, il traite de plus des objets d'information concernant les messages, e.g., *post-it-notes*. POMS est basé sur le standard d'archiveurs de messages (MS pour Message Store) des CCITT/ISO (X.413), il supporte en plus la classification des messages, les opérations automatiques. Donc il peut être vu comme une extension de MS X.413.

POMS est un serveur personnel de stockage des messages . Il est conçu pour être utilisé dans un environnement mixte avec des machines puissantes (mainframes, stations de travail de haut de gammes, etc) et les *petits* ordinateurs (e.g., station de travail de bas de gammes, PC, etc). Le serveur (POMS) se situe généralement sur des grosses machines avec beaucoup de ressources et actives la plupart du temps, les clients de POMS se situent dans des petites machines qui ne disposent pas de beaucoup de ressources et ne sont pas toujours actives (figure 8-1).

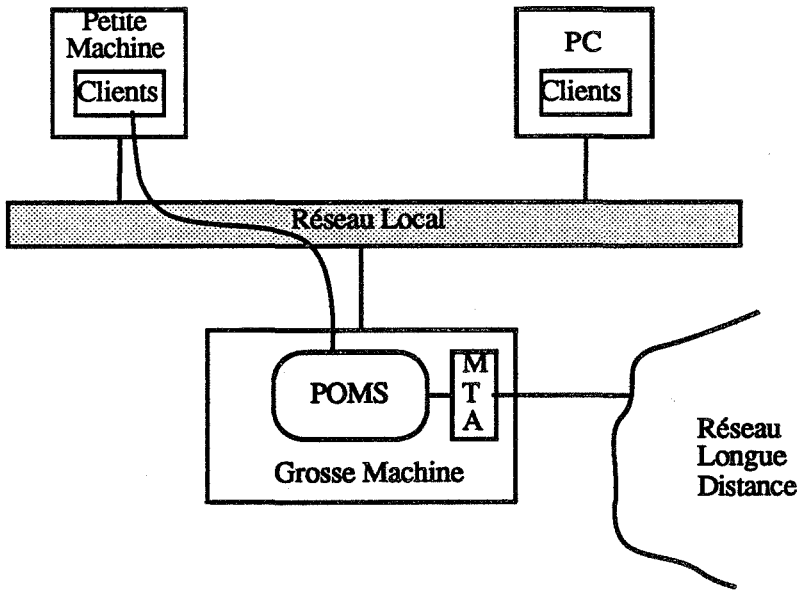


Figure 8-1. L'environnement de POMS

2. Manipulations des Messages dans un Environnement Bureautique

La figure 8-2. illustre les activités typiques de manipulation de messages dans un environnement bureautique et le flux d'information:

- **Extraire** ou **lire** (Get) des nouveaux messages à partir du *mail-box* où on dépose les nouveaux messages.
- **Classer** (Classify) des messages dans les *cabinets* d'après les critères personnels, et avec les commentaires personnels.
- **Rechercher** ou **lire** (Search) des messages désirés dans les *cabinets* d'après des critères comme "auteur", "sujet", etc.
- **Jeter** (Throw) des messages inutiles dans la *poubelle.*
- **Remettre** (Send) les messages dans l'*out-box* pour les envoyer. Les messages dans l'*out-box* seront remis au MTA. Les copies des messages envoyés sont conservées dans le *history-box.*
- **Consulter** (Look-Up) le *history-box* pour obtenir les messages envoyés.
- **Sortir** des messages des boxes (cabinets, poubelle, out-box, etc).

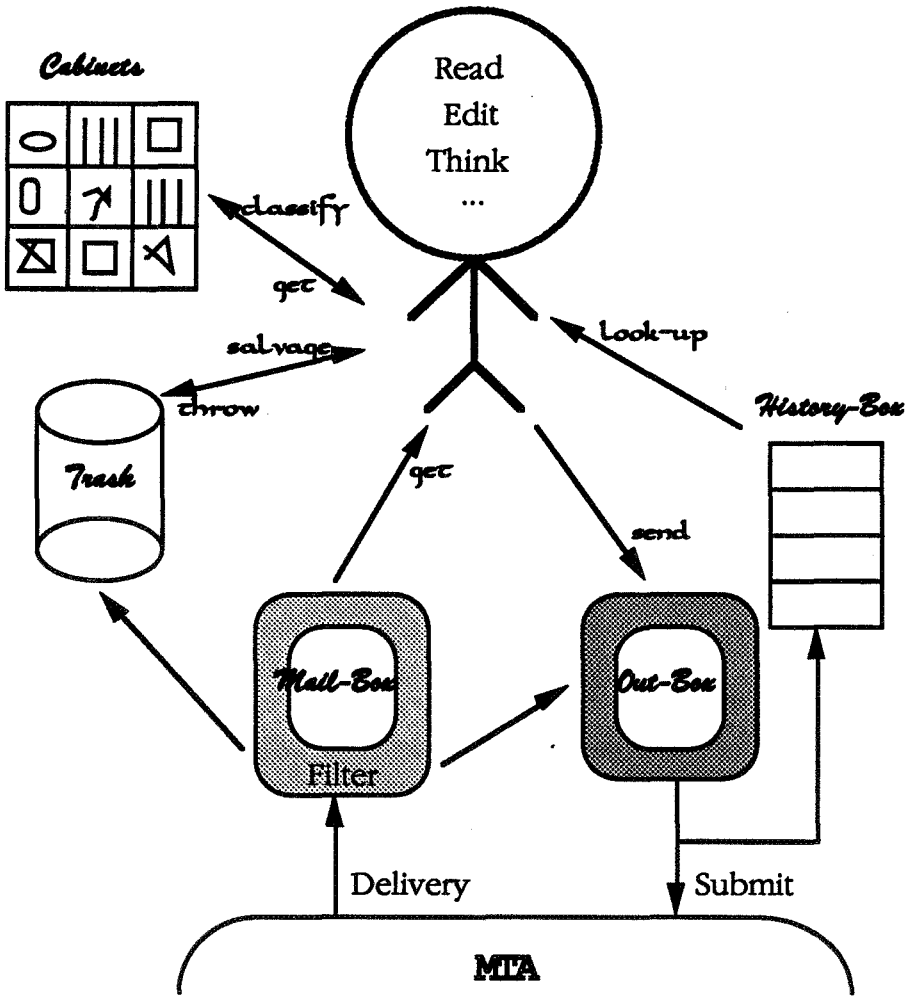


Figure 8-2. Flux d'information

3. Modèle POMS

3.1. Modèle fonctionnel

POMS est un archiveur personnel des messages, il est mono-utilisateur. Un archiveur POMS communique d'une part avec son client, i.e. l'agent utilisateur, d'autre part avec le système de transfert de messages, i.e. MTS. Le modèle fonctionnel (figure 8-3) spécifie les relations entre POMS et les éléments avec lesquels il communique.

La fonctionnalité primaire du POMS est d'accepter les messages venant du MTS et les garder pour ensuite être recherchés ou lus par son utilisateur final. Il fournit aussi les services de la *soumission-indirecte* de messages et de l'*administration* à son utilisateur.

A part de ces fonctionnalités essentielles, POMS fournit les autres services orientés-utilisateurs. Il permet à l'utilisateur final d'organiser des messages dans des *cabinets, classeurs*, etc, d'ajouter des commentaires personnels à un message particulier ou à un ensemble de messages, et de stocker d'autres types d'objets d'information simples et de les relier aux messages existants. La figure 8-3 montre le service abstrait fourni par POMS en relation avec son utilisateur et le MTS.

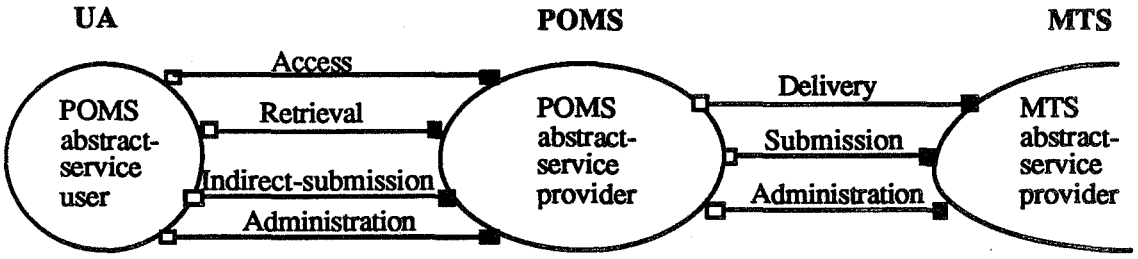


Figure 8-3. Modèle Fonctionnel

Dans le modèle fonctionnel, le POMS est vu comme un objet atomique, il fournit son service via les portes *Access*, *Retrieve*, *Indirect-Submission* et *Administration*. La porte **Access** fournit les services additionnels du POMS par rapport à ceux définis par X.413.

3.2. Modèle d'Information

Les trois types d'objets d'information sont introduits dans POMS pour représenter les messages, cabinets/classeurs et les objets d'information de bureau associés aux messages:

- **Entrées**: elles sont utilisées pour représenter des messages, en particulier, les messages inter-personnels. Chaque message est représenté par une entrée. Cette entrée peut avoir des *filles* qui représentent des messages emboîtés dans l'entrée, ces entrées filles sont reliées à l'entrée principale.
- **Collections**: les collections sont utilisées pour représenter des classeurs, cabinets, etc. Chaque collection contient un ensemble de références à d'autres objets d'information. Les collections sont organisées en hiérarchie.
- **Post-it-notes**: les post-it-notes contient des informations de l'utilisateur final. Ces informations peuvent être le résumé d'un message, ou le sommaire d'une collection de messages. Les post-it-notes sont souvent liés aux messages.

Dans les sections suivantes, nous allons étudier en détails chaque type d'objets d'information.

3.2.1. Entrées

Chaque entrée représente un message, qui peut être un vrai *message* inter-personnel, un *essai* (probe) ou un *rapport* (cf.[X.411] et [X.420]). Chaque entrée est composée d'un ensemble d'attributs qui est groupé en quatre catégories (figure 8-4):

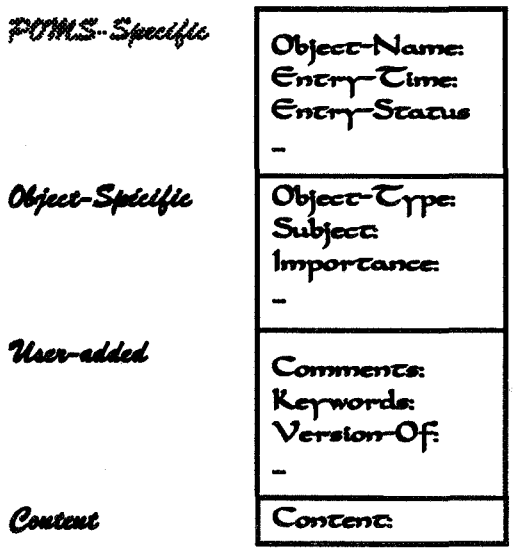


Figure 8-4. Les attributs d'une entrée

- *POMS-Spécifique*: ce sont des attributs spécifiques à POMS qui ne sont pas présents dans le message original. Par exemple, le nom interne de l'entrée ou la date de l'entrée dans le POMS. Les attributs de cette catégorie sont obligatoires pour toutes les entrées.

- *Object-Spécifique*: ce sont des attributs contenus ou déduits du message original. Par exemple, l'expéditeur, les destinataires, l'importance, la priorité, etc. Deux attributs particuliers sont définis qui reflètent la méthode de représentation des messages de POMS, ils sont *P1* et *P2*. Ces deux attributs contiennent l'enveloppe complète du message au niveau MTS et l'en-tête complet au niveau IPM (si elle existe). Ces deux attributs nous garantissent qu'il n'y aura pas de perte d'information, ils nous permettent d'extraire seuls les champs intéressants du message.

- *Contenu*: c'est le corps du message inter-personnel, les messages emboîtés sont remplacés par les références aux entrées représentant ces messages. Les entrées correspondantes aux messages emboîtés sont créées s'ils ne sont pas déjà stockés.

- *User-Spécifique*: ce sont des attributs ajoutés dynamiquement par l'utilisateur, par exemple, les commentaires, les mots-clé, les liens avec d'autre messaged.

POMS stocke tous les messages déposés par le MTS et tous les messages envoyés par l'UA. Selon les types des messages représentés par des entrées, on peut classifier les entrées en quatre classes:

- **Message**: une entrée *message* représente un message IP. Les attributs de l'entrée sont extraits de l'enveloppe MTS et l'en-tête IP, e.g., priorité, importance, sujet, EnRéponseA (InReplyTo), etc. On peut utiliser ces attributs pour effectuer la recherche. Chaque message doit au moins être lié à une *collection.* Une entrée *message* est soit fabriquée automatiquement par POMS à partir d'un message IP en format *P1+P2* déposé par MTS ou envoyé par l'UA, soit stockée explicitement en format *entrée* par l'UA.

- **Probe**: une entrée *probe* représente un "essai" de la messagerie X.400 envoyé par l'UA. Il n'a pas d'attributs P2 ni de contenu. (**Remarque**: un utilisateur ne reçoit jamais de *probe*)

- **Rapport**: une entrée *rapport* est un message contenant un rapport qui est fabriqué soit par MTS (un MTA) soit par un UA. Seuls les attributs *objets-spécifiques* sont définis, i.e. *ReportOf* qui lie l'entrée au message dont il est rapport et *ReportType* qui indique le type de rapport, e.g., *négatif* ou *positif*. Le contenu de rapport contient le rapport entier dans son format original X.400. Il faut noter que les rapports et probes ont souvent une durée de vie très courte dans le POMS, un probe est généralement supprimé quand le rapport correspondant est arrivé et lu. Un rapport est automatiquement supprimé lorsque le message ou probe auquel il est associé est supprimé.

- **Double**: il est possible que le POMS reçoive plusieurs fois un même message IP. Par exemple, lorsque l'utilisateur est sur plusieurs listes de distributions et le message a été envoyé à plusieurs de ces listes; ou lorsque l'utilisateur reçoit directement le message et il est aussi retransmit par un ami emboîté dans un autre message. Ces messages sont différents de leurs enveloppes MTS qui peuvent être utiles à garder, e.g., pour connaître les chemins par lesquels ils sont arrivés. Cependant il est souhaitable de conserver une seule copie du message pour que l'on puisse référer uniquement au message et économiser le stockage. Une entrée *double* est créée quand un message arrive dont une copie existe déjà. Un attribut spécial pour les entrées *doubles* est *DoubleOf* qui contient la référence de la copie principale. Une entrée *double* n'a pas de *contenu* ni d'attribut *P2*. Quand un message est référencé, le nom du double peut être attaché avec la référence pour indiquer la copie du message si elle est différente de la copie principale. Par exemple, pour référencer un message emboîté qui est le double d'un message existant, le RCP fait référence à la copie principale du message avec un indicateur sur le double. Quand un message principal est supprimé, tous les doubles seront également supprimés.

L'utilisateur manipule les attributs *User-Spécifiques* de n'importe quelle entrée, e.g., ajouter un *commentaire* ou un *mot-clé*. Il peut lire une entrée ou plusieurs attributs de l'en-tête (Remarque: on traite ici le contenu comme un attribut spécial).

Les listes des attributs pour chaque type d'entrée sont omises ici, le lecteur intéressé est prié de se référer à [You 89d].

3.2.2. Post-it-notes

Les **post-it-notes** sont utilisés pour représenter les informations diverses liées aux messages, en particulier, les étiquettes collées avec des messages. Chaque post-it-note a un ou plusieurs liens *StickTo* qui lient le post-it-note à un ou plusieurs d'autres objets: entrées, collection ou même d'autres post-it-notes. Un post-it-note a la même structure qu'une entrée, mais son contenu ne contient pas d'objets emboîtés, il n'a pas non plus d'attributs *P1* et *P2*. Un post-it-note est automatiquement supprimé s'il n'a plus de lien *StickTo*. On supprime automatiquement un lien *StickTo* si l'objet référencé est supprimé.

3.2.3. Collections

Les messages (i.e. entrées) sont organisés en *collections*. Chaque collection contient un ensemble d'objets et un ensemble d'attributs. Une collection a la même structure qu'une entrée (figure 8-4), le *contenu* d'une collection est un ensemble de *références* aux objets contenus. Evidemment les attributs *object-spécifiques* d'une collection ne sont pas les mêmes que ceux des entrées, la liste est donnée dans [You 89d]. Les membres d'une collection peuvent être des entrées, des post-it-notes, ou d'autres collections. Une collection peut avoir une ou plusieurs *super-collections*. Les collections forment une hiérarchie.

Six collections sont prédéfinies dans POMS qui ont les sémantiques spécifiques:

- **Root** est la racine de la hiérarchie des collections, elle est la seule collection qui n'a pas de super-collection.
- Toutes les entrées venant du MTS sont automatiquement ajoutées dans la collection **In**. Une entrée n'est pas retirée automatiquement de la collection **In** après être lue.
- **Out** contient seuls les messages qui ne sont pas encore soumis au MTS. POMS essaie de les soumettre au MTS périodiquement. Lorsqu'un message est soumis, il est retiré de l'*Out* automatiquement et ajouté dans l'*Out-History*. L'utilisateur ne peut pas ajouter ni retirer un membre de l'*Out*.
- **Out-History** contient toutes les entrées/messages soumises. Ses membres sont ajoutés automatiquement par POMS. L'*Out-History* sert donc le *chrono* des envois pour l'utilisateur. L'utilisateur ne peut pas ajouter un membre directement dans L'*Out-History*.
- **Cabinets** est la racine des classeurs de POMS, l'utilisateur peut ajouter des entrées ou créer des nouvelles collections.
- **Trash** est la poubelle de POMS. Tous les objets supprimés sont ajoutés comme ses membres. Le *Trash* est vidé périodiquement par POMS ou explicitement par l'utilisateur. Un objet dans le *Trash* peut être lu et restoré en le retirant et pour le lier à une autre collection.

Une entrée *message/probe* ou une collection doit appartenir à au moins une collection. Un tel objet ne peut être supprimé s'il est membre d'une collection. Un utilisateur peut ajouter un objet dans une collection ou retirer un membre de la collection.

La figure 8-5 illustre la hiérarchie des collections.

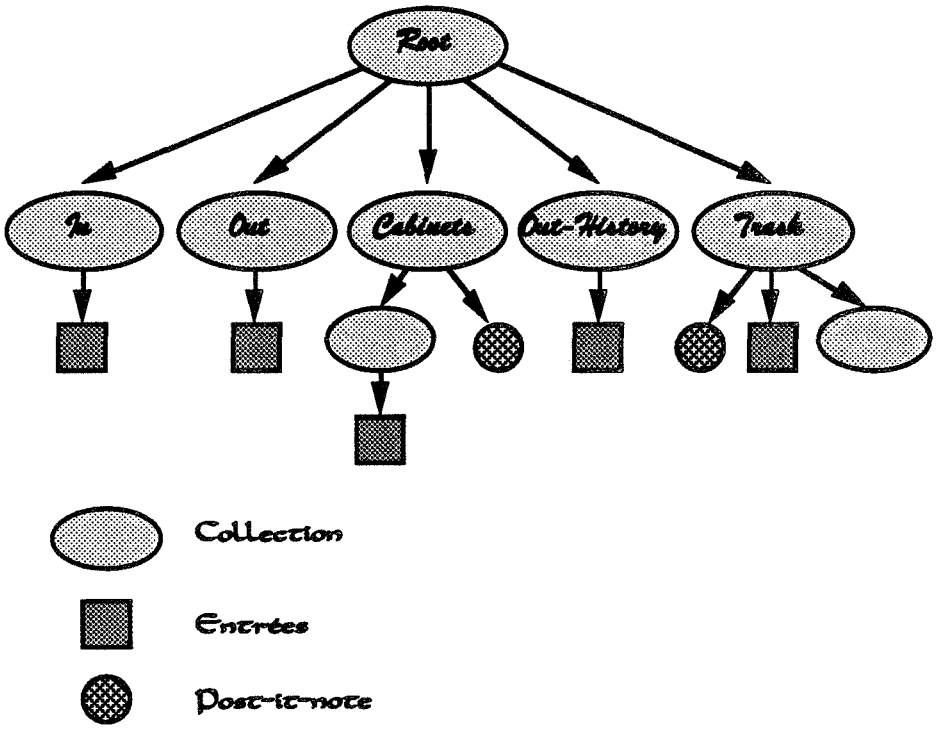


Figure 8-5. Hiérarchie de Collections

3.2.4. Identifications

A chaque entrée est associé un numéro de séquence (*SequenceNumber*) qui identifie l'entrée de façon unique dans POMS. Le numéro de séquence est affecté automatiquement par POMS lorsque l'entrée est créée. En plus du *SequenceNumber*, chaque entrée peut avoir un identificateur externe qui est soit un IPM-ID (pour les messages IP) soit un Identificateur-MPDU (Cf. [X.400]). Deux messages IP sont considérés comme étant les mêmes si et seulement si ils ont le même identificateur externe.

Le nom interne d'une collection ou post-it-note est une chaîne de caractères donnée par l'utilisateur lors de sa création. Ils n'ont pas de nom externe.

3.2.5. Attributs et Liens

Les attributs représentent diverses informations, e.g., les propriétés, liens, etc. Chaque attribut a un type, une ou plusieurs valeurs. Les descripteurs des attributs sont prédéfinis par POMS.

Les objets dans POMS sont liés, ils peuvent avoir tous les types de liens dont certains sont prédéfinis avec des sémantiques spécifiques:

- *Membership* relie une collection avec ses membres et une entrée *message* avec ses messages emboîtés.
- *ReportOf* relie un rapport avec l'entrée à laquelle il se rapporte. Quand l'entrée est supprimée, le rapport l'est aussi.
- *StickTo* relie un post-it-note à un ou plusieurs autres objets. Lorsque tous ces objets sont supprimés, le post-it-note est aussi supprimé.

Il y a aussi d'autres liens prédéfinis, cependant leurs sémantiques ne sont pas différentes pour POMS. L'utilisateur peut ajouter dynamiquement un lien de n'importe quel type.

3.2.6. Utilisateurs

POMS suppose qu'il existe un serveur de directory qui gère les utilisateurs et leurs attributs. Un utilisateur peut être identifié soit par son nom dans le directory qui est un O/RName dans la version actuelle du système, soit par un alias. Chaque utilisateur peut avoir des attributs rôles associés qui spécifient les rôles de l'utilisateur dans l'organisation.

4. Service Fourni

Le service du POMS est fourni via ses portes. Il permet à l'utilisateur de *soumettre* un message au MTS, *stocker* une entrée, *lire* une entrée en format P1+P2 ou en format POMS, *créer* une collection, *attacher* un post-it-note, *supprimer* un objet, *déplacer* ou *lier* les objets dans différentes collections, *lister* un ensemble d'objets et *rechercher* des entrées d'après leurs attributs, *vider* la poubelle, *ajouter* le commentaire et mots-clé dynamiquement, *manipuler* les liens, etc. POMS communique automatiquement avec le MTS pour recevoir les messages et les stocker dans la collection *In*.

Les descriptions détaillées sont omises, la liste des opérations et la description du service abstrait fourni par POMS sont données dans [You 89f].

5. Actions Automatiques

Un des avantages d'un archiveur de messages géré par l'ordinateur est que certaines actions peuvent être effectuées automatiquement par l'ordinateur quand un message arrive, cela peut être analogique à la secrétaire d'un directeur. Les exemples des actions effectuées automatiquement sont *filtrage automatique*: les messages jugés inutiles sont jetés automatiquement à la poubelle; *classification automatique*: les messages sont classifiés dans les classeurs appropriés. Un autre exemple est d'envoyer automatiquement une réponse après avoir reçu un message qui demande une réponse, lorsque le destinataire est absent ou en vacances.

Dans notre système, l'utilisateur peut spécifier des actions qui seront effectuées automatiquement quand un nouveau message est déposé par le MTS. L'utilisateur spécifie un ensemble de *règles* (**AAR** pour Auto-Action Rule) qui sont *réveillées* par les *dépôts des messages*.

5.1. Règle d'Action Automatique

Une *AAR* permet à l'utilisateur de spécifier les actions à effectuer automatiquement quand une condition sur le message reçu est satisfaite. Une AAR se compose d'une *identification*, d'une partie *description*, d'une partie *condition* et d'un ensemble d'*actions*.

La partie description d'une règle se compose des champs suivants:

- Une description textuelle de la règle.
- La date de création de la règle.
- La date de la dernière modification de la règle.
- La date de la dernière activation de la règle.

- Optionnellement l'historique des activations de la règle, i.e. pour chaque activation si les actions sont exécutées avec succès, sinon la raison de l'échec.

La partie condition est un moule de messages (*Template*), elle est composé des conjonctions d'un ensemble de conditions chacune portant sur un attribut, cette dernière pouvant être la disjonction d'un ensemble de conditions sur le même attribut. La figure 8-5 donne un exemple.

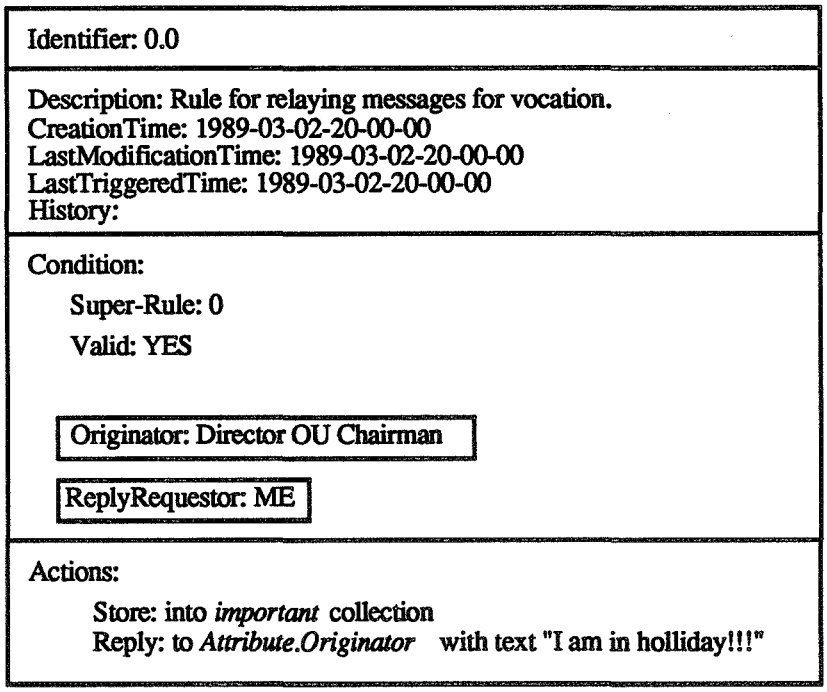


Figure 8-5. Un exemple de Règles

Il n'est pas difficile d'observer que la partie de condition d'une règle définit une classe de messages. D'après la classe des messages que la condition représente, les règles forment une hiérarchie. Pour la simplicité, seulement un arbre est supporté (Figure 8-6). La partie condition contient un indicateur qui spécifie si la règle est *activable* ou non à un moment donné, cela permet d'activer une règle pendant les vacances et de la désactiver après la rentrée. Voici quelques exemples des champs dans une *template*:

- Type d'Objet: si le message est un message IP, un rapport ou un double.
- Expéditeur
- Destinataire
- Sujet
- Importance
- Longueur du message
- ReplyRequestors: les utilisateurs à qui une réponse est demandée.

La partie action est composée d'un ensemble d'actions qui doivent être effectuées quand la règle est activée et que la condition est satisfaite. Chaque action a un code d'opération et des arguments. On donne la liste d'actions disponibles comme suite:

- **Store**: avec une collection comme argument, permet de stocker le message reçu dans la collection sauf *trash*. Cela permet donc d'effectuer la classification automatique.

- **Throw**: jeter le message dans la poubelle.

- **Delete**: avec un identificateur de l'objet comme argument, cette action permet de supprimer un objet après l'arrivée du message. Par exemple, supprimer une ancienne version automatiquement après l'arrivée d'une version plus récente.

- **Forward**: retransmettre le message automatiquement aux utilisateurs indiqués dans l'argument de l'action comme si le message était retransmis par l'utilisateur de POMS.

- **Reply**: envoyer automatiquement une réponse au message reçu et aux utilisateurs indiqués dans l'argument, comme si c'était l'utilisateur de POMS qui l'envoyait. Le corps de la réponse est spécifié comme argument de l'action. Par exemple, l'utilisateur peut s'en servir pour répondre automatiquement à un message qui demande une réponse pendant son absence, en disant: "Je suis absent, désolé de ne pouvoir vous répondre tout de suite".

- **Alert**: envoyer une alarme à l'utilisateur de POMS s'il est en conversation avec POMS lorsqu'un message lui est adressé.

- **ExternalProgram**: c'est un programme externe qui est appelé par POMS avec l'argument donné et le message reçu. Le passage de l'argument et le message peut se faire via une *tube Unix* ou un fichier selon le système d'exploitation utilisé et l'implantation choisie. Le fait de permettre d'appeler les fonctions/programmes externes rend le mécanisme de l'action automatique flexible.

L'identification d'une règle est le nom de sa super-règle concaténée avec un entier unique parmi les sous-règles de la même super-règle. La racine est une pseudo-règle qui n'a pas de *template* ni d'actions, son identificateur est **0**.

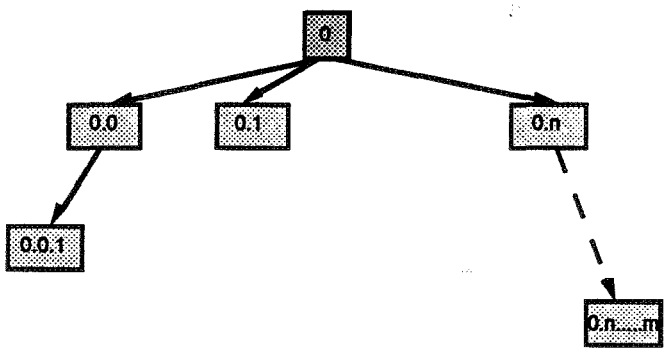


Figure 8-6. Arbre de Règles

Les valeurs d'un attribut, par exemple, le nom d'un objet, les utilisateurs, etc, peuvent être identifiés soit explicitement en donnant leurs valeurs, soit en indiquant un attribut d'un objet qui désigne les valeurs de l'attribut. Il existe une fonction particulière PARENT(Objet) pour identifier les *objets* qui sont les parents du premier objet. Ceci permet de stocker les réponses d'un message dans la même collection que le message question question. On peut utiliser des **rôles** pour identifier des utilisateurs, e.g., le directeur, la secrétaire, etc.

5.2. Résolution de Conflit

Quand un message est déposé par le MTS, POMS cherche dans l'arbre de règles pour trouver les règles activées, et ensuite exécuter les actions associées. Il est possible que plusieurs règles soient activées par le même message et que les actions à exécuter soient incohérentes. Par exemple, l'action *store* et *throw*, ou plusieurs actions *store* sont activées. Il faut donc une stratégie de résolution de conflit.

La stratégie de POMS est:

- Dans le même chemin de l'arbre de règles, les actions d'une sous-règle a la priorité sur les actions de la super-règle. Ceci est la méthode classique de l'héritage dans des systèmes orientés-objet.
- Les actions des règles dans les différents chemins de l'arbre ont les mêmes priorités.
- Si les actions *store* et *throw* sont à exécuter et qu'elles ont la même priorité d'après les deux points précédents, alors l'action *store* a la priorité sur l'action *throw*.
- Une seule action *alert* est exécutée si plusieurs sont activées.
- En cas de conflit, seules les actions qui ont la priorité la plus haute sont exécutées.

6. Réalisation basée sur MAP

Un prototype de POMS (Sauf la partie des actions automatiques) a été réalisé par trois élèves de la section spéciale d'études informatiques de l'Ecole des Mines de St-Etienne ([Cocirta 89]), sur HP9000/318 sous le système d'exploitation UNIX.

Cette réalisation est faite en créant un processus du FE_POMS qui communique d'une part avec le client (UA) de POMS, et d'autre part avec le MTA (Figure 8-7). Elle utilise MAP pour stocker et manipuler tous les objets de POMS. Le processus de POMS est lancé soit par une demande de connexion du client, soit par le MTA (Mail+™). Dans le reste de cette section, nous nous concentrerons sur l'utilisation de MAP, les détails de la réalisation du FE sont omis ici.

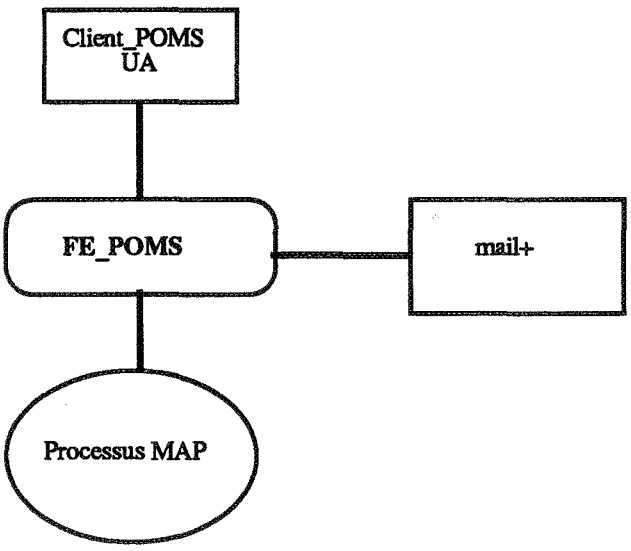


Figure 8-7. La réalisation de POMS

6.1. Utilisation de MAP

Chaque POMS correspond à une instance de MAP, i.e. il utilise une base d'information. Le MAP correspondant a un seul contexte qui fournit l'environnement de travail à l'utilisateur du POMS tel que seul l'utilisateur peut manipuler la base d'information et que tout le monde puisse déposer un message. En fait, il est aussi simple de contrôler les accès par le FE directement qu'utiliser le mécanisme de MAP. La base des définitions de MAP contient les définitions de tous les attributs définis par POMS (voir le mapping des attributs).

6.1.1. Mapping des Objets d'Information

Les objets d'information de POMS sont représentés directement par les items de MAP:

- **Collection**: les collections sont représentées par des items avec seulement les RCP (Reference Content Part) non-ordonnées. Les attributs des collections sont représentés par les attributs des items correspondants.
- **Post-it-note**: les post-it-notes sont représentés par les items avec seulement un DCP (Document Content Part). Les attributs sont représentés comme les attributs des items correspondants.
- **Entrée**: les entrées sont représentées par les items.

La plupart des attributs définis par POMS sont représentés directement par les attributs prédéfinis de MAP, sauf *P1*, *P2* et certains liens (En effet, le choix des attributs prédéfinis par MAP est basé sur les messages IP et l'enveloppe MTS). P1 et P2 sont définis comme attributs spécifiques dans MAP et prennent une chaîne d'octets comme valeur.

Chaque *SequenceNumber* est un OID, les identificateurs externes sont représentés par les noms externes des items. Le nom d'une collection ou d'un post-it-note est représenté comme le nom externe de l'item correspondant.

Chaque opération de POMS est réalisée soit par une opération du MAP (e.g., ajouter un lien), soit par plusieurs opérations. Par exemple, ajouter une entrée dans une collection consiste à ajouter d'abord l'entrée dans MAP, ensuite une RCP dans la collection. La correspondance est pratiquement directe.

6.1.2. Mapping des Utilisateurs

FE_POMS doit associer à chaque utilisateur un UID qui est enregistré dans le directory (un directory local dans notre réalisation). Il gère la correspondance entre les UID et les noms ou alias des utilisateurs, i.e. il doit les changer *au vol* chaque fois entre FE_POMS et MAP, et entre FE_POMS et le client. FE_POMS contient un *cache local* pour accélérer la vitesse du mapping.

Une autre solution consiste de laisser le mapping dynamique des UID et noms des utilisateurs au client, puisqu'il accède aussi au directory. L'endroit où le mapping est effectué est décidé pendant l'établissement de la conversation entre le client et le POMS.

7. Conclusion

Dans ce chapitre, nous avons présenté l'utilisation de SAM pour la conception d'un archiveur de messages personnel (POMS) dans un environnement bureautique. POMS sert d'intermédiaire entre l'utilisateur et le MTA pour la messagerie X.400. Il est une extension de la norme X.413 (sur le stockage de messages) du CCITT et de l'ISO. POMS stocke non seulement des messages reçus du MTA, mais aussi des messages soumis par l'utilisateur, et fournit des classeurs pour organiser tous ces messages. Il permet de plus à l'utilisateur de spécifier des *actions automatiques* qui sont examinées et exécutées automatiquement par POMS après la réception d'un nouveau message. Cela permet donc de la classification et du filtrage automatiques.

Deux aspects spécifiques de POMS ont influencés la conception de MAP dans sa version actuelle (en fait, POMS est conçu avant la version actuelle, ce dernier bénéficie des expériences acquises dans la conception de POMS):

- Les objets d'information de POMS sont typés/classés. Chaque classe contient des objets de mêmes natures (par exemple, messages IP) et a des opérations spécifiques aux objets de la classe. Cela nous a conduit à supporter des classes dans MAP.

- POMS permet des actions automatiques. C'est un sujet plus compliqué, puisqu'il dépend non seulement des états d'objets d'information (stockés et gérés par MAP), mais aussi des états d'utilisateurs (stockés et gérés par DS). MAP ne peut donc supporter tout seul des opérations automatiques; de plus, il n'est pas possible de prédéfinir toutes les actions possibles. MAP choisit donc de ne permettre que le stockage des règles (comme un item par exemple), c'est aux FE de les utiliser. L'expérience de POMS a confirmé ce choix.

Il est temps d'indiquer des futurs travaux possibles sur ce sujet. Un travail très important est de concevoir et de réaliser une interface utilisateur adaptée, l'interface graphique est pratiquement obligatoire pour l'utilisation de POMS. Un autre travail consiste à améliorer la conception selon des expériences d'utilisations.

Chapitre 9
Un Système de Vote en Mode Mesagerie

Dans ce chapitre, nous allons présenter un exemple de communication de groupe et son archiveur de messages.

1. Introduction

Dans notre société, on a souvent besoin de connaître les opinions ou les idées des gens sur certains sujets, sondages et élections sont alors utilisés. Dans ce chapitre, on utilise la terminologie **vote** pour les désigner.

Par exemple, dans un département pédagogique, un *vote* est souvent nécessaire pour nommer le comité d'enseignement; le responsable diffuse un appel de candidatures pour distribuer les responsabilités des cours, etc.

En effet, un vote est une activité de coopération entre celui qui demande le vote (on l'appelle désormais **Initiateur**) et ceux auxquels l'initiateur demande leurs opinions (on les appelle **Electeurs**). Par exemple, les électeurs pourraient avoir besoin de savoir si leurs votes sont secrets ou non, dans le premier cas, ils doivent en être assurés. La messagerie électronique inter-personnel ne pourra supporter directement ce genre d'activité (cf. [Smith 89]). Notre objectif est donc de construire un système de vote basé sur la messagerie X.400 MHS. Ce système doit fournir les outils de coopération qui permettent aux utilisateurs d'organiser et de participer aux activités de vote en échangeant des messages.

Dans le texte qui suit, nous allons d'abord présenter différents types de votes, puis décrire le modèle d'activité de vote, nous présentons enfin la réalisation du modèle dans le contexte X.400.

2. Types de Votes

Un vote peut être *anonyme*, i.e. les votes individuels sont secrets; ou *public*, i.e. les noms des électeurs sont publiés avec leurs votes. Les paramètres qui caractérisent une activité de vote sont:

- **Anonyme**: un vote est *anonyme* si les noms d'électeurs ne sont pas inclus dans des résultats du vote. C'est souvent le cas dans les élections.
- **Résultat_Partiel**: Les résultats partiels peuvent être obtenus ou non. Par *Résultat Partiel*, on veut dire des résultats non-définitifs à un instant donné avant la fin du vote. Ceci peut être utile pour un vote destiné à la négociation. Suite à un résultat partiel, l'initiateur peut rappeler les électeurs à voter.
- **Modifiable**: ceci spécifie si les électeurs peuvent modifier leurs réponses. Un vote modifiable sera utile pour des négociations. Par exemple, dans une collection d'opinions pour la date d'une réunion, un électeur peut changer sa décision quand il a un rendez-vous plus prioritaire.

La combinaison de ces paramètres donne des types de votes. Le type d'un vote doit être indiqué par l'initiateur quand il initialise l'activité de vote.

3. Modèle d'activité de vote

Une activité de vote est le déroulement complet d'un vote. Une activité de vote est un exemple de travail en groupe, nous allons utiliser la méthode AMIGO (cf. [PANKOKE 89]) pour décrire le modèle (**Remarque:** Nous utilisons seulement la méthode, i.e. la syntaxe de définition ne sera pas strictement respectée.). Il s'agit de définir, pour l'activité en question, les différents rôles et leurs fonctions, les messages échangés entre ces rôles, et les règles qui définissent leur comportement.

3.1. Rôles

Dans chaque activité de vote, on peut identifier 4 rôles (figure 9-1), ce sont:

- **Initiateur**: c'est celui qui demande le vote et donc initialise le vote avec des questions à poser.
- **Electeurs**: ceux à qui ont demandés de répondre aux questions, posées par l'initiateur.
- **Publishers**: ceux à qui (où) le resultat final sera communiqué (ou publié).
- **Coordinateur**: le coordinateur assure le bon déroulement de l'activité de vote, il sert d'intermédiaire entre l'initiateur, les électeurs et les publishers. En réalité, dans la plupart des cas, celui qui demande un vote ne communique pas directement avec les électeurs.

Remarque: dans le texte suivant, on utilisera souvent le nom d'un rôle pour désigner celui (ceux) qui joue(nt) le rôle.

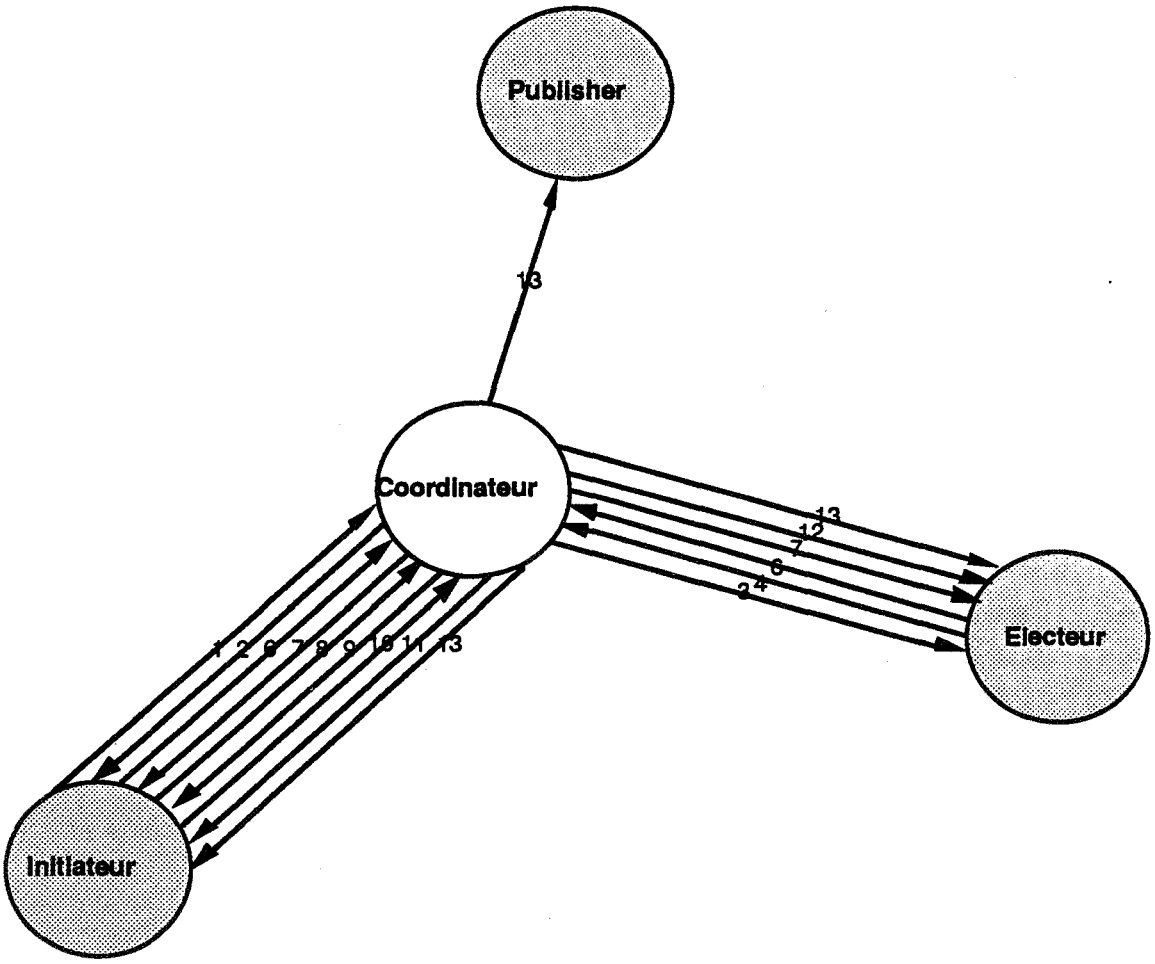
Logiquement, une activité de vote comprend, *un initiateur*, *un coordinateur*, *un* ou *plusieurs électeurs* et *zéro*, *un* ou *plusieurs publishers*.

3.2. Fonctions des Rôles

La fonction essentielle de l'**initiateur** est d'initialiser l'activité de vote. Il s'agit de déterminer les participants et leurs rôles dans l'activité en indiquant le groupe (voir les sections prochaines) concerné; d'indiquer le type de vote; de spécifier le(s) question(s) et le(s) réponse(s) possible(s); et de donner la date limite avant laquelle les réponses doivent parvenir, i.e. la date où les réponses seront comptées. L'initiateur contrôle également l'activité de vote en modifiant la date limite, ou en demandant au coordinateur de diffuser un rappel aux électeurs qui n'ont pas encore voté. L'initiateur réalise ces fonctions en communiquant avec le coordinateur de l'activité.

La fonction essentielle des **électeurs** est de *voter*, i.e. donner un bulletin de réponse rempli au coordinateur. Ceci consiste à faire un ou plusieurs choix pour un sous-ensemble de questions posées, avec éventuellement des explications de ces choix. Il peut aussi modifier sa réponse ou demander des résultats partiels si le type du vote le permet. Un électeur pourrait bien sûr refuser de voter.

La fonction des **publishers** est de recevoir le résultat final du vote. Dans le cas où un publisher est un Bulletin-Board, le resultat final est *publié*.



Note:

1: Initialiser
2: Confimer l'initialisation
3: Diffuser le bulletin de vote
4: Voter (répondre aux questions)
5: Confirmer un vote
6: Demander un résultat partiel
7: Distribuer un résultat partiel
8: Demander une diffusion de rappel
9: Confirmer une demande de diffusion de rappel
10: Modifier les paramètres
11: Confirmer une modification de paramètres
12: Diffuser un rappel
13: Diffuser le résulat final

Figure 9-1. Rôles et leur coopération

La fonction essentielle du **coordinateur** est de coordonner l'activité de vote. Il reçoit la demande d'initialisation de l'initiateur, envoie un bulletin de vote à chaque électeur et collecte les bulletins de réponses. Il obtient ensuite le résultat final à partir des bulletins de réponse reçus et l'envoie à tous les autres participants. Outre les fonctions précédentes, le coordinateur effectue aussi les fonctions suivantes:

- Confirmer la demande de l'initialisation.
- Confirmer les réponses des électeurs.
- Diffuser un rappel peu avant la date limite, ou par une demande de l'initiateur aux électeurs qui n'ont pas encore voté.
- Envoyer des résultats partiels à l'initiateur ou aux électeurs sur leur demande.

La figure 9-1 montre les fonctions de chaque rôle.

3.3. Processus de vote

On divise le processus d'une activité de vote en trois *phases*, i.e. *Initialisation*, *Négociation*, et *Terminaison*. Pendant la phase d'initialisation, l'initiateur envoie un message d'initialisation au coordinateur pour demander d'initialiser l'activité de vote, l'initialisation du groupe de travail correspondant doit être effectuée extérieurement dans un système de directory (DS) [X500·88]. Les bulletins de vote et de réponse envoyés et les autres échanges (obtention de résultats partiels, contrôles de paramètres, etc) constituent des activités de la phase Négociation. Cette phase commence après la confirmation de l'initialisation et se termine quand la date limite est dépassée. Pendant la phase de Terminaison, le coordinateur de l'activité de vote distribue le résultat final à tous les autres participants. Cette phase termine l'activité (figure 9-2).

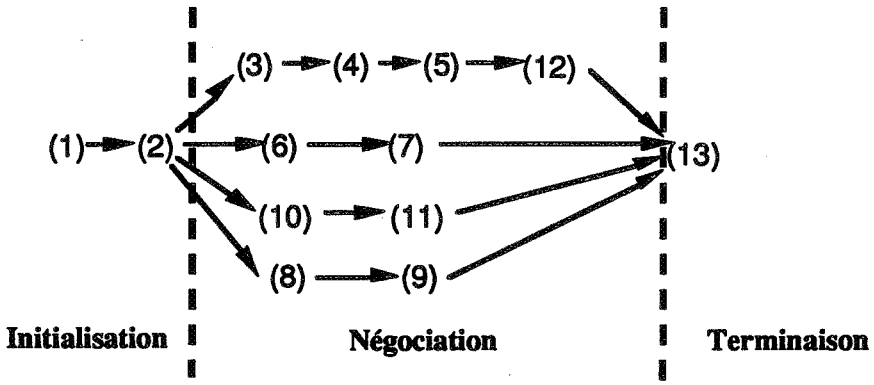


Figure 9-2. Processus de Vote

3.4. Messages Echangés

Les participants à une activité de vote échangent des messages pour accomplir leurs tâches. La sémantique d'un message est spécifiée par son type, ci-dessous sont listés les types de messages utilisés dans une activité de vote. Leurs définitions sont omises de cette thèse, le lecteur intéressé peut trouver les détails dans [You 89e].

INITIALISATION

Un message du type INITIALISATION est envoyé de l'initiateur au coordinateur pour demander l'initialisation du processus de vote. Ce message contient des paramètres nécessaires pour initialiser une activité de vote, comme par exemple, la description de vote, le type de vote, les questions à poser, le groupe de participants, etc.

Un exemple de ce type de message est donné dans la figure 9-3.

```
Identificateur: <1356.leonardon@emsesmc.emse.fr
Type: INITIALISATION
Expéditeur: Jori Leonardon <leonardon@emse.fr>
Destinataires: Agent de Vote EMSE <avemse@emse.fr>
Type-de-Vote: Public,Resultats-partiels,Modifiable
Groupe-de-Travail: Groupe-enseignant
Date-de-Limite: 09-05-89
Bulletin-de-Vote:
 Description: Pour organiser l'enseignement
informatique 1988-1989, peux-tu m'indiquer quels
sont les cours dont tu Souhaites, Acceptes ou
Refuses la responsabilité.
 Question 1: Génie Informatique (1 trim. 30u)
  Choix Maxmum: 1
  Choix:
    1: S
    2: A
    3: R

 ...

 Question 20: Cours Images (DEA, 20u)
  Choix Maxmum: 1
  Choix:
    1: S
    2: A
    3: R
```

Figure 9-3. Un exemple des messages INITIALISATION

CONFIRMATION-INITIALISATION

Ce type de messages est utilisé par le coordinateur pour confirmer une demande d'initialisation de vote, il est donc envoyé du coordinateur vers l'initiateur. Une confirmation positive contient un identificateur de l'activité initialisée par la demande. Cet identificateur sera utilisé dans tous les autres messages échangés pendant toute la durée de l'activité pour référer à l'activité. Ce champ est présent si et seulement si l'initialisation réussit.

BULLETIN de VOTE

C'est le bulletin de vote envoyé par le coordinateur à chaque électeur pour lui demander une réponse. Logiquement, le coordinateur envoie un message de ce type pour chaque électeur.

BULLETIN de REPONSE

Les messages de ce type sont envoyés par les électeurs pour répondre aux bulletins de vote.

CONFIMATION de REPONSE

Le coordinateur envoie un message de ce type à son expéditeur (i.e. l'électeur), pour confirmer une réponse reçue. Elle peut être positive ou négative.

RAPPEL

Les messages de ce type sont envoyés par le coordinateur aux électeurs qui n'ont pas encore voté. Un rappel est envoyé par le coordinateur juste avant la date limite ou sur la demande de l'initiateur.

CONTROLE
L'initiateur envoie un message de ce type pour demander un rappel et/ou pour modifier la date limite du vote. La deuxième fonction est intéressante quand il s'agit, par exemple, d'un "Appel à Communications".

CONFIRMATION de CONTROLE
Le coordinateur confirme un message de CONTROLE en envoyant un message de ce type à l'initiateur.

DEMANDE d'un RESULTAT PARTIEL
Les électeurs ou l'initiateur envoient des messages de ce type pour obtenir un résultat partiel.

RESULTAT-PARTIEL
Un message du type RESULTAT-PARTIEL est envoyé par le coordinateur à l'initiateur ou à un électeur pour confirmer sa demande de résultat partiel.

RESULTAT FINAL
Un résultat final est envoyé à tous les participants par le coordinateur.

3.5. Règles

Pour chaque activité, le modèle d'AMIGO (cf. [Pankoke-89]) associe un ensemble de règles qui définissent les fonctions des différents rôles dans l'activité. L'activité de vote a les règles suivantes:

- L'initiateur demande l'initialisation de l'activité.

Il spécifie le type de vote, les questions à poser, la date limite et le groupe de travail.

- L'initiateur demande un résultat partiel.

Il peut demander à n'importe quel moment le résultat partiel si le type de vote le permet.

- L'initiateur demande de modifier la date limite.

Il peut retarder la date limite.

- L'initiateur demande de diffuser un rappel.

Le rappel sera diffusé seulement aux électeurs qui n'ont pas voté.

- L'initiateur reçoit le résultat final.

Ce résultat est envoyé automatiquement par le coordinateur, il contient également l'explication.

- Le coordinateur reçoit la demande d'initialisation et la confirme.

Il doit examiner la demande afin de décider s'il l'accorde ou pas.
Si oui, il affecte un identificateur unique pour cette activité de vote et initialise ses données locales, par exemple, l'archiveur, etc.

- Le coordinateur diffuse un bulletin de vote à chaque électeur.

Les questions sont posées par l'initiateur dans son message de demande d'initialisation.

- Le coordinateur reçoit les réponses (bulletins de réponse) des électeurs et les conserve à sa façon. Il envoie ensuite une confirmation (accusé de réception).
- Le coordinateur reçoit la demande du résultat partiel et confirme la demande.

La confirmation contient le résultat partiel, s'il est autorisé.

- Le coordinateur envoie un rappel aux électeurs qui n'ont pas encore voté.

Le rappel est, soit généré juste avant la date limite automatiquement, soit demandé explicitement par l'initiateur.

- Le coordinateur distribue le résultat final à tous les participants après la date limite.
- Un électeur reçoit un bulletin de vote et envoie une réponse (i.e. voter) au coordinateur.
- Un électeur demande un résultat partiel au coordinateur, il reçoit ensuite une confirmation (accusé de réception).

L'électeur peut ne pas voter.
Une confirmation positive contient un résultat partiel.

- Un électeur reçoit le résultat final après la date limite.
- Un publisher reçoit le résultat final après la date limite.

Remarque: Toutes les demandes de l'initiateur et des électeurs sont envoyées au coordinateur. Tous les résultats sont recensés et envoyés par le coordinateur.

4. Intégration dans l'Environnement X.400

Dans cette section, on présente la réalisation d'un système de vote basée sur la messagerie X.400.

4.1. Le Modèle Fonctionnel

D'après la description de la section précédente, on peut décomposer un système de vote suivant le modèle client-serveur. Un système de vote se compose d'un fournisseur du service de vote (Agent de Service) et de clients qui consomment le service fourni par le fournisseur (Agent d'Utilisateur). La figure 9-4 montre ce modèle.

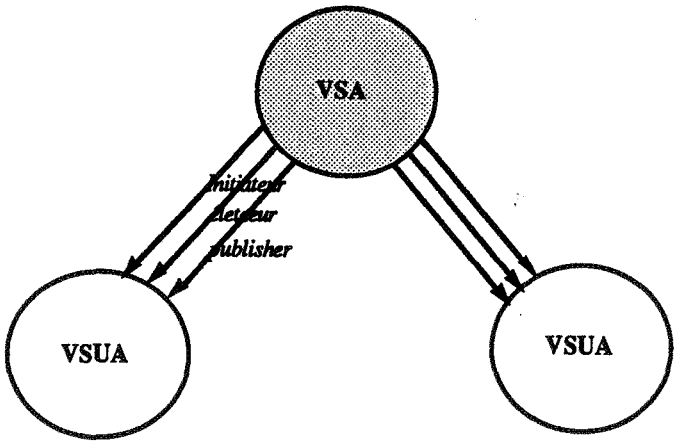


Figure 9-4. Le Modèle Fonctionnel

Un système de vote se compose d'un **VSA** (Voting Service Agent) et plusieurs **VSUA** (Voting Service User Agent). Le VSA est le *coordinateur* de vote et fournit le service via 3 portes: **initiateur, électeur** et **publisher**. Les VSUA fournissent des interfaces aux utilisateurs finaux et leurs permettent d'accéder au VSA via les portes appropriées. Avec les utilisateurs finaux, les VSUA remplissent les rôles "*initiateur*", "*électeur*" et "*publisher*".

Un utilisateur jouant le rôle d'*initiateur* accède au service fourni par le VSA en utilisant un VSUA via la porte "initiateur". Un utilisateur étant *électeur* accède au service en utilisant un VSUA via la porte "électeur".

Un *publisher* accède au service (reçoit le résultat final) via la porte "publisher". La description formelle (en ASN.1) du modèle et le protocole entre VSA et VSUA sont omis dans cette thèse.

Il faut noter que les agents d'une application de communication de groupe (Group Communication Applications) ont souvent besoin d'accéder au service de directory (pour connaître informations sur le groupe et les membres du groupe) et au service de stockage (pour stocker et retrouver des messages échangés).

4.2. Modèle Architectural dans le Contexte X.400

Principalement, le système X.400 a deux sortes d'entités (ou composants). *MTA* forment la couche transport (MTS) du système X.400, ils permettent de transférer des messages "*bruts*" en suivant le protocole **P1**. D'autres composants sont des agents utilisateurs (UA) qui utilisent (plus exactement, qui permettent aux utilisateurs finaux d'utiliser) le service MTS. Ils sont regroupés en classes, chaque classe d'UA communiquent entre elles suivant leur propre protocole, elles réalisent un service spécial au-dessus du MTS. Un exemple d'UA est IPM-UA (Inter-Personel Messaging User-Agent), les IPM-UA communiquent suivant le protocole **P2**, ils fournissent aux utilisateurs finaux le service de courrier personnel. C'est le seul service normalisé au niveau UA par CCITT et ISO.

Le principe d'intégration d'une application au système X.400 est donc de construire des agents de l'application comme UA spéciaux ayant leur(s) propre(s) protocole(s) de communication, tout en gardant le MTS comme le support de transport. Dans notre cas, on doit donc construire le VSA et les VSUA comme l'UA du MTS. Un VSUA communique avec le VSA en suivant le protocole **Pv**, qui spécifie les messages et leurs formats d'échange.

Avec les accès au service de directory (DS) et celui de messagerie, on obtient l'architecture de notre système, voir figure 9-5. Il faut noter qu'on traite l'archiveur de messages comme un composant interne de VSA. En fait, un VSA est un gestionnaire de stockage de messages, tous les messages reçus sont des requêtes (initialiser, stocker, etc) et les messages envoyés sont les réponses aux requêtes.

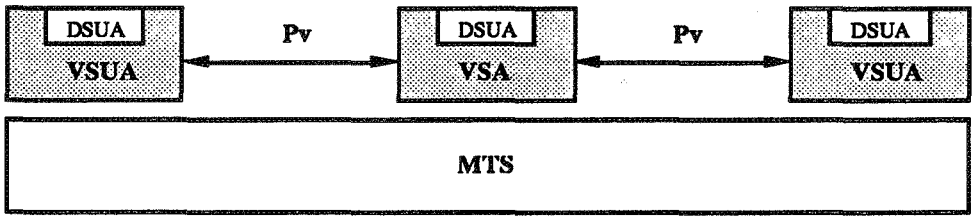


Figure 9-5. Intégration dans l'Environnement

4.3. Utilisation du DS

Généralement tous les utilisateurs et les agents d'un système distribué doivent accéder au service de directory. On présente donc dans cette section notre *utilisation* du service de directory.

Chaque activité de vote a un groupe d'utilisateurs qui participent à l'activité, comme toutes les activités en groupe. Le groupe est enregistré dans le système de directory avec les informations suivantes (elles sont représentées comme attributs):

- Les membres du groupe, i.e. les participants du groupe. Notons qu'un membre peut être un humain ou un programme, il est identifié par son nom DS.
- Les rôles du groupe, i.e. *Initiateur*, *Coordinateur*, *Electeur* et *Publisher*.
- L'affectation des rôles aux participants, ceci définit la composition du groupe. Dans un groupe d'activités de vote, il doit y avoir un initiateur, un coordinateur, un ou plusieurs électeurs et zéro ou plusieurs publishers.
- Autres informations.

Le DS permet également d'obtenir l'adresse de chaque participant du groupe.

Donc, l'initialisation d'un groupe de vote consiste à créer un groupe de travail dans le DS en donnant:

- les membres du groupe.
- les rôles dans l'activité.
- l'affectation des rôles.
- les autres informations facultatives.

Le groupe de travail d'une activité de vote doit être initialisé avant l'initialisation de l'activité par son initiateur.

4.4. Le Protocole Pv et Les Messages Echangés

La communication entre un VSA et un VSUA se fait en suivant le protocole *Pv* qui définit les données échangées entre ces deux objets. Comme indiqué dans les sections précédentes, ces données sont spécifiées comme des messages du modèle d'activité. Donc le protocole Pv définit la représentation en X.400 de ces messages échangés .

Il est clair que l'on représente chaque message de vote comme un message de X400. Cependant deux choix sont possibles:

- On représente les messages de vote directement par des messages P1, i.e. on définit un nouveau type de contenu de messages P1 pour les messages du protocole Pv.
- On utilise le message P2 (Inter-Personel Message) pour contenir les messages échangés.

L'approche P2 permet de réutiliser des IPM-UA existant comme VSUA (voir la section prochaine). De plus, la recommandation X400.88 a fourni aux utilisateurs la possibilité d'avoir les attributs additionnels. Donc nous avons décidé de représenter les données du protocole Pv en messages P2, ce qui consiste à projeter les attributs (ou champ) d'un message du modèle (i.e.

définir dans le modèle d'activité) en attributs d'IPMessages ou le corps du message P2 correspondant, tout en conservant le plus possible les sémantiques des attributs P2. Les détails sont omis ici, nous donnons simplement un exemple d'un message INITIALISATION représenté en P2 dans la figure 9-6.

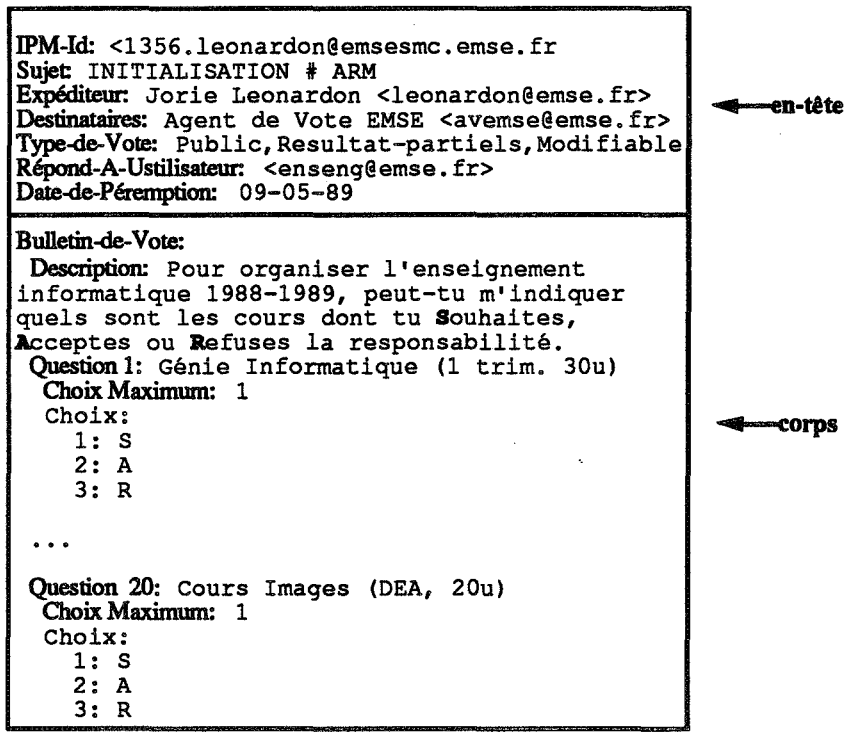


Figure 9-6. Représentation d'un message INITIALISATION en P2

4.5. Agent Usager de Service de Vote

Un VSUA fournit une interface pour accéder au service de vote et/ou aux autres services comme DS, il est réalisé comme un UA du MTS.

En pratique, deux méthodes pour réaliser un agent utilisateur existent:

1. Utiliser les IPM-UA existants. Cela implique l'utilisation de messages inter-personnels et le protocole P2. Les données de protocole du vote Pv (i.e. les messages échangés) sont donc transférées par les messages inter-personnels. L'avantage de cette approche est qu'il est simple de réaliser les agents utilisateurs de l'application (VSUA), et qu'il y aura la même interface utilisateur pour le courrier personnel et le service de vote. La construction d'un agent de service est suffisant pour que le système soit disponible et utilisable dans un environnement aussi vaste que le réseau de la messagerie X.400. En fait, il n'est pas toujours évident d'imposer un service quand il faut installer un nouveau logiciel. Cependant l'utilisation d'IPM-UA comme interface d'utilisateur a aussi des inconvénients. Les UA ne comprennent pas la sémantique de l'application. Par conséquent, ils ne peuvent fournir une interface appropriée ni garantir la conformité des données transmises au protocole de l'application.

2. Construire des UA spécialisés. Aujourd'hui, les avantages de cette méthode sont les inconvénients de la méthode précédente, de même pour ces inconvénients. Cette méthode sera supérieure,
(1) si un standard de l'interface de programmation d'application (API) existe pour X.400 MTS, et (2) si un standard de l'interface utilisateur est parfaitement défini. Ce dernier semble plus difficile.

Notre représentation du protocole (en message P2) permet d'utiliser les deux méthodes de réalisation. Quand on utilise directement un UA-IPM, il est nécessaire de posséder un moyen d'accès au service de directory (en ayant, par exemple, un agent utilisateur du service de directory DSUA).

4.6. Agents de Service de Vote

Un VSA est un agent centralisé qui contrôle une ou plusieurs activités de vote et joue le rôle de *coordinateur dans* ces activités. L'identificateur unique de l'activité, porté dans chaque message échangé, permet au VSA de reconnaître l'activité concerné par le message.

Un VSA accède à un DS (Directory System) et à un MS (Archiveur de Messages) pour obtenir les informations concernant le groupe de travail et stocker les messages échangés (figure 9-7).

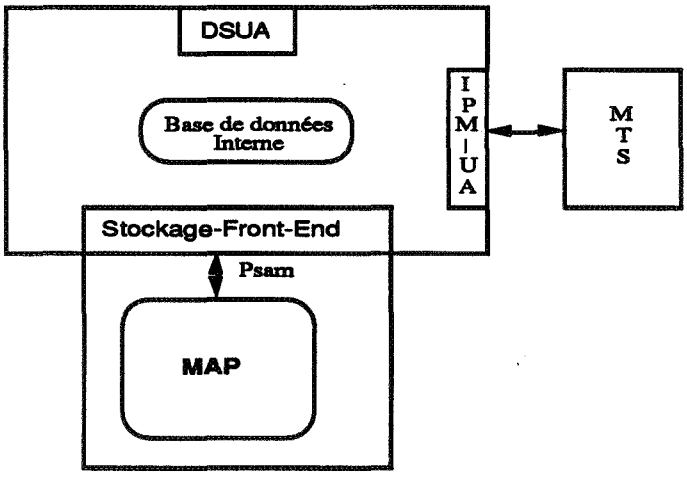


Figure 9-7. VSA

On donne ci-dessous quelques exemples du fonctionnement d'un VSA.

- Le VSA reçoit un message d'INITIALISATION, il examine d'abord la conformité du message, i.e. si le message est syntaxiquement correct. Puis il obtient le nom de l'expéditeur (i.e. initiateur), et le groupe de travail. Il vérifie si l'expéditeur joue le rôle *initiateur* dans le groupe en consultant le DS, les participants et les rôles qu'ils jouent peuvent ainsi être obtenus. Ensuite si tout est correct, le VSA crée une nouvelle activité, ce qui consiste à définir un identificateur unique de l'activité, initialiser la partie correspondante dans le MS (voir la section suivante) et les autres bases de données

internes. A la fin, le VSA envoie une confirmation (positive si tout va bien sinon négative) à l'expéditeur du message INITIALISATION.

- Le VSA construit et envoie un message BULLETIN-VOTE (requête de vote) aux électeurs identifiés dans le groupe.

- Le VSA reçoit un message du type BULLETIN-REPONSE, il vérifie d'abord si l'expéditeur est un électeur de l'activité, puis il vérifie la conformité du message, notamment les réponses. Ensuite il stocke ce message dans le MS en vérifiant si une réponse de cet électeur existe déjà et si le vote est *modifiable*. Tout cela détermine la nature de la confirmation qui sera envoyée à l'expéditeur du message BULLETIN de REPONSE.

- Les messages BULLETIN-REPONSE reçus après la date limite seront abandonnés, le VSA enverra une confirmation négative à chacun des expéditeurs de ces messages.

- Un message RAPPEL sera envoyé à tous les électeurs qui n'ont pas encore voté, juste avant la date de clôture du vote.

4.7. Archiveur de Messages

Chaque VSA a un archiveur de messages dédié qui stocke tous les messages reçus par le VSA, en tant que coordinateur. L'archiveur de message est construit en ajoutant un FE (Vote_FE) au dessus de MAP (figure 9-8). Il permet au VSA d'organiser et de rechercher des messages. Les messages sont groupés par les activités auxquelles ils appartiennent, chaque activité correspondant à un contexte de MAP (un sous-contexte direct de top-contexte), et chaque message de l'activité est représenté par un item associé au contexte.

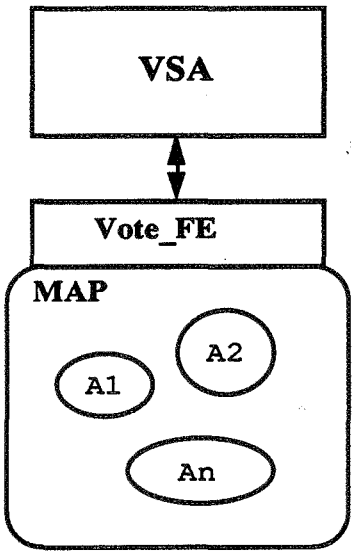


Figure 9-8. Le Schéma Logique

Les messages d'une activité de vote sont stockés et organisé de la façon suivante (Figure 9-9):

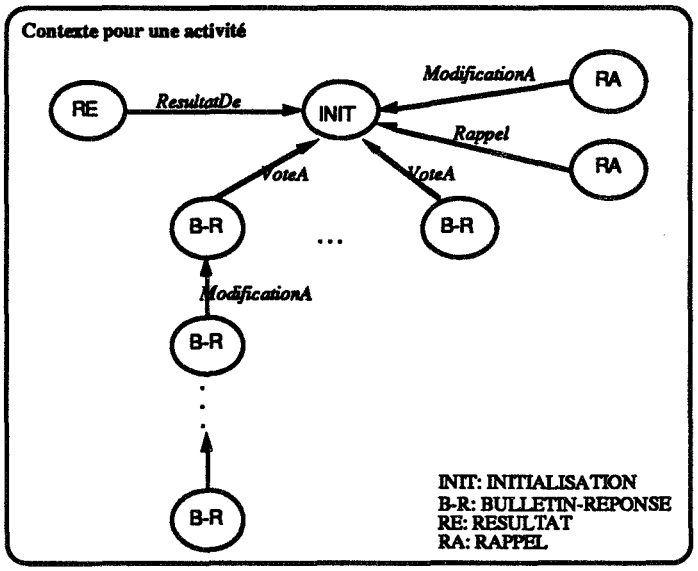


Figure 9-9. Organisation des Messages dans l'Archiveur

- Les messages concernant une activité sont stockés comme les membres du contexte correspondant à l'activité.
- Les messages de BULLETIN-(de)-REPONSE du même électeur sont liés linéairement par des liens *ModificationA* suivant l'ordre chronologique de leurs soumissions.
- Le premier dans la chaîne de messages BULLETIN-REPONSE d'un électeur a un lien *VoteA* au message INITIALISATION. Ainsi, on peut obtenir facilement la liste de ceux qui ont voté et la réponse exacte de chaque électeur.
- Chaque RAPPEL a un lien *Rappel* entrant au message INITIALISATION.
- Les messages RAPPEL qui modifient les paramètres de l'activité sont liés au message INITIALISATION par des liens *ModificationA*.
- Un résultat représentant la somme des votes reçus est mis à jour chaque fois qu'un BULLETIN-REPONSE est accepté, il est obtenu d'après le résultat précédent et la réponse reçue. Ce résultat est lié avec le message INITIALISATION par le lien *RésultatDe*. Il devient le résultat final quand le processus de vote se termine.
- Les messages pour obtenir des résultats partiels ne sont pas stockés, puisqu'ils ne représentent aucune valeur.
- Deux attributs locaux sont ajoutés à chaque message stocké, ils sont *statut* et *Raison-de-Refus*. Ils indiquent si un message reçu est accepté ou refusé, dans ce dernier cas, ils indiquent la raison du refus.

Puisque MAP supporte les divers types de messages et la recherche selon les liens, il est facile de réaliser les fonctions nécessaires pour le VSA, que nous décrivons comme suite:

- Initialiser une activité: ceci consiste à créer un nouveau contexte dans MAP et à lui associer un Contrôleur. Le message INITIALISATION est stocké dans le contexte.

L'identificateur du contexte est donné comme résultat, ce qui permet au VSA de fabriquer l'*identificateur-d'Activité.*

- Stocker un nouveau message: le message est stocké dans la base d'information de MAP et associé au contexte correspondant à l'activité auquelle le message appartient. Selon le type du message et son statut, un lien approprié sera ajouté, le résultat sera mis à jour.
- Lire un ou plusieurs messages: cette fonction peut être divisée en plusieurs sous-fonctions.
- Lire un message d'après son identificateur interne dans MAP, i.e. d'après l'OID de l'item correspondant.
  - Lire le message INITIALISATION de l'activité. Le FE peut soit garder l'identificateur du message, soit effectuer une recherche d'après le type de message.
  - Lire le résultat: la solution est la même pour la fonction précédente, on peut, en plus, suivre le lien depuis le message INITIALISATION.
  - Lire les réponses d'un utilisateur donné: cela peut être fait par la recherche d'après le lien *VoteA* et l'*expéditeur* des messages.
  - Suivre un lien en bon sens ou en sens inverse: il suffit de lire les liens sortants et les liens entrants de l'item correspondant dans MAP. Cette opération permet de lire n'importe quel message en suivant son lien avec le message INITIALISATION.
- Supprimer une activité: le contexte correspondant et les messages seront supprimés et effacés. Cette opération n'est pas exécutée immédiatement après la fin de l'activité, puisque les messages peuvent servir comme moyen de vérification.

Les règles d'accès à l'archiveur peuvent être facilement exprimées comme Access Control List (ACL) de MAP:

- Seul *initiateur* de l'activité peut stocker un message INITIALISATION et RAPPEL.
- *Electeurs* de l'activité ne peuvent stocker que des message du type BULLETIN-REPONSE.
- Le *coordinateur* de l'activité a tous les droits, i.e. a le droit d'effectuer toutes les opérations.

On peut modifier cet ACL dynamiquement quand un électeur a stocké un BULLETIN-REPONSE et que le vote n'est pas modifiable, on ajoute dans ce cas une exception pour interdire le stockage d'autres réponses par ce même électeur.

5. Une Extension: Vote Ouvert

Dans certains cas, il est très intéressant d'avoir des votes dont les électeurs ne sont pas prédéterminés, tous ceux qui le veulent peuvent voter. Par exemple, un sondage ouvert consiste à savoir combien de personnes seront intéressées par la création d'un nouveau "news-group".

Dans ce but, un groupe ouvert avec un nouveau rôle *électeur-secondaire* est nécessaire pour une activité de vote, le fonctionnement du rôle *publisher* a aussi besoin d'être modifié. Un autre point spécifique du groupe *ouvert* est que tout le monde en est membre et que chacun peut jouer le rôle *électeur-secondaire.*

Les bulletins de vote ne sont pas distribués directement à ceux qui jouent le rôle d'*électeur-secondaire*, contrairement à ceux qui sont *électeurs*. Ils peuvent obtenir le bulletin de vote chez les *publishers* auxquels le bulletin est envoyé automatiquement par le coordinateur. Cependant, le *coordinateur* accepte des bulletins réponse de tout le monde quel que soit le moyen d'obstention de son bulletin de vote. Les électeurs-secondaires ont le droit de demander un résultat partiel mais pas de rappel, puisque l'on ne peut déterminer tous les électeurs-secondaires. Tous les électeurs-secondaires ayant voté recevront le résultat final. Les*publishers* sont généralement des bulletin-boards ou conférences. Le bulletin de vote, des rappels et le résultat final sont envoyés aux publishers.

La nature d'un groupe (ouvert ou non) doit être indiquée dans le groupe lui-même par un attribut (dans le DS). L'initiateur doit aussi indiquer la nature du vote dans son message INITIALISATION.

Le processus de vote sera identique, quelle que soit sa nature: *ouvert* ou *fermé*.

6. Conclusion

Dans ce chapitre, nous avons présenté un système de vote aidant les utilisateurs coopérant dans des activités de vote. Ce système de vote supporte non seulement des activités comme l'élection d'un président d'un comité de recherche, mais aussi des activités de collections d'opinions ou d'idées (sondages), ainsi que d'autres activités similaires. Ce système est basé sur la messagerie X.400, il n'impose aucune contrainte géographique à ces utilisateurs s'ils peuvent accéder à la messagerie X.400.

SAM message store prouve, une fois de plus, sa flexibilité et son adaptabilité aux applications de MHS et GC.

En effet, notre système est conçu pour fonctionner dans un environnement amical ou académique. Les utilisations dans des environnements fortement administratifs ayant besoin d'une grande sécurité (e.g., élire le président de la république) ne sont pas encouragées, à cause des problèmes de sécurité dans la messagerie X.400 (cf. [Pays 89]), du système de DS et du système informatique en général.

Le format fixé du bulletin de vote pourra poser des problèmes à certains usagers qui ont besoin de données plus complexes (graphique, son, image, etc). L'absence d'UA dédiés augmente la possibilité d'erreurs de la part d'utilisateurs. Cela nous indique les améliorations à apporter à notre système, des expériences acquises lors d'utilisations du système nous permettront certainement de l'améliorer dans le futur.

Conclusion

Dans cette thèse ont été présentées la conception et la réalisation d'un archiveur de messages général: SAM. Il supporte diverses applications MHS (Message Handling System) et GC (Group Communication), des boîtes aux lettres personnelles jusqu'aux conférences réparties. Se basant sur un modèle conceptuel d'applications MHS et GC et un modèle architectural très flexible, SAM peut être utilisé à la fois comme un outil de réalisation et de conception d'archiveurs particuliers d'applications. L'utilisation de SAM permet d'enrichir de façon notoire des applications MHS et GC. C'est sans doute l'argument important qui a permis d'envisager un projet industriel "pilote" patronné par la DAII, sur la base de SAM.

L'originalité de ce travail réside dans sa "**généralité**". A notre connaissance, c'est le premier système d'archivage qui supporte diverses applications MHS et GC ayant des besoins variés. Une telle généralité provient directement du choix d'un modèle d'architecture approprié, qui nous permet de séparer les fonctions communes des fonctions spécifiques. Cette architecture décompose un archiveur en un noyau commun et des unités fonctionnelles spécifiques. Elle nous fournit non seulement la possibilité d'avoir une architecture générale, mais représente aussi beaucoup d'avantages du point de vue "génie logiciel", tel que la *réutilisabilité* et l'*extensibilité*.

Quelques points intéressants dans la conception de MAP me semblent mériter d'être mentionnés.
Le premier point est son modèle de données: il est à la fois orienté-objet et hypertexte, et possède également les possibilités de recherche des bases de données relationnelles. La combinaison de ces propriétés nous a donné un modèle très puissant.
Prédéfinir un ensemble d'attributs constitue le deuxième point intéressant; ceci permet non seulement une réalisation simple et performante, mais aussi l'utilisation de MAP comme un outil/modèle de conception. Il nous fournit de plus la possibilité de traiter les sémantiques particulières des attributs.
Le troisième point réside dans le support de la distribution et de la synchronisation d'informations. L'introduction et l'utilisation des concepts tels que "*objets incomplets*", "*objets locaux ou globaux*", et "*attributs locaux ou globaux*" me semblent nouvelles dans ce domaine, ainsi que les mécanismes associés.
Enfin, le dernier point est la particularité du mécanisme d'historique, il est conçu afin d'être utilisé pour supporter la récupération après des incidents, pour pouvoir revenir en arrière, pour gérer des versions de façon simple, mais aussi pour supporter la distribution et la synchronisation d'informations. Sur ce dernier point, je dois confirmer mon effort particulier pour concevoir un ensemble cohérent d'opérations idempotentes, qui a aussi influencé le modèle lui-même.

Cette thèse a été beaucoup influencée par les travaux du groupe AMIGO+, dont l'auteur était membre. Le modèle conceptuel des applications de SAM est dérivé directement de celui proposé par AMIGO+ ([Benford 88]), ainsi que les utilisations du service d'annuaire. Le modèle de stockage et d'archivage de SAM peut être vu comme une généralisation de celui d'AMIGO+

([Smith 89] et [Palme 88]). Ils proposent en effet une autre approche intéressante de la conception d'archiveurs de messages. Leur approche consiste à concevoir un système d'archivage en imposant une stratégie de distribution fixe. Du point de vue de notre modèle, l'approche combinerait les FE avec MAP. L'avantage de cette approche est son efficacité et son "haut niveau": une application n'a pas besoin de s'occuper elle-même de l'aspect distribution. L'inconvénient est qu'il n'existe malheureusement pas une stratégie universelle de distribution; cela limite largement son champ d'application. Comparé à cette approche, notre système fournit plutôt des supports de "bas niveau"; cela peut être cependant compensé par les FE qui fournissent des fonctions de "haut niveau".

Bien que nous ayons atteint entièrement nos objectifs de départ, plusieurs évolutions sont envisageables.

La première évolution possible consiste à enrichir les sémantiques des attributs prédéfinis. Par exemple, la version actuelle de MAP traite les liens "InReplyTo" et "VersionOf" de façon identique. Une technique est de classer plus finement ces attributs, comme dans CRMM pour les liens. Cependant cela nécessite une étude plus approfondie des applications MHS et GC.

L'utilisation de SAM pour les messages multi-média ([Huitema 89]) est un point très intéressant. Du point de vue du stockage et de l'archivage, la particularité des messages multi-média réside dans leur taille immense de représentation en mémoire. MAP fournit actuellement les facilités suivantes pour stocker et gérer les messages multi-média:

- MAP n'impose aucune limite sur la taille d'un message, nous pouvons donc stocker des messages multi-média de n'importe quelle taille dans la limite des mémoires réellement disponibles.
- MAP permet d'extraire une ou plusieurs parties de contenu (CP) d'un message. Cela permet aux utilisateurs de choisir les parties de contenu *interprétables*. Cette facilité est particulièrement utile lorsque le lecteur d'un message multi-média ne dispose pas, au moment de sa lecture, de moyen matériel ou logiciel pour interpréter les données autres que textes. Les items incomplets permettent même de ne pas stocker physiquement les parties ininterprétables.
- MAP accepte le type de parties de contenu *brut*: ceci permet de stocker des données de n'importe quelle nature. L'intérêt apporté est évident lorsqu'on considère le nombre de natures de données existantes (images, animations, sons, vidéos, etc).

Cependant il est possible d'enrichir MAP pour mieux supporter la messagerie électronique:

- La structure des contenus d'items dans MAP est un ensemble ordonné ou non de parties de contenu. MAP ne supporte pas "directement" d'autres structures, par exemple, parallèle ou simultanée entre un son et une image (nota: il est possible de les supporter "indirectement" en attachant un attribut spécifique). Une extension possible est donc de supporter d'autres structures de contenu.
- Une des difficultés de la messagerie électronique est le manque de standards, ou l'abondance de candidats à la standardisation. Par exemple, chaque système d'imagerie a sa propre représentation d'images (format "mac paint", format "sun raster", format

"PostScript", etc). Il est possible pour MAP de "comprendre" la plupart des standards du marché et de les convertir automatiquement dans un format pivot universel. Cela implique aussi la nécessité d'avoir de nouveaux types de parties de contenu (par exemple, animation).

- La grande taille d'un message multi-média exige d'effectuer une compression avant de le stocker. Par exemple, une image 1000x1000(x24 bit couleur) occupe 3 méga octets, on arrive souvent à la réduire par un facteur de 4-5 par l'application de techniques de compression ([Campell 86]). Ce facteur peut être même plus important pour les sons. La version actuelle de MAP suppose que les FE d'applications effectuent la compression appropriée.
- La quantité de stockage nécessaire aux messages multi-média rend l'utilisation des nouvelles technologies de stockage comme CD-ROM intéressante ([Apple 88]). Cela indique une autre direction pour le travail futur.

La recherche "plein texte" est parfois très utile. Elle permet de chercher des messages d'après le "contenu", par exemple, les mots ou phrases qui apparaissent dans le corps (textes ou sons) de messages, ou un élément particulier d'une image (un portrait du président). Elle nécessite cependant des recherches dans les domaines du type "graphisme", "imagerie", "son", etc. et le domaine de la recherche d'information dans les messages multi-média ([Paget 89]). Cela constitue une extension possible à long terme. Le travail immédiat peut être l'ajout de la fonctionnalité consistant à chercher un mot dans des contenus "textes" d'items en utilisant la technique de "signature" ([Faloutsos]).

La réalisation du prototype que nous avons construit est plutôt simple et directe. Une réalisation plus sophistiquée doit tirer parti de nouvelles possibilités du matériel et du logiciel.

- CD-ROM est un nouveau média de stockage, il possède plus de capacité (au même prix) que les disques "durs" classiques. Cependant leur temps d'accès est plus lent. On peut donc envisager une stratégie d'utilisation comme suit:

  - on stocke les "gros" contenus statiques dans CD-ROM (images, sons ... des messages), et les attributs dans les disques durs. Cela garantit le temps de réponse pour l'opération de recherche et le temps d'accès aux petits messages, tout en utilisant la capacité de CD-ROM.

  - on stocke les messages peu consultés, par exemple, les "anciens" messages, en CD-ROM. Cela nous permet aussi d'utiliser la capacité de CD-ROM et en même temps de ne pas dégrader le temps de réponse pour la plupart des accès.

  - les informations d'historique sont statiques et ont souvent des volumes importants, le temps de réponse pour leur consultation est moins critique. On peut donc envisager de les stocker en CD-ROM.

- Les machines "base de données" (par exemple, le Serveur 32 de COPERNIC) fournissent la possibilité d'avoir une réalisation efficace et simple. Elles gèrent en principe directement le stockage de données et permettent la recherche "analogique", i.e. par contenu. Un des travaux futurs consiste donc à étudier l'utilisation des machines "base de données" pour la réalisation de MAP.

- La nature "orienté-objet" du modèle d'information de MAP implique la possibilité de l'utilisation des bases de données orientées-objet. Ces dernières sont en pleine expansion et sont en train d'entrer dans la phase de maturité.

Sans doute, les futurs travaux les plus importants consistent à définir et à construire divers composants ou modules ré-utilisables au niveau FE, réalisant chacun une fonction spécifique pour une classe d'applications. Ces modules peuvent être ensuite utilisés pour construire des FE de diverses applications. Par exemple:

- On peut envisager de construire un ensemble de modules de "distribution", gérant chacun une stratégie de distribution différente: duplication complète, duplication partielle avec un site de référence, "distribution hiérarchique", etc.

- Il est possible de concevoir plusieurs modules pour réaliser différentes interfaces d'accès, par exemple, en mode message (protocole P2+) ou en mode interactif (en utilisant ROSE: Remote Operation Service Element).

Pour ce qui me concerne, cette thèse a été enrichissante à plusieurs titres. Elle m'a permis de me spécialiser dans un domaine de l'informatique en pleine expansion et de connaître les divers standards dans les domaines réseaux et télématiques. J'ai acquis une idée précise sur la façon dont devait être mené un projet informatique à la fois réaliste et ambitieux, dans un temps et avec des moyens limités. J'ai pu maîtriser l'utilisation des outils formels de description de données ASN.1 via les spécifications des protocoles. De plus, la réalisation de cette thèse m'a permis de me familiariser avec le système Unix et ses outils de développement. Mais le principal intérêt de cette thèse réside dans la formation à la recherche que j'ai reçue. Je pense que je retiendrai un grand profit de ce travail. Enfin, je voudrais insister sur l'environnement particulièrement favorable que j'ai trouvé à l'Ecole des Mines de Saint-Etienne.

BIBLIOGRAPHIE

[AMIGO+ 88] AMIGO MHS+ Group, *The AMIGO MHS+ Report.* Edited by Nottingham University. March, 1988.

[ASN.1 88] *Specification of Abstract Syntax Natation One.* ISO 8824, ISO 8825, CCITT X.208, X209, 1988 version.

[Adiba 84] Adiba M, Azrou F., *An authorization mechanism for a generalized DBMS,* 3rd. seminar on distributed data sharing systemes, param, Italie, mars 1984.

[Akscyn 88] Robert M. Akscyn, Donald L. McCracken, and Elise A. Yoder, *KMS: A Distributed Hypermedia System for Managing Knowledge in Organisations.* Communication of the ACM, July 1988.

[Allman 86] Eric Allman, *SENDMAIL -- An Internetwork Mail Router.* In UNIX System Manager's Manual. University of California, Berkeley, 1986.

[Amouzegh 86] C. Ben Amouzegh, C. Chrisment, G. Zurfluh *Base d'Informations Généralisées -Concept de Version-,* Deuxième Journées de Bases de Données Avancées, INRIA France, 22-25 Avril 1986.

[Apple 88] Apple Computer, *AppleCD SC Developer's Guide.* Final Draft, Apple Developer Products Publications, Apple Computer, 7/05/88.

[Ashford 85] John H. Ashford *Text as Information: A review of current interactive storage and retrieval systems,* in [Miller 85].

[Azrou 84] Azrou F. *Un systeme d'autorisation pour une base de donnees generalisees,* These de 3eme cycle, INPG, Juin 1984.

[Bancilhon 88a] François Bancilhon, *Object-Oriented Database Systems,* Rapport Technique Altair 16-88, 19 Janvier 1988.

[Bancilhon 88b] F. Bancilhon, G. Barbedette, V. Benzaken, C. Delobel, S. Gamerman, C. Lécluse, P. Pfeffer, P. Richard, and F. Velez, *The Design and Implementation of O2 : an Object-Oriented Database System,* Rapport Technique Altair 20-88, 13 Avril 1988.

[Bannerjee 87] J. Bannerjee et al., *Data Model Issues for Object-Oriented Applications.* in ACM Transaction on Office Informations Systems 5(1) Jan. '87.

[Barbedette 86] Gilles Barbedette, Philippe Richard, *VOOD: Un Modèle de Données Orienté-Objet,* Rapport de Recherche No 580, INRIA, Novenbre 1986.

[Barbic 85] Barbic F., and Rabitti F., *The Type Concept in Office Document Retrieval,*

Proc. of 11th Int. Conf. on VLDB, 1985, Stockholm.

[Benford 88] S. Benford, *A Data Model for the AMIGO Communications Environment.* in EUTECO 88, April 1988. Vienna, North-Holland Publishers

[Berntein 81] Philip A. Bernstein, Nathan Goodman, *Concurrency Control in Distributed Database Systems,* ACM Computing Surveys, Vol 13, No 2, June 1982.

[Bogen 88] M. Bogen and K.-H. Weiss, *Group Co-ordination in a Distributed Environment.* in EUTECO 88, April 1988. Vienna, North-Holland Publishers

[Bono 85] Bono P. R. *A Survey of Graphics Standard and Their Role in Information Interchange,* IEEE COMPUTER, Vol 18, No 10, Special issue in "Multimedia Communications".

[Brown 84] Mark R. Brown, Karen Kolling, and Edward A.Taft *The Alpine File System,* Research Report of Palo Alto Research Center, CSL-84-4, October 1984.

[Brun 87] Philippe Brun, *Conference Répartie En Mode Messagerie.* Thèse de Doctorat, Elcole Nationale Supérieure des Mines de St-Etinnne. Novembre 1987.

[Bui 85] Bui Quang Ngoc, *Gestion des historiaues pour la base de donnees generalisées TIGRE,* Rapport de Recherche TIGRE No 29

[CSCW 86] *CSCW'86 -- Conference on Computer-Supported Cooperative Work.* Avril, 1989, Austin, USA.

[Campbell 88] Brad Campbell and Joseph M. Goodman, *HAM: A general Purpose Hypertext Abstract Machine.* Communication of the ACM, July 1988.

[Campell 86] G. Campell, T.A. DeFanti, J. Frederiksen, S.A. Joyce, L.A. Leske, J.A. Lindberg and D.J. Sandin, *Two Bits/Pixel Full Color Encoding.* SIGGRAPH 86, Vol 20, No 4, 1986. Dallas USA.

[Carracedo 86] J. Carracedo, M.A. Nunez, and E. Pastor, *Sets, A Way to Store Documents.* E_MAIL, MID <97*miquel@euittm.iris>, Oct. 1987.
AMIGO working document

[Chemin 85] Chemin L., Ferreira P., et Segond L., *SERVICES MULTIMEDIA SUR UN RESEAU LOCAL:L'ARCHIVAGE ET LA MESSAGERIE ELECTRONIQUES.* Rapport MASI No 100, Decembre 1985.

[Chen 76] Chen P.P-S, *The entity relationship Model - toward a unified view of data,* ACM TODS, Vol.1, No1, 1976.

[Chess 87] D.M. Chess and M.F. Cowlishaw, *A large-scale computer conferencing system,* IBM System Journal, Vol 26, No 1, 1987.

[Cocitra 89] Rodica Cocitra, Mohamed Daya, et Antoine Pause, *Archiveur pour Ajent Usager sur un Frontal.* Rapport du projet long logiciel, EMSE, Mai, 1989.

[Codd 70] Codd E.F., *A relational Model of Data for Large Shared Data Banks,* CACM, Vol 13, No.6, 1970.

[Conklin 87] Jeff Conklin, *Hypertext: An Introduction and Survey.* IEEE Computer, Sept.

1987.

[Coulouris 89] George F. Coulouris and Jean Dollimore, **DISTRIBUTED SYSTEMs - Concept and Design.** ADDISON WESLEY, 1989.

[Crocker 82] D.H. Crocker, *Standard for the Format of Arpa Internet Text Messages,* RFC 822. Network Information Center, SRI International, Menlo Park, California. August 1982.

[Danielsen 86] Thore Danielsen, Uta Pankoke-Babatz, Wolfgang Prinz, Ahmed Patel, Paul-Andre Pays, Knut Smalland, and Rolf Speth, *The AMIGO Project: Advanced Group Communicqtion Model for Computer-Based Communications Environment,* in [CSCW 86].

[Danielsen 87] Thore Danielsen, Trine Folkow, et Per Wiggo Richardsen, *Relation and Inheritance in Group Communication,* in MESSAGE HANDLING SYSTEMS, Proceedings of IFIP 6.5, Munich, 1987.

[Date 85] C.J. Date, **A Guide to DB2,** Addison-Wesley Publishing Company, 1985.

[Davis 83] Davis R. and Smith R.G., *Negotiation as a Metaphor for Distributed Problem Solving,* Artificial Intelligence 20, 1983, pp.63-109.

[Delgado 86] Jaime Delgado, Manuel Medina, Berthold Butscher, and Michael Tschichholz, *Storage Agents in Message Handling Systems.* 1986.

[Delgado 88] Jaime Delgado and Manuel Medina, *Use of the Abstract Service Definition Conventions for Distributed Information Processing Systems Specification.* in the proceedings of the IFIP TC 6/WG 6.5 working Conference on Message Handling Systems and Distributed Applications. Oct 1988, Costa Mesa CA, USA.

[DeSousa 81] DeSousa M.R., *Electronic information interchange in an office enviro-nemnt,* IBM Systems Journal 20, No.1, 4-22 (1981).

[DeWitt 84] DeWitt D., *Implementation Techniques for Main Memory Database Systems,* Proc.ACM SIGMOD, Conf. Management of Data, ACM, NewYork, 1984, pp.1-8.

[Dollimore 88] J. Dollimore, *The design of an object server as a storage module for the COSMOS messaging system.* in EUTECO '88, Vienna, April '88, North-Holland Publishers

[Donahue 85] James Donahue, and Willie-Sue Orr, *Walnut: Storing Electronic Mail in a Database,* Xerox PARC, CSL-85-9, Novembre 1985.

[Don 83] Donzeau-Gouge, V. *Outline of a tool for document manipulation,* IFIP 83, Paris, R.E.A. Maison ed., Elsevier Science Publishers B.V., 1983, 615 - 620.

[Economopoulos 83] Economopoulos P. *A system for image managing data.* Proc of the 9th IFIF Congress, pp 89-94, 1983

[Ehrlich 87] Susan F. Ehrlich *Strategies for Encouraging Successful Adoption of Office Communicatioon System.* ACM Trans. on Office Information System, Vol 5, No 4, Oct. 1987.

[Ellis 80] Ellis, C. and Nutt, G. *Computer science and office automation.* Computer surveys, 12, 1, (Marche 1980) 27-60.

[Ellis 84] Ellis C.A., (edited) Second ACM-SIGOA Conf. on OFFICE INFORMATION SYSTEMS, June 25-27, 1984, Toronto, Canada, vol.5 No 1-2.

[Engelbart 84] Engelbart, D.C., *Authorship Provisions in Augment,* IEEE 1984 COMPCOM Proceedings, Spring 1984, 465-472.

[Faloutsos 85] Christos Faloutsos, *Access Methods for Text.* Computer Surveys, Vol 17, No. 1, March 1985.

[Feldman 86] Martha S. Feldman, *Constaints on Communication and Electronic Mail,* in [CSCW 86].

[Ferreira 85] Ferreira P., Segond L., Fdida S., et Pujolle G., *Un Projet bureutique multimedia et le service d'Archivage electronique multimedia,* Rapport MASI No 108, decembre, 1985.

[Fikes 81] Fikes, R.E *Odyssey: A Kowledge based Personal Assistant.* Artifial intelligenc, 16 (3), pp. 331-361, July 1981.

[Foss 88] Carolyn L. Foss, *Effective Browsing in Hypertext Systems.* in RIAO 88, March 1988, Cambridge MA. U.S.A.

[Foster 86] Gregg Foster and Mark Stefik, *Cognoter, Theory and Practice of a Colab-orative Tool.* In [CSCW 86].

[Fujitani 84] L. Fujitani, *Laser Optical Disk: The Coming Revolution in On-line Storage.* Communication of the ACM, 27(6), pp. 546-554, June 1984.

[Goldberg 83] Adele Goldberg and David Robson, *Smalltalk-80: The Languageand its Implementation.* Addison-Wesley Publishing Company, 1983.

[Gray 81] Jim Gray, Paul McJones, Mike Blasgen, Bruce Lindsay, Raymond Lorie, Tom Price, Franco Putzolu, and Irving Traiger, *The Recovery Manager of the System R Database Manager.* ACM Computing Surveys, Vol 13, No 2, June 1981.

[Grimm 89] R.Grimm, and D.Heagerty, *Reccommendation for a short hand X.400 Adrress Notation.* RARE WG1 Recommendation, WG1-MHS-89.02.14, Feb. 1989.

[Hiltz 85] Starr Roxanne Hiltz and Murray Turoff, *Structuring Computer-Mediated Communication System to Avoid Information Overload.* Communication of the ACM, Vol. 28, No. 7, July 1985.

[Hoppenstand 88] Gregory S. Hoppenstand and David K. Hsiao, *A Technique to Improve the Precision of Full-Text Database Search.* In RIAO-88, March 1988, MA, USA.

[Horak 83] Horak W., *Interchaging Mixted Text Image Documents in the Office Environmement.* Computer & Graphics Vol 7, No 1, pp 13-29, 1983

[Horak 84] Horak W., Kronert B. *An Object-Oriented Office Document Architecture*

Model for Processing and interchange of documents. In 2th ACM-SIGOA Conf. on Office Information Systems, June, 1984, Toronto.

[Huitema 88] Christian Huitema, *The X.500 Directory Services.* Computer Networks and ISDN Systems 16, 1988, pp.161-166.

[Huitema 89] Christian Huitema, *Le Langage ASN-1.* Cours et Séminaires en Graphique en Environment Reparti, INRIA, Mai 1989, ROCQUENCOURT, France.

[Huitema 89] Christian Huitema, *The Challenge of Multimedia Mail.* Computer Networks and ISDN Systems, 17 (1989) 324-327. North-Holland.

[Kaye 87] A.Roger Kaye and Gerald M. Karam *Cooperation Knowlodge-Based Assistance for the Office* ACM Trans. on Office Information System. Vol 5, No 4, 1987.

[Kim 88] Won Kim, Nat Ballou, Hong-Tai Chou, Jorge F. Garza, Darrell Woelk, and Jay Banerjee, *Integration an Object-Oriented Programming System with an Database System.* OOPSLA '88 Proceedings, Setp 1988.

[Koller 81] Walter H. Kohler *A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems.* ACM Computing Surveys, Vol 13, No 2, June 1981.

[Kurose 84] James F. Kurose, Mischa Schwartz, and Yechiam Yemini, *Multiple-Access Protocols and Time-Constrained Communication.* ACM Computing Surveys, Vol 16, No. 1, March 1984.

[Laborie 87] Bernard Laborie, *Une Messagerie aux Normes X.400 sous Unix: Implentation, Interconnexion et Administration.* Thèse de Doctorat, Université Pierre et Marie CURIE, Setp. 1987.

[Lai 88] Kum-Yew Lai, Thomas W. Malone, and Ken-Chiang Yu, *Object Lens: A "Spreadsheet" for Cooperative Work.* in ACM Transactions on Office Information Systems, Vol 6, No 4, October 1988.

[Lanceros 88] A.G. Lanceros and J.A. Saras, *Group Communication Support in the MHS Environment.* in EUTECO 88, April 1988. Vienna, North-Holland Publishers

[Legatheaux 88] José Legatheaux Martins et Yolande Berbers, *La Désignation dans les systèmes d'exploitation répartis.* T.S.I, vol. 7, No. 4, 1988.

[Liwin 87] Witold Liwin, and David B. Lomet, *A New Method for Fast Data Searches with Keys.* IEEE Software, March 1987.

[Lockovsky 86] Lockovsky F.H. (edited), *Office and Data Base Systems Research'86.* Technical Report CSRI-183, July 1986.

[Lum 83] Lum V., Schek H. *Complex data object: text, voice, images, can DBMS namage them?* Proc. 9th. Conf. on VLDB, Florence, Nov. 1983

[Mackay 88] Wendy E. Mackay, *Diversity in the use of Electronic Mail: A Preliminary Inquery.* in ACM Transactions on Office Information Systems, Vol 6, No 4, October 1988.

[Maier 86] David Maier, Jacob Stein, Allen Otis and Alan Purdy, *Development of an Object-Oriented DBMS.* in OOPSLA'86 Proceedings. September 1986.

[Malone 87] Thomas W. Malone, Kenneth R. Grant, Franklyn A. Turbak, Stephen A. Brobst, and Michael D. Cohen, *Intelligent Information-Sharing Systems.* Communication of the ACM, Vol 30, No 5, May 1987.

[Marchionini 88] Gary Marchionini and Ben Shneiderman, *Finding Facts vs. Browsing Knowledge in Hypertext Systems.* Computer, January 1988.

[Mayer 85] Alastair J.W. Mayer, *Storage Architectures - Their Implications for Conferencing Systems.* BYTE, December 1985.

[Merrow 87] Thomas Merrow ans Jane Laursen, *A Progmatic System for Shared Persistent Objects.* in OOPSLA'87 Proceedings. October 1987.

[Metcalfe 76] Robert M.Metcalfe and David R.Boggs *Ethernet: Distributed Packet Switching for Local Computer Network.* Communicationde ACM, Vol 19, No 7, July 1976.

[Meyer 88] Bertrand Meyer, *Object-Oriented Software Construction.* Prentice Hall International.

[Morris 86] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. H. Rosenthal and F. Donelson Smith. *Andrew: A Distributed Personal Computing Environemnt.* Communication of the ACM 29(3):184-201, March 1986.

[Nunez 88] M.A. Nunez, J. Carracedo, and G. Gonzalez, *A Pilot Implementation of a Multiuser Message Store.* in EUTECO 88, April 1988. Vienna, North-Holland Publishers

[Paget 89] G. Paget, *Base dïmages et Réseaux de Neurones.* Troisième université d'été "Réseaux Connexionnistes en Informatique: Méthodes et Applications", 3-7 juillet 1989, Lyon.

[Palme 85a] Jacob Palme, *Conferencing Standards.* December 1985, Byte pp 187-194.

[Palme 85b] Jacob Palme, *Data Structure in PortaCOM.* December 1985, Byte pp 195-202.

[Palme 87] J. Palme, *AMIGO: Document Storage and Retrieval.* In Proceedings of the IFIP TC 6/WG 6.5 Working Conference on Message Handling Systems, Munich, F.R.G., 27-29 April, 1987.

[Palme 88] J. Palme, *Two techniques for a distributed message database.* in EUTECO 88, April 1988, Vienna, North-Holland Publishers

[Pankoke 89] Uta Pankoke-Babatz, Thore Danielsen, Ahmad Patel, Paul-André Pays, Wolfgang Prinz, and Rolf Speth, **Computer Based Group Communication - the AMIGO activity model** edited by Uta Pankoke-Babatz, ELLIS HORWOOD, 1989.

[Papageorgiou 88] Mario Papageorgiou, *Le système de gestion de fichiers répartis dans CHORUS*, T.S.I, Vol. 7, No. 4, 1988.

[Partridge 88] Craig Partrige and Marshall T. Rose, *A Comparison of External Data Formats*. in the proceedings of the IFIP TC 6/WG 6.5 working Conference on Message Handling Systems and Distributed Applications. Oct 1988, Costa Mesa CA, USA.

[Pays 87] P.A. Pays and Ph. Brun, *CRMM - A Distributed Bulletin Board*. in IFIP 6.5 International Working Conference on Message Handling Systems. Avril, 1987.

[Pays 88] P.A. Pays and R. Speth, *An Architecture Framework for Group Activities*. in EUTECO 88, April 1988. Vienna, Austria.

[Pays 89] Paul-André Pays, Alain Zahm, et Philippe Brun, *Les Problèmes de Sécurité dans Logiciels de Messagerie aux Normes X.400*. TRIBUNIX, Vol. 5, No 25, Mars/Avril 1989.

[Poggio 85] Poggio A., Garcia Luna Aceves J.J., Craighill E.J., Moran D., Aguilar L., Worthington D., and Hight J., *CCWS: A Computer-Based, Multimedia Information System*. In special issue on Multimedia Communication, IEEE Computer, Oct, 85, Vol18, No 10.

[Pollock 88] Stephen Pollock, *A Rule-Based Message Filtering System* in ACM Transactions on Office Information Systems, Vol.6, No.3, July 1988.

[Postel 82] J. B. Postel., Simple Mail Transfer Protocol. RFC 821. Network Information Center, SRI International, Menlo Park, California. August 1982.

[Pudry 87] A. Pudry, B. Schard and D. Maier *Integrating an object server in other worlds*. in ACM Trans on Office Informations Systems 5(1) Jan. 87.

[Pujolle 88] G. Pujolle et M. Schwartz *Réseaux Locaux Informatiques*. EYROLLES, 1988.

[Reynolds 85] Reynolds J.K., Postel J.B., Katz A.R., Finn G.G., and DeSchon A.L., *The DARPA Experimental Multimedia Mail System*. in Special issue on Multimedia Communication, IEEE Computer, Oct, 85, Vol18, No 10.

[Rollin 86] Florence Rollin, *Transfert de Fichiers en Mode Messagerie*. Thèse de Doctorat, Ecole des Mines de St-Etienne, Décembre 1986.

[Rosenberg 88] Jonathan Rosenberg, Craig Everhart and Nathaniel S. Borenstein, *An Overview of the Andrew Message System,*

[Sakata 85] Sakata S., and Ueda T., *A Distributed Interoffice Mail System*. In Special Issue on Multi-media Communications, IEEE Computer, Vol 18, No10, Oct, 1985.

[Saras 87] Juan A. Saras, Andres G. Lanceros, *A Model for a Multiuser Message Store,* Working Document, February 9, 1987.

[Schicker 88] Pietro Schicker, *Message Handling Systems, X.400,* in IFIP WG 6.5 Conference on "Message Handling Systems and Distributed Applications", Costa Mesa, CA, USA, Oct 1988.

[Schick 82] Schick T., and Brockish R.F., *The Document Interchange Architecture: A*

membre of a family of architectures in the SNA environment. IBM Systems Journal, Vol 21, No 2, 1982.

[Shneiderman 83] Shneiderman B., University of Maryland *Direct Manipulation: A Step Beyond Programming Languages.* IEEE Computer, August 1983.

[Smith 87] Peter D. Smith and G.Michael Barnes *Files & Databases, an introduction.* Addison-Wesley Publishing Company, 1987.

[Smith 88] H.T. Smith, *The Requirements for Group Communication Services.* in EUTECO 88, April 1988. Vienna, North-Holland Publishers

[Stallings 84] William Stallings, *Local Networks.* ACM Computing Surveys, Vol 16, No. 1, March 1984.

[Stefik 87] Stefik, M., G. Foster, D.G. Bobraw, K.M. Kahn, S. Lanning and L.A. Suchman *Beyond the Chelkboard: Using Computers to Support Collaboration and Problem Solving in Meetings.* CACM, 30 (1), Jan. 1987.

[Svobodova 84] Liba Svobodova, *File Servers for Network-Based Distributed Systems.* in Computing Serveys, Vol. 16, N0. 4, December 1984.

[Swinehart 79] Daniel Swinehart, Gene McDaniel, and David Boggs, *WFS: A Simple Shared File System for a Distributed Environment.* Research Roport of Palo Alto Research Center, CSL-79-13, Oct.1979.

[Tanenbaum 81] Andrew S. Tanenbaum, *Network Protocols,* Computing Surveys, Vol.13, No.4, December 1981.

[Terry 89] Douglas Brian Terry, *Distributed Name Servers: Naming and Caching in Large Distributed Computing Environments.* Research Report, Xerox Palo Alto Research Center, February 1985.

[Trigg 86] Trigg, R.H. and M. Weiser, *TEXTNET: A Nerwork-Based Approach to Text Handling.* ACM Transactions on Office Information Systems, 4 (1), Jan.1986.

[Trigg 86] R. Trigg, L. Suchman and F. Halasz, *Supporting Collaboration in NoteCards.* In [CSCW 86].

[Tsichritzis 82a] Tsichritzis D., Lochovsky, F., *Data Models.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[Tsichritzis 83] Tsichritzis D., Chistodoulakis S et .al. *A Multimedia Office Filing System.* Proc. of the 9th. Int. Conf. on VLDB, Florence, Nov. 1983

[Tsichrizis 82b] Tsichrizis, *Form management.* Communication of the ACM, Vol 25, No 7, pp 453-478, 1982.

[Vervest 87] Peter H.M. Vervest, *Innovation in Eletronic Mail.* 1987, North-Holland.

[Whang 87] Kyu-Young Whang and al. *Office-by-Example: An Integrated Office System and Database Manager.* ACM Trans. on Office Information System. Vol 5, NO 4, Oct 1987.

[X400 84] *CCITT's Series of x.400 Recommentations.* 1984 version.

[X400 88] *CCITT's Series of x.400 Recommentations.* 1988 version.

[X.500 88] *The Directory.* CCITT Draft Recommandations X.500, ISO OIS 9594, 1988.

[Yankelovich 85] N. Yankelovich, N.K. Meyrowitz, and A. van Dam, *Reading and Writing the Electronic Book.* Computer 18 (10), Oct.1985, 15-30.

[Yankelovich 88] N. Yankelovich, B.J. Haan, N.K. Meyrowitz, and S.M. Drucker, *Intermedia: The Concept and the Construction o a Seamless Information Environment.* IEEE Computer, January, 1988.

[You 87a] Y.Z. You, *Serveur d'Archivage de Document,* Rapport Technique, EMSE, No. 87-01, Juillet 1987.

[You 87b] Y.Z. You et P.A. Pays, *SAD: Serveur d'Archivage de Dcouments,* Rapport de recherche, EMSE No. 87-01, Juillet 1987.

[You 88] Y.Z. You and P.A. Pays, *SAM: A general Multi-User Message Store.* Rapport de Recherche, EMSE, No. 88-03, Juin, 1988.

[You 89a] Y.Z. You and P.A. Pays, *A general Multi-User Message Store.* In **Message Handling Systems and Distributed Applications,** Edited by E. Stefferud, O-J. Jacobsen, and P. Schicker. North-Holland 1989.

[You 89b] Y.Z. You and P.A. Pays, *POMS: A Message Store for an Office Environnement.* Rapport de Recherche, EMSE, No. 89-02, juin, 1989.

[You 89c] Y.Z. You and P.A. Pays, *A Message Store for an Office Environement,* International Symposium'89 ICCC, Sept 1989, Beijing China.

[You 89d] Y.Z. You, *Specification of POMS,* Document Technique. Mars, 1989.

[You 89e] Y.Z.You, *Un système de Vote en Mode Messagerie.* Document Techinique, Juillet, 1989.

[You 89f] Y.Z. You, *Définitions Formelles des Services Abstraits de MAP et de POMS en ASN.1.* Rapport de Recherche, EMSE, No. 89-10 Octobre, 1989.

[Zdonic 84] Zdonik S. *Object Management System Concepts.* In [Ellis 84].

[Zimmerman 80] Hubert Zimmerman, *OSI Reference Model - the ISO model of architecture for open systems interconnection,* IEEE Trans. Commun. COM-28 (Avril 1980) 425-432.

[Zloof 82] M.M. Zloof *Office-by-Exemple: A Business Language that Unifies Data and Word Processing and Electronic Mail,* IBM System Journal, 21(3), pp.272-304, 1982.

Abréviations

ACID : Acess Controler IDentifier.

ACL : Acess Control List.

AID : Activity IDentifier.

AMIGO : Advanced Messaging In GrOup.

ASN.1 : Abstract Syntax Notation One.

AU : Access Unit.

BDOO : Base de Données Orientées-Objet.

BDR : Base de Données Relationnelles.

CCITT : Comité Consultatif International Télégraphique et Téléphonique.

CP : Content Part.

CRMM : Conférence Répartie en mode Messagerie.

DCP : Document Content Part.

DIB : Directory Information Base.

DIT : Directory Information Tree.

DS : Directory System.

DSA : Directory Service Agent

DSUA : Directory Service User Agent.

EID : Event IDentifier.

FA : Filtre Analytical.

FC : Filtre Contenu.

FE : Front-End.

FL : Filtre Lien.

FU : Filtre Utilisateur.

GC : Group Cmmunication.

IPM : Inter-Personal Messaging.

IPMessage : Inter-Personal Message, ou Message IP.

IPMS : Inter-Personal Messaging System.

IPM-UA: Inter-Personal Messaging User Agent.

ISO : International Standard Organisation.

MAP : Message Archive Processor.

MH : Message Handler.

MHS : Message Handling System.

MMM : Messagerie Multi-Média.

MS : Message Store.

MTA : Message Transfert Agent.

MTS : Message Transfert System.

OID : Object IDentifier.

P1 : Protocole entre MTA défini par la version 1984 des recommandations X.400

P2 : Protocole entre les IPM-UA défini par la version 1984 des recommandations X.400

PDAU : Physical Dilevery Access Unit.

POMS : Personal Office Message Store.

Ppoms : Protocol du système POMS.

Psam : Protocol d'Accès entre MAP et les Front_Ends.

Pv : Protocole du système de vote.

RCP : Reference Content PArt.

RID : Role IDentifier.

SAD : Serveur d'Archivage de Documents.

SAM : Serveur d'Archivage de Messages.

UA : User Agent (of MTS).

UID : User IDentifier.

USD : User Set Descriptor.

VSA : Vote Service Agent.

VSUA : Vote Service user Agent.

X.400 : La série de recommandations de CCITT pour un système de messagerie.

X.413 : La recommandation de la série X.400 sur le stockage de message (MS).

X.500 : La recommandation de CCITT pour le système d'annuaire.

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 36 ID
Année 1989

S A M : Un Serveur d'Archivage de Messages

Yi Zhi YOU

Spécialité : Informatique, Image, Intelligence Artificielle et Algorithmique

Mots Clés : Archivage de Messages, Messages, Messagerie X400, Communication de groupe, Conférence, Vote.

Résumé : La messagerie électronique fournit un nouveau moyen de communication entre individus. Les applications de communication de groupe en mode messagerie permettent aux utilisateurs d'effectuer des travaux coopératifs. Dans ces applications, un outil général est nécessaire pour stocker et gérer les messages d'une façon distribuée dans un environnement multi-utilisateurs. Cette thèse présente la conception et la réalisation d'un tel outil : SAM (Serveur d'Archivage de Messages).

SAM peut être caractérisé par trois modèles et un protocole d'accès. Le modèle architectural décompose SAM en deux parties : un noyau (MAP) qui stocke et gère les messages et fournit des fonctions de base, et un ensemble de composants (FE) qui adaptent SAM à différentes applications. Le modèle conceptuel définit d'une façon abstraite des objets que SAM doit gérer du point de vue des applications. Le modèle de MAP spécifie les objets à stocker et gérer, leurs relations et leur représentation, ainsi que le service fourni par MAP pour manipuler ces objets. Le protocole d'accès Psam définit de façon formelle l'interface d'accès entre MAP et les FE, en utilisant le langage de description ASN.1 recommandé par le CCITT et l'ISO. Quelques exemples d'utilisation de SAM sont également présentés afin de valider notre conception de cet outil.