République Algérienne Démocratique et Populaire
Ministre de l'enseignement supérieur et de la recherche scientifique

UNIVERSITE MENTOURI DE CONSTANTINE
FACULTE DES SCIENCES DE L'INGENIEUR
DEPARTEMENT D'INFORMATIQUE

N° d'ordre :…..
Série :…..

MEMOIRE

Présenté pour l'obtention du diplôme de
Magister en Informatique
Option : Information and Computation

Coopération multi agents pour le traitement des requêtes sur des sources de données hétérogènes et distribuées

Présenté par :

Mr. Benharzallah Saber

Dirigé par :

Pr. Sahnoun Zaidi

Soutenu le: 05/03/2005

Devant le jury:

- Président : **Mr. Boufaida Mahmoud** (Professeur, Université de Constantine)
- Rapporteur : **Mr. Sahnoun Zaidi** (Professeur, Université de Constantine)
- Examinateurs :**Mme. Boufaida Zizette** (Professeur, Université de Constantine)
  Mr. Zarour Nacereddine (Maître de Conférence, Université de Constantine)

Dédicace

A mon père et ma mère

A mes frères Abdelkader et Tayeb et ma soeur

A tous les membres de la famille

A tous mes amis

Benharzallah Saber

Remerciements

En tout premier lieu, je remercie Dieu, tout puissant, qui m'a aidé à réaliser ce travail. Je remercie très sincèrement Monsieur Sahnoun Zaidi, Professeur à l'Université Mentouri de Constantine et Directeur du Laboratoire LIRE d'avoir accepté d'être rapporteur de ce mémoire. J'ai beaucoup profité de sa grande expérience et de la documentation. Ses remarques pertinentes et constructives sur mon travail m'ont aidé à en améliorer la qualité.

Je tiens à exprimer ma profonde gratitude aux membres de mon jury qui me font le grand honneur d'avoir accepté de juger mon travail.

Je remercie Monsieur Boufaida Mahmoud, Professeur à l'Université Mentouri de Constantine, Directeur de SIBC (équipe de Systèmes d'Information et Bases de Connaissances), pour l'honneur qu'il me fait, en acceptant la présidence de ce jury.

Je remercie Madame Boufaida Zizette, Professeur à l'Université Mentouri de Constantine et Monsieur Zarour Nacereddine, Maître de Conférence à l'université Mentouri de Constantine auxquels je suis très reconnaissant d'avoir accepté d'être examinateurs de ce travail.

Je voudrais remercier mes amis : Djamel et Kamel pour leur aide.

Je remercie tout particulièrement mes parents qui m'ont toujours soutenu et m'ont permis de mener à bien mes études.

Je remercie aussi les deux enseignants de l'université de Batna Mr A. Dkhinet et Mr M. Drid pour leur aide.

Enfin, je remercie tous ceux qui ont contribué directement ou indirectement à l'aboutissement de ce travail.

Table des matières

Introduction générale

Depuis plusieurs années, l'intérêt de faire collaborer des systèmes ne cesse de croître. Aujourd'hui différentes entreprises et organisations doivent faire face au challenge de faire interopérer des systèmes de bases de données afin de pouvoir réaliser des fonctions critiques[GENO 02]. Ces entreprises et organisations possèdent une grande latitude pour définir le format des données et le mode d'accès les plus adéquats de leurs données.

De ce fait l'utilisateur doit respecter la méthodologie d'accès propre à chaque source. Cela implique de connaître la localisation de la base, la description du contenu, les possibilités d'interrogation, le format des résultats. Muni de ces informations, il formule des requêtes afin de recevoir le résultat attendu.

L'objectif principal est donc d'offrir aux utilisateurs les moyens de manipuler d'une manière transparente des données issues d'un ensemble de sources de données hétérogènes, autonomes et distribuées. La résolution d'une requête dans un environnement interopérable passe par l'intégration de données distribuées sur plusieurs systèmes hétérogènes. Le problème d'intégration de données hétérogènes et distribuées est devient l'un des plus importants axes de recherche dans le domaine des systèmes d'information[TAME 99][GENO 02][BOUZ 03].

Ce problème ne consiste pas uniquement à homogénéiser les modes de représentation des données en utilisant un format commun tel que le eXtensible Markup Language (XML). Mais réellement plusieurs problèmes se posent lorsqu'on veut réaliser un système d'intégration de données hétérogènes et distribuées:

1- Le problème de distribution et d'hétérogénéité des sources de données;
2- Le traitement d'une requête posée indépendamment de la localisation des différentes données intervenant pour calculer le résultat. Cela introduit les difficultés:
   - la reformulation d'une requête en termes des requêtes destinées aux sources,
   - la recomposition des résultats, et
   - le niveau de l'optimisation.

3- La gestion des conflits de données (les conflits syntaxiques, schématiques et sémantiques). Différents modèles de données existent. Ils possèdent des concepts qui sont rarement équivalents et permettent une représentation plus ou moins riche des données. Néanmoins, même en disposant de systèmes basés sur un modèle de représentation identique, l'hétérogénéité réapparaît au niveau des choix conceptuels faits pour représenter les données (conflits schématiques). Les conflits sémantiques apparaissent à deux niveaux: au niveau conceptuel et au niveau de valeurs. Au niveau conceptuel il consiste à identifier les différences et les ressemblances cognitives des informations. Au niveau des valeurs il consiste à interpréter d'une manière correcte le contenu d'une donnée malgré les différences de représentation d'un système à l'autre.
4- Les solutions doivent assurer l'autonomie des systèmes, l'extensibilité de l'architecture pour maîtriser l'ajout et le retrait de sources de données, la scalabilité du système face à l'évolution du nombre de participants et la transparence d'accès à la localisation et au format des données.

Plusieurs solutions ont été proposées et chaque solution fournit quelques avantages au détriment d'autres. L'objectif de notre travail est de proposer une architecture pour l'intégration et l'interrogation des sources de données hétérogènes et distribuées. Cette architecture offre aux utilisateurs les moyens de manipuler de manière transparente des données issues d'un ensemble de sources de données hétérogènes, autonomes et distribuées. Elle combine entre l'approche de type médiation et l'utilisation du paradigme agent. Notre solution est caractérisée par:

1- Architecture multi agents décentralisée qui évite les points faibles des architectures centralisées. Offre une transparence d'accès à la localisation, aux schémas sources et aux langages de requêtes des données sources;
2- Nous avons défini des stratégies de coopération entre agents pour l'efficacité du traitement des requêtes et l'amélioration du rendement global du système et assure la scalabilité du système. Chaque agent décide son comportement pour satisfaire les besoins définis au niveau global;
3- Elle offre une intégration sémantique des données en utilisant les nouvelles technologies pour la représentation de la sémantique des informations;
4- Les données hétérogènes supportées sont les données relationnelles, objets et XML;

5- Utilise le langage d'interrogation expressive commun XQuery et le modèle XML Schéma comme modèle de données commun;
6- Approche d'intégration descendante et une adaptation des règles de mapping GLAV offrent une flexibilité aux changements;
7- Traitement des sous requêtes destinées aux sources de données permettant de supporter des systèmes anciens (legacy system);
8- Les agents communiquent en terme de transfert de connaissances plutôt qu'en terme de transfert de données.

Afin de présenter notre travail, nous proposons le plan de lecture suivant:

Le premier chapitre est consacré à la présentation des différentes approches d'intégration de données. Nous donnerons premièrement quelques concepts liés au domaine, ensuite les problèmes engendrés de l'hétérogénéité des données, la classification des SIs et les critères de classification des systèmes d'information fédérés qui sont considérés comme des systèmes d'intégration de données.

Le deuxième chapitre concerne la présentation des conflits des données et les modèles communs adoptés par les différentes solutions et nous présenterons avec plus de détail le modèle XML Schéma et le langage d'interrogation commun XQuery adoptés dans notre solution. Ce chapitre présente aussi les outils utilisés pour la représentation de la sémantique des informations.

Le troisième chapitre concerne d'étudier les avantages et les inconvénients des approches de traitement des requêtes dans les systèmes à base d'un médiateur. Et nous décrirons les problèmes de la reformulation, décomposition et d'optimisation des requêtes.

Le quatrième chapitre est consacré à la présentation d'un aperçu sur les systèmes multi agents, qui est en relation avec notre travail et d'une étude sur l'utilisation des SMA pour l'intégration d'informations.

Le cinquième chapitre est consacré à la présentation de notre solution. Ce chapitre est divisé en deux parties : dans la première partie nous donnerons un aperçu générale de la solution, nous décrirons les idées, les caractéristiques de la solution, l'architecture générale,

l'architecture interne d'un agent et la coopération entre agents. Et dans la deuxième partie nous donnerons l'architecture détaillée permettant d'expliquer comment les agents coopèrent pour traiter efficacement des requêtes et améliorer le rendement global du système.

Le mémoire se termine par une conclusion générale et propose des visions pour les travaux futurs.

Chapitre 1 : L’intégration de données hétérogènes et distribuées

1.1 Introduction

L'évolution constante en matière de réseaux et de bases de données ces trente dernières années poussent à la réalisation de systèmes d’intégration de données de grande envergure. Ces systèmes fournissent aux utilisateurs une vue uniforme sur un ensemble de sources de données.

Dans ce chapitre nous allons présenter les dimensions de classification des SIs, les problèmes engendrés de l’hétérogénéité des données, la classification des SIs et les critères de classification des systèmes d’information fédérés qui sont considérés comme des systèmes d’intégration de données. Ensuite nous comparerons les trois approches permettent de réaliser l'intégration des sources de données.

1.2 Dimension de classification des SIs

Un système d’information (SI) fournit l’accès aux informations à travers un gestionnaire de données. En effet, les SIs se caractérisent selon trois dimensions: autonomie, hétérogénéité et distribution.

1.2.1 Autonomie

Lorsqu’un SI est composé de multiples composants (multiples systèmes locaux), l’autonomie des composants intégrés devient un problème critique. Normalement il accepte de partager les données et pas le contrôle. Il y a différents niveaux d’autonomie[BUSS 99][PUCH 99][GENO 02]:

- **Autonomie de conception :** signifie que chaque composant est différent des autres dans leur conception. Chacun possède son propres modèle de données, langage d’interrogation, interprétation sémantique des données, contraintes, fonctions, etc...

- **Autonomie de communication :** les composants d’un SI décident quand et comment répondent aux requêtes.
- **Autonomie d’exécution :** les composants d’un SI n’ont pas besoin d’informer d’autres systèmes locaux de l’ordre d’exécution des transactions locales ou des opérations externes. Un système local ne distingue pas les opérations locales et globales.

1.2.2 Hétérogénéité et distribution

L'hétérogénéité est indépendante de la distribution physique des données à travers un réseau. Les systèmes d'information ou les bases de données peuvent être réparties en plusieurs sources séparées géographiquement, et cependant rester homogènes.

Selon [NAAK 99] un système d'information est qualifié d'homogène si le logiciel qui crée et manipule les données est identique pour toutes les sources, et si toutes les données ont le même format (ou modèle). Par opposition, un système est dit hétérogène s'il ne satisfait pas toutes les exigences d'un système homogène.

C’est à dire, toute dissimilitude du système, aussi bien au niveau de la conception que de l'implémentation, suffit pour le rendre hétérogène. Par exemple, les sources peuvent avoir des modèles de données différents, des langages différents pour créer les données et pour les manipuler.

Plus les dissimilitudes sont grandes, et plus il est difficile de masquer cette hétérogénéité aux utilisateurs.

1.3 Problèmes d’hétérogénéité de données

L'intégration des données hétérogènes est confrontée à de nombreux conflits de données qui sont classer généralement dans trois classes : **Les conflits syntaxiques,** proviennent de l'utilisation de modèles de données différents d'un système à l'autre. **Les conflits schématiques,** résultent d'une structuration et d'une classification différente des informations. Ils sont liés aux choix de conception. **Les conflits sémantiques,** proviennent des différences d'interprétation des informations entre différentes sources de données. Nous décrirons ces conflits avec plus de détail dans le chapitre 2.

1.4 Classification des SIs

En se basant sur les deux dimensions de distribution et d'hétérogénéité, [BUSS 99] a défini trois classes des systèmes d'information.

- *SI centralisé* : est exécuté sur un ordinateur. Il offre un ou plusieurs interfaces aux utilisateurs.
- *SI distribué* : les données sont physiquement distribuées sur multiples sites qui sont connectés sur un réseau de communication.
- *SI hétérogène :* est une collection des systèmes d'information qui n'assurent pas au moins un critère de l'homogénéité.

En ajoutant la dimension d'autonomie, on obtient une quatrième classe, c'est les SI fédérés (SIF). Un SIF est défini selon [BUSS 99], comme un ensemble de systèmes d'information autonomes et participant à une fédération. Les applications et les utilisateurs accèdent à un ensemble de sources de données hétérogènes à travers une couche de fédération qui offre un chemin d'accès uniforme. L'uniformité est réalisée par des stratégies spécifiques d'interopérabilité.

Dans ce qui suit, nous donnons les critères de classification des systèmes d'information fédérés.

1.4.1 Critères de classification des Systèmes d'information fédérés

Il existe dans la littérature plusieurs classifications des systèmes d'information fédérés. Quelques classifications suivent la dimension de la manière d'accès aux données, d'autres suivent la dimension de degré d'autonomie et d'hétérogénéité des données[GENO 02][BENA 02]. Nous présentons des critères de classification des SIF. Ces critères sont complets et prennent en compte toutes les classes des systèmes d'information fédérés. Ces critères sont inspirés de celle présentés dans [BUSS 99][LESE 00][WOHR 04].

a- Types des données supportées

Signifie les types de données que le système peut intégrer: structuré (exemple: données relationnelles), semi-structuré (exemple: données XML) et /ou non-structuré (exemple: fichier).

b- Fortement versus faiblement fédéré

Il existe deux types des SIF: fédération forte et fédération faible. Les systèmes fortement fédérés ont un schéma global unifié accéder par les utilisateurs. Les systèmes faiblement fédérés n’ont pas un schéma global mais offrent un langage commun pour l’accès aux données[BUSS 99].

Les systèmes fortement fédérés offrent un schéma, langage et une interface transparente, alors que les systèmes faiblement fédérés n’offrent qu’un langage commun. Dans les systèmes fortement fédérés, le processus d’intégration des schémas peut suive une méthode semi-automatique ou ad-hoc[BUSS 99]. Ils peuvent supporter plus l’intégration sémantique, au contraire des systèmes faiblement fédérés.

Le critère important pour distinguer l’intégration faible de l’intégration forte est les composants des schémas présentent aux utilisateurs. Dans l’intégration forte les schémas sources ne sont pas visibles; par contre dans l’intégration faible les sources sont visibles, bien qu’il est possible d’utiliser des vues additionnelles[WOHR 04].

c- Modèle de données commun de SIF

Un SIF est caractérisé par le modèle de données commun utilisé. Ce modèle doit être le plus puissant possible afin de permettre, tant que faire se peut, l'expression des concepts de l'ensemble des modèles de données sources.

Les modèles les plus connus sont le modèle de données relationnel, objet et semi-structuré. [PREE01] n’utilise pas un modèle commun, chaque utilisateur accède aux sources hétérogènes selon son modèle de données local. L’inconvénient de cette approche est que les différents modèles n’ont pas les mêmes puissances d’exprimer les différents concepts.

d- La transparence

La transparence est le principal objectif[BUSS 99]. Elle donne l’illusion aux utilisateurs qu’ils interagissent avec un système local, central et homogène. Ils existent différents types de transparence : la transparence à la localisation, aux schémas sources et aux langages de requêtes des données sources.

e- La formulation d'une requête

Les systèmes d'information sont classifiés par le type de requête : requête structurée ou requête de recherche d'information (information retrieval). Par exemple les SGBDs appartient au type de requête structurée, alors que les moteurs de recherche à l'autre type.

f- Stratégies de développement

Pour développer un système fortement fédéré ce là fait apparaître également le problème de trouver un schéma global approprié.

Il existe dans la littérature deux approches de conception du schéma global[LESE 00] : Approche descendante (Top down) ou ascendante (Bottom up).

Dans l'approche ascendante, les schémas de sources de données sont considérés comme des vues qui doivent être intégrées dans un schéma global uniforme et homogène. Et dans l'approche descendante, la conception du schéma global fait exactement de la même manière comme un schéma centralisé d'une base de données. A partir des conditions globales ont comme résultat des vues globales, et finissons avec un schéma homogène. Les schémas sources ne sont pas considérés dans ce processus, mais doivent être liés au schéma global ultérieurement (la correspondance entre le schéma global et des schémas sources).

Le concepteur spécifie dans l'approche descendante des règles de correspondance entre le schéma global et schémas sources. Cette approche offre une flexibilité plus élevée par rapport à l'approche ascendante, elle est plus adaptée aux changements des sources de données.

Le concepteur définit dans l'approche ascendante des correspondances entre les différents schémas sources qui sont utilisés pour décrire le schéma global. Dans cette approche l'intégration des schémas sources est réalisée généralement en quatre étapes [PARE 96] : la première étape est la pré-intégration, tous les schémas sont transformés dans un modèle de données commun. La deuxième étape est la comparaison des schémas pour identifier les correspondances et les conflits. Et la troisième étape pour résoudre les conflits trouvés (conformer les schémas). La dernière étape est de fusionner tous les schémas en un schéma intégré. Il faut que ce schéma intégré doit être complet, correct, minimal et compréhensible[BOUZ 03]. Dans cette approche l'intégration des schémas est de nature statique, si n'importe quelle source change son schéma, ou on ajoute une nouvelle source, il

faut répéter le processus d’intégration. Au contraire l’approche descendante ne nécessite que de modifier les règles de correspondance entre schémas sources et le schéma global.

g- Intégration virtuelle ou matérielle

L’intégration matérielle est l'approche entrepôts de données (data warehousing) consiste à voir l’intégration comme la construction de bases de données réelles, appelées entrepôts, regroupant les informations pertinentes pour les applications considérées. L'utilisateur pose alors ses requêtes ou lance un traitement directement sur les données stockées dans l'entrepôt.

L'intégration virtuelle (c’est notre contexte) consiste à fonder l'intégration d'informations sur l'exploitation de vues abstraites (ou virtuelles, non matérialisées) décrivant le contenu des différentes sources de données. Les données sont accessibles qu'au niveau des sources de données. L'interrogation et la détermination des sources d'information pertinentes nécessitent la construction de plans de requêtes dont l'exécution permettra d'obtenir l'ensemble des réponses à partir des sources disponibles.

h- Accès de lecture seul ou lecture / écriture

On peut distinguer entre les SIF qui permettent l’insertion / mise à jours à travers la couche de fédération et les systèmes qu’ils ne permettent pas (seulement requête d’interrogation).

Dans les projets actuels et la plupart des prototypes de systèmes d´écrits dans la littérature [GENO 02] sont des systèmes orientés vers l’interrogation. La raison principale est due au fait que les mises à jour dans un environnement distribué ont un impact sur le contrôle de la concurrence, la journalisation et la sécurité[GENO 02].

Le contrôle de la concurrence globale c’est l’aspect clé pour pouvoir exécuter des mises à jour globales et de coordonner les opérations exécutées concurremment pour éviter des interférences inacceptables.

La complexité du problème du contrôle de la concurrence dans des environnements distribués vient de l’hétérogénéité et de l’autonomie des bases de données[GENO 02].

Selon [PUCH 99] pour maintenir la cohérence globale, le mécanisme de gestion de la concurrence globale doit garantir : (1) l'atomicité de chaque transaction globale, c'est à dire que soit tous ses effets sont pris en compte sur l'ensemble des sources locales soit tous ses effets sont annulés et (2) la sérialisabilité globale des exécutions, c'est à dire qu'une exécution concurrente de transactions globales donne un résultat équivalent à une quelconque exécution en série de ces mêmes transactions globales. La condition (1) est difficile du fait de l'autonomie d'exécution qui autorise une source locale à abandonner unilatéralement une branche de transaction globale. La condition (2) est quant à elle rendue difficile, du fait de l'hétérogénéité structurelle qui autorise différents sources à sérialiser leurs transactions en utilisant des protocoles différents (verrouillage à deux phases, estampillage,…), du fait de l'autonomie de communication qui interdit toute observation des protocoles locaux et enfin du fait de la présence de transactions locales qui échappent un contrôle global.

1.5 Approches d'intégration de données hétérogènes et distribuées

De nombreuses solutions ont été proposées pour résoudre le problème de l'intégration de données (intégration virtuelle)[BENS 99][BUSS 99][JOUA 01]. On distingue trois approches (figure 1.1). Les projets réalisés selon ces approches sont considérés comme des systèmes d'information fédérés.

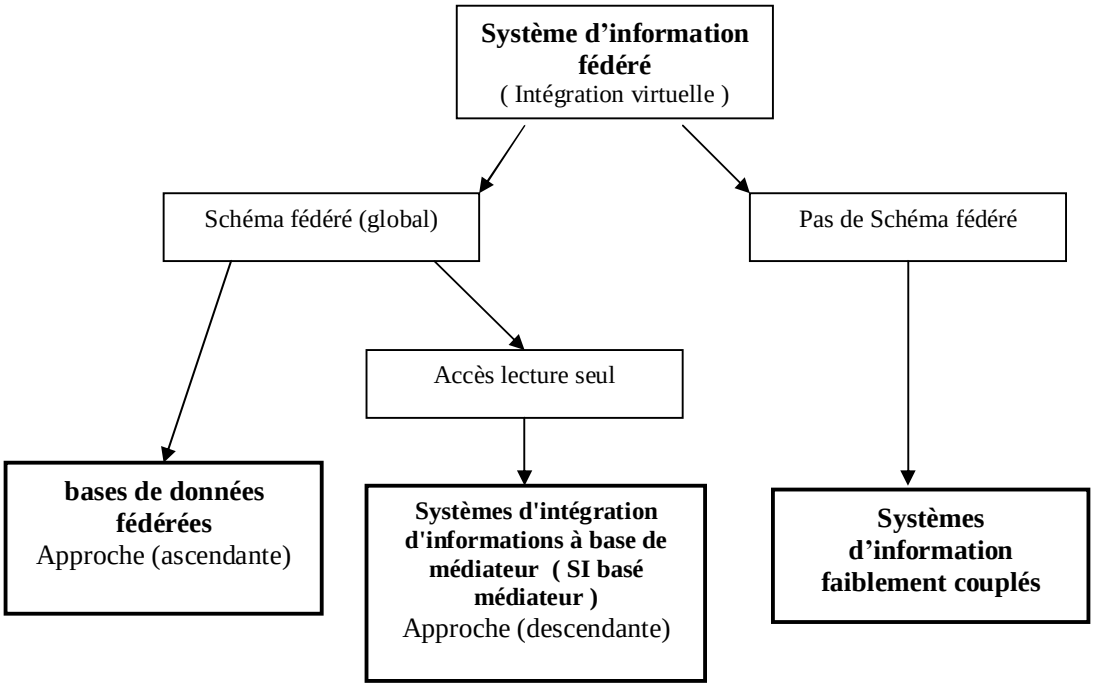


Figure 1.1 : Les approches d'intégration de données hétérogènes et distribuées

1.5.1 Systèmes d’information faiblement couplés

Ce type de systèmes sont basés sur l'utilisation d'un langage d'interrogation et d'un modèle de représentation commun, ils n’offrent pas un schéma fédéré (global). Chaque système d’information exporte ses informations sous la forme d'un schéma décrit dans le modèle commun, le langage multibases permet une interrogation multi-sites.

Les systèmes réalisés selon cette approche sont extensibles mais n'offrent pas de transparence à la localisation, et aussi la résolution des conflits schématiques et sémantiques reste entièrement à la charge de l'utilisateur[BUSS 99][CULL 03]. Parmi les travaux de cette approche :

Le projet Garlic[CODY 95][JOUA 01]. Dans cette solution une interface objet (un schéma objet) est fournie pour chaque système d'information par un module adaptateur pour accéder aux données locales. Les interfaces sont interrogées par un langage de requêtes dérivé d'OQL. Cette solution offre un module d'optimisation de requêtes génère un plan d'exécution en tenant compte des capacités d'interrogation de chaque adaptateur. Elle prise en charge seulement la résolution des conflits syntaxiques.

1.5.2 Bases de données fédérées

Ce type étendre le schéma classique à trois niveaux à un schéma sur cinq niveaux permettant lui accès aux données de systèmes d’information hétérogènes et distribués. Les cinq niveaux de représentation [SHET 98] sont (figure 1.2). Le premier niveau est la représentation du schéma local de chaque système dans son modèle local. Le second niveau est l'expression de ce schéma local dans le modèle commun retenu par la fédération. Le troisième niveau est un ensemble de schémas d'export qui présente les informations sous des formats différentes des informations. Le quatrième niveau correspond aux schémas fédérés obtenus par intégration de schémas d'exports. Le cinquième niveau est constitué de vues dérivées d’un schéma pour s’adapter à des catégories d’utilisateurs[JOUA 01]. Dans ce type (l’approche bases de données fédérées), la transparence est assurée, mais elle ne prend pas en compte les conflits sémantiques de manière explicite. En effet, l’inconvénient majeur de cette approche est qu’elle n’est pas extensible et flexible aux changements des schémas sources, une modification aux schémas sources entraîne une régénération totale du schéma fédéré. Parmi les travaux de cette approche:

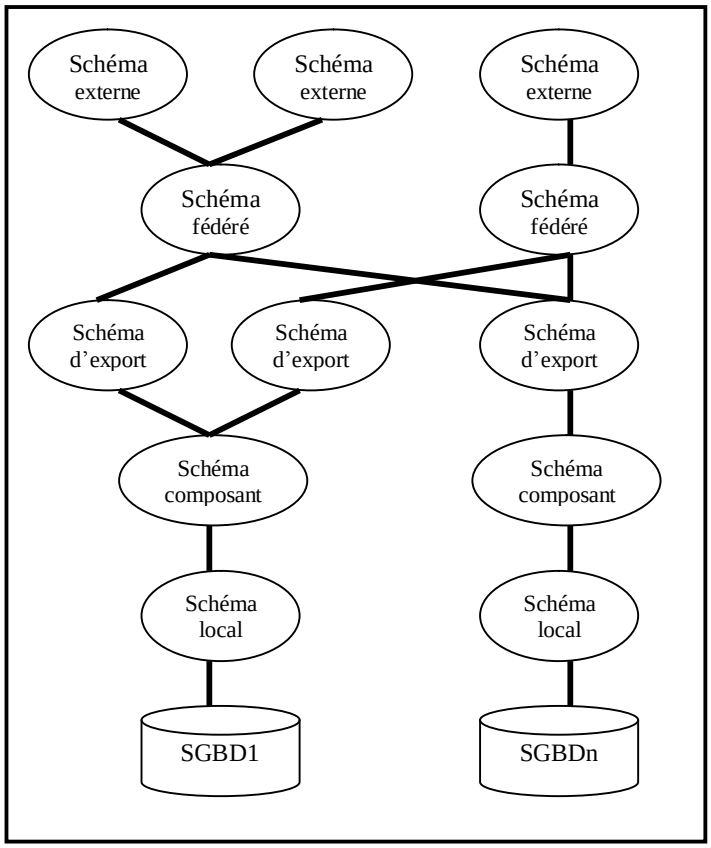


Figure 1.2 : les cinq niveaux de représentation[SHET 98]

Le projet MIND (Metu Interoperable DBMS) [DOGA 96]. Dans ce projet, les deuxième et troisième niveaux de l'architecture en cinq niveaux sont combinés en un seul niveau composé de schémas d'export exprimés dans le modèle commun (extension du langage de définition d’interface IDL de OMG). L’intégration de schémas repose sur une architecture à objets distribués et se fait en quatre niveaux.. Ce projet n’utilise aucune information sémantique explicite pour simplifier la construction des schémas d'export. Cette solution utilise des règles de mises en correspondance dérivées de requêtes OQL relient les éléments du schéma fédéré aux schémas d'export. Les systèmes de bases de données sources sont encapsulés dans des objets CORBA. Ce projet accepte des requêtes d’interrogations et de mises à jours.

1.5.3 Systèmes d'intégration d'informations basé médiateur

La plupart des projets réalisés selon l’approche de type médiation [CULL 03][JOUA 01]. Cette approche étende la fédération en ajoutant davantage de flexibilité.

La médiation est basée sur deux concepts : le médiateur et le wrapper. Le médiateur est une interface entre l'utilisateur et la collection de sources de données lui donnant l'illusion d'interroger un système d'information homogène. Le médiateur est chargé de répondre aux requêtes à partir de connaissances mise à sa disposition. Il permet de localiser l'information et de résoudre les conflits des données.

Le wrapper est une interface permettant la traduction des informations entre le médiateur et les sources de données. Il traduit les requêtes posées par le médiateur en des requêtes compréhensibles par les sources de données. Ensuite, il traduit les réponses envoyées par les sources de données en des réponses compréhensibles par le médiateur (figure 1.3). Cette approche assure la transparence et peut prend en compte les conflits sémantiques de manière explicite. Dans cette approche il existe deux méthodes d'expliciter la sémantique :

1- Dans la première méthode les conflits de données résolus statiquement. Il consiste à associe au médiateur un ensemble de connaissances permettant de localise les sources de données. Le traitement d'une requête suive un plan d'exécution établi par des règles qui définissent où se trouvent les données pertinentes pour traiter une requête. Le rôle du médiateur est de diviser, en fonction du schéma global, les requêtes utilisateur en plusieurs sous requêtes supportées par les sources pour recomposer à la fin la réponse. Dans cette méthode le coût d'évaluation d'une requête est plus réduit que dans la deuxième méthode et elle assure aussi plus de fiabilité.

2- Dans la deuxième méthode le médiateur identifie et localise, transforme et intègre les données pertinentes selon la sémantique associée à une requête. Cette méthode est plus adaptée aux requêtes de type recherche d'information (comme le Web sémantique) et dans des environnements subis à des changements très fréquents. L'inconvénient de cette méthode est le coût élevé pour traiter une requête, à chaque fois qu'on pose une requête, il est nécessaire de faire des réconciliations sémantiques pour trouver des règles de similitudes afin de traiter la requête. Un autre inconvénient consiste de la nature de cette méthode, la surveillance humaine pour valider les règles de similitudes extraites est

impossible[HAKI 03]. En conséquence, le manque de surveillance humaine rend cette méthode moins fiable.

La plupart des projets réalisés par l'approche de type médiation ont des limitations à l'utilisation d'une approche centralisé[SERA 98], aussi, il n'existe aucune coopération entre les wrappers. Le médiateur offre le seul point d'accès et contrôle l'exécution de toutes les requêtes. Dans les applications important le médiateur devient surcharger et cela engendre des problèmes de performance et de scalabilité du système. Le fait q'un système soit centralisé peut provoquer des problèmes de blocages et un embouteillage[ZARO 04]. Parmi les travaux de cette approche :

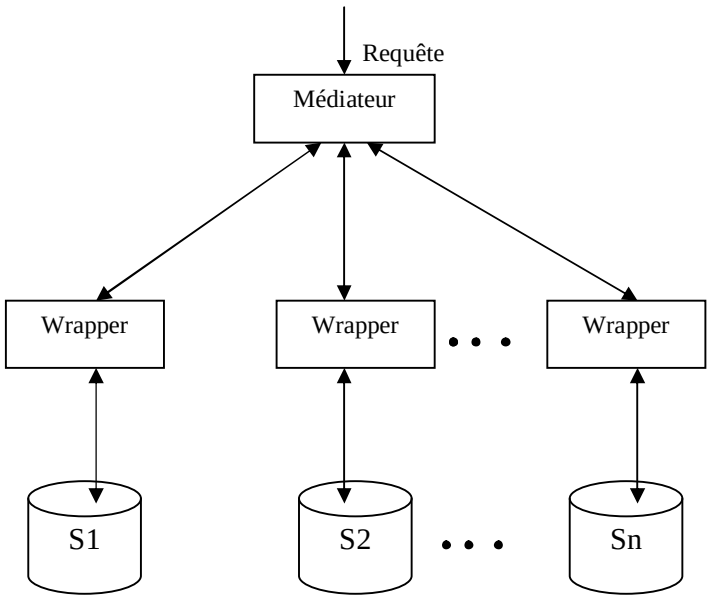


Figure 1.3 : Architecture d'un système d'intégration d'informations à base de médiateur[BONN 99]

Le projet **MOMIS** [BERG 02]. L'architecture du système est composée d'un modèle de données objet global ODL sous-ensemble du modèle ODMG, de wrappers, d'un médiateur et d'un processeur de requêtes. Les sources de données peuvent être structurées ou semi-structurées. Il utilise une logique de description riche (ODL- I3) pour décrire les schémas des sources de données à intégrer. MOMIS utilise un thésaurus commun construit par analyse des descriptions des données ODL et en exploitant une logique de description dérivée de la famille KL-ONE. Ce thésaurus sert à l'identification des informations sémantiquement liées afin de faciliter leur intégration.

Le projet **TSIMMIS** [GARC 97]. Le système utilise un modèle à objet complexes OEM comme modèle global, et un langage de règles MSL pour l’interrogation des données. MSL est aussi utilisé comme langage de spécification de haut niveau des médiateurs et wrappers, favorisant ainsi leur génération automatique. TSIMMIS permet de définir plus d’un médiateur. TSIMMIS n'apporte aucun support concernant la résolution des conflits sémantiques.

1.6 Conclusion

Dans ce chapitre nous avons expliqué les dimensions de classification des SIs et les problèmes d’hétérogénéité des données qui empêchent une intégration souple des données.

Ensuite nous avons présenté la classification des SIs suivant les trois dimensions (hétérogénéité, distribution et autonomie) et après on a donné les critères de classifications des SIF, ces critères précisent les différentes variantes des SIF.

Plusieurs solutions ont été proposées pour résoudre le problème de l’intégration de données (intégration virtuelle), on a étudié les avantages et les inconvénients de chaque approche. D’après cette étude on a trouvé que chaque approche (solution) offre quelques avantages au détriment d’autres. Durant les chapitres qui suivent, nous expliquerons d’une manière claire et précise les caractéristiques de notre solution.

L’exécution des requêtes dans des bases de données hétérogènes conduit à de nombreux problèmes qui peuvent se classer en deux catégories : l'intégration des données hétérogènes et le traitement des requêtes. Dans le chapitre suivant nous étudierons les problèmes liés à l’intégration de données hétérogènes.

Chapitre 2 : Sémantique des informations

2.1 Introduction

L'intégration de sources de données n'est pas une chose facile. En effet, les sources sont souvent hétérogènes car elles ont été définies indépendamment les unes des autres. Le processus d'intégration doit donc permettre de traiter des sources qui ont des modèles de données et/ou des schémas différents. Un problème essentiel est de gérer l'hétérogénéité et la distribution des différentes sources de données.

L'intégration de sources de données nécessite des outils qui prennent en charge ces problèmes de distribution et d'hétérogénéité et qui fournissent un accès efficace à un ensemble de sources de données.

L'objectif de ce chapitre est d'étudier les différents conflits de données et les modèles de données utilisés pour homogénéiser la représentation des données hétérogènes et ainsi les outils utilisés pour la représentation de la sémantique des informations. Nous représenterons aussi le modèle de données XML Schéma et le langage d'interrogation XQuery adoptés par notre solution.

2.2 L'hétérogénéité des données

Les conflits des données sont issus des différences de modélisation et d'interprétation des entités du monde réel par les concepteurs de système d'information.

Plusieurs travaux ont proposé des taxonomies des conflits de données [SHET 98][BUSS 99][PITO 95][GENO 02]. Nous allons présenter une classification détaillée des conflits qui peuvent se présenter lorsqu'on intègre des sources de données hétérogènes. Cette classification est inspirée de celle proposée dans [JOUA 01], elle permet d'illustrer les différents conflits possibles. Les différents conflits de données sont classifier en trois niveaux: les conflits syntaxiques, les conflits structurels (schématiques) et les conflits sémantiques. Nous décrivons chaque niveau de ces conflits.

2.2.1 Conflits syntaxiques

Les conflits syntaxiques sont liés au modèle de données utilisé pour représenter les informations. Une même information peut être représentée par différents modèles avec une syntaxe et des concepts différents. De nombreux modèles de représentation existent il en résulte de très nombreux conflits syntaxiques. Les solutions d'interopération résolvent ces conflits syntaxiques en définissant un modèle commun de données [SHET 98].

La propriété essentielle d'un tel modèle commun est de posséder un nombre suffisant de concepts pour représenter la sémantique des différents modèles des sources de données avec lesquels les informations sont exprimées.

2.2.2 Conflits structurels (ou schématiques)

Les conflits structurels sont liés, aux choix de conception et la manière de représenter les informations pour chaque système. Ces conflits apparaissent lorsque le même élément du monde réel est perçu avec des structures différentes. Par exemple [BENS 99] un concessionnaire automobile modélisera le propriétaire d'un véhicule avec un sous-ensemble d'attributs dans une relation Véhicule alors que le service de gestion des cartes grises modélisera un propriétaire de véhicule avec une relation Personne liée par une relation de dépendance à une table véhicule.

Il existe également des conflits de généralisation ils sont produits par des différences dans les liens de généralisation / spécialisation entre entité. Par exemple un système ne peut utiliser qu'un seul concept pour représenter tous les types de routes tandis qu'un autre emploiera deux concepts pour distinguer les entités autoroute et route départementale liées à un concept plus général route.

Des conflits de type de données peuvent aussi intervenir lorsqu'une même valeur est représentée par des types différents. Par exemple la date de construction d'un produit peut être représenté par une chaîne de caractères « Janvier 2005 » ou par un type composé de deux champs mois et année 01/2005.

2.2.3 Conflits sémantiques

Les conflits sémantiques proviennent des hétérogénéités de description des informations. Deux grandes catégories de conflits sémantiques[JOUA 01]: Les conflits taxonomiques sont liés aux hétérogénéités de vocabulaire et les conflits de valeurs sont liés à l'utilisation et au codage de la valeur d'une donnée.

Les conflits taxonomiques sont issus de l'autonomie des systèmes à nommer et à classifier les entités du monde réel. Ils concernant en particulier les problèmes de synonymie (des termes différents expriment la même réalité), d'homonymie (deux termes identiques expriment des réalités différentes), de polysémie (un terme change de signification en fonction du contexte), d'hypernymie et d'hyponymie (un terme peut être plus général ou plus spécifique qu'un ou plusieurs autres termes)[JOUA 01].

Les conflits de valeurs sont liés à la marnière de coder la valeur d'une entité du monde réel dans différents systèmes. Des valeurs différentes d'un même information peuvent être considérées comme similaires selon le contexte.

2.3 Modèle de représentation des données hétérogènes

Comme nous avons dit précédemment, l'utilisation d'un modèle de représentation commun assure d'une part la compréhension du schéma des informations et d'autre part un accès uniforme à l'ensemble des données distribuées. On trouve dans la littérature[DANG 03][TAME 99][ROUS 02][BUSS 99] différents modèles de données sont utilisés pour représenter les données intégrées. Il existe quatre catégories de modèles:

2.3.1 Un modèle relationnel

Une représentation du monde réel basé sur le concept de relation au sens mathématique de sous-ensemble d'un produit cartésien d'une liste de domaines. L'avantage de ce modèle est son utilisation dans de nombreux systèmes d'information et son nombre restreint de concepts. La simplicité du modèle relationnel, si elle est avantageuse pour la facilité d'emploi et pour ses fondements mathématiques, est néanmoins limitative. De nombreuses données sont trop complexes pour être représentées par des tables et des requêtes relationnelles.

2.3.2 Un modèle orienté objet

Une représentation du monde réel basé sur les concepts d'objet (données + traitements ou méthodes agissant sur ces données sont assemblés dans une même entité informatique baptisé objet) et de hiérarchies de classes (concept d'héritage et de réutilisation). Un schéma objet permet de représenter une partie de la sémantique des informations. En effet les attributs d'un objet, ses fonctionnalités, ses relations d'héritage et de composition définissent sa sémantique[JOUA 01].

2.3.3 Un modèle logique

Ce modèle peut être combiner avec des concepts orienté objets[JOUA 01]. Ce modèle est utilisé pour représenter des concepts qu'un modèle classique ne le permet pas. Il existe deux types du modèle logique : les modèles basés sur la logique de description adaptée à une représentation taxonomique des informations et les modèles basés sur la logique déductive.

2.3.4 Un modèle propriétaire

Est utilisé pour intégrer directement dans le modèle des concepts adaptés à la représentation de données particuliers (par exemple les données spatiales). Ce type de modèle permet une représentation plus complète de données pour des concepts spécifiques à un domaine particulier.

2.3.5 Un modèle semi structuré

La tendance actuelle s'oriente vers le modèle semi structuré XML[DANG 03]. XML (eXtensible Markup Language) est une recommandation du W3C. C'est un langage à balise définissant un format universel de représentation des données. Un document XML contient à la fois des données et les indications sur le rôle que jouent ces données. Ces indications permettent de déterminer la structure du document : ce sont des balises.

XML est un format largement répandu et accepté par la communauté informatique. De nombreuses sources de données sont disponibles et de nombreuses applications permettent d'exporter leurs données en XML. XML offre deux techniques pour créer une structure de document XML[GIRAR 01]:

2.3.5.1 Les DTDs

DTD signifie Document Type Definition (ou définition de type de document), c'est la grammaire historique des documents XML. La puissance de description des DTD est faible : une DTD permet uniquement de décrire la structure d'un document XML (liste des balises et organisations des balises), et non la typologie des données contenues (chaîne de caractère, date, entier, etc)[GIRAR 01].

2.3.5.2 Les Schémas XML

Un schéma XML[THOM 01][CHAZ 01] a le même rôle qu'une DTD, c'est à dire définir la structure d'un document XML. L'objectif des schémas XML est de proposer une solution aux inconvénients majeurs des DTD[COST 02][CHAZ 01]:

1. Une syntaxe différente que celle de XML : on doit donc écrire un document XML dans un langage et la DTD dans un autre;
2. La notion d'espace de noms n'existe pas (pour faire face aux risques de collisions de balises);
3. Pas de notion de type de données;
4. Pas de notion d'héritage: Les systèmes orientés objet s'appuient sur l'idée de décrire de nouveaux objets à partir d'objets existants. Ceci n'est pas possible.

Un schéma XML est composée principalement d'éléments qui sont représentés par les balises. Ces éléments sont associés à un type de données qui peut être défini comme « type simple » ou « type complexe ».

Exemple de document XML Schema :

Par exemple, pour modéliser l'élément <entree> défini par <nom> (string) et <telephone> (decimal). En utilisant XML Schema.

Le document XML :

```
<entree>
<nom>Zireg Mohamed</nom>
<telephone>0102030405</telephone>
<nom>Zideni Ahmed</nom>
<telephone>0112030405</telephone>
</entree>
```

Le document XML Schema :

```
<schema xmlns:="http://www.w3.org/2000/10/XMLSchema">
<element name="entree">
  <complexType>
   <sequence>
        <element name="nom" type="xsd:string" minoccurs=”1” maxoccurs=”1”/>
        <element name="telephone" type="xsd:decimal"/>
   </sequence>
  </complexType>
 </element>
</schema>
```

Les types de données simples ne peuvent pas comporter d'attributs ou contenir de sous-éléments. Les schémas XML ont à disposition un nombre important de types de donnée, contrairement à une DTD. L’attribut *type* peut prendre une valeur parmi : string, boolean, float, decimal, integer, time, month , year, ID,...

La puissance de XML Schema réside surtout dans sa capacité à pouvoir définir des types personnels (des types complexes)[COST 02]. Ces types complexes sont créés en utilisant l'élément *complexType* composé généralement d'une série de déclarations d'éléments et d'attributs et de références d'éléments. Ces déclarations sont rassemblées dans des groupes modèles permettant de définir, un modèle de contenu constitué d'une séquence sequence (ensemble ordonné), d'un choix exclusif (choice) ou d'un ensemble non-ordonné (all).

Un élément (ou un attribut) est déclaré globalement lorsqu'il est fils de l'élément *schema*. Si on a déclaré un élément ou un attribut global, il peut être référencée dans une ou plusieurs déclarations en utilisant l'attribut *ref*. Les éléments ou attributs qui est déclarés à l'intérieur d'un type complexe, sont dits déclarés localement. Les éléments sont déclarés en utilisant le mot-clef *element* et les attributs sont déclarés en utilisant le mot-clef *attribute*.

XML Schema propose deux techniques de dérivations de types : la première appelée dérivation par restriction du type ancêtre, et la seconde appelée dérivation par extension du type ancêtre. Ces deux mécanismes de dérivation s'appliquent aux deux types de données simples ou complexes, ce qui les rend très puissants[CHAZ 01].

Le nombre minimum (resp. maximum) de fois qu'un élément peut apparaître est déterminé dans sa déclaration par la valeur de l'attribut minOccurs (resp. maxOccurs). Cette valeur peut être un entier positif ou encore le mot unbounded qui signifié qu'il n'y a pas de valeur limite. La valeur par défaut dans les deux cas (minOccurs et maxOccurs) est 1.

XML Schema fournit plusieurs fonctionnalités s'appuyant sur XPath et permettant de décrire des contraintes d'unicité et des contrôles référentiels[CHAZ 01][THOM 01]. La première est une déclaration simple d'unicité et utilise l'élément "*unique*". La seconde déclaration, "*Key*" est similaire à "*unique*" et spécifie en plus que cette valeur unique pourra être utilisée comme une clé, ce qui lui donne deux contraintes supplémentaires: elle doit être non nul et être référençable. La déclaration, "*Keyref*" définit une référence à une clé.

2.3.5.3 Langages d'interrogation pour XML

Il y a eu de nombreuses propositions de langages de requêtes pour XML (XPath, XQL, XSLT, Quilt,…)[GIRAR 01]. Récemment, le W3C a proposé le langage XQuery [W3C 01] qui est plus adopté pour l'interrogation des schémas XML. XQuery reprend les avantages de XPath, XML-QL et XQL[DANG 03].

Ce langage a été conçu pour permettre de créer des requêtes précises tout en pouvant s'adapter à tout type de source de données XML, qu'il soit bases de données ou documents. Comme OQL, XQuery est un langage fonctionnel où chaque requête est une expression. XQuery est issue du langage de requête Quilt qui lui même empruntait de nombreuses

fonctionnalités de XPath, XQL, XML-QL SQL, OQL. Il adopte de XPath et XQL la syntaxe de l'expression chemin (path expression) qui s'accorde avec les documents hiérarchiques. Il prend de SQL l'idée des séries de clauses basées sur les mots clés qui produisent une forme pour restructurer des données (Select-From-Where dans SQL)[W3C 01][BOUZ 03].

L'élément de base du langage XQuery est l'expression. Tout en XQuery est une expression. Un programme ou script XQuery n'est rien d'autre qu'une expression, avec en option des fonctions ou définitions.

Le langage reconnaît de nombreuses sortes d'expressions qui peuvent être construites à partir de mots-clés, de symboles ou d'opérandes. Il existe sept types d'expressions[W3C 01]: expression de chemin, constructeurs d'éléments, expressions FLWR, expressions mettant en jeu des opérateurs et des fonctions, expressions conditionnelles, expressions quantifies et expressions qui testent ou modifient les types de données.

a) Expressions de chemin

Une expression de chemin dans XQuery est basé sur la syntaxe de XPath. XPath est une notation pour accéder aux chemins dans un document XML utilisée comme requête peut être appliquée à un document ou une collection de documents. Son résultat sera un ensemble de sous-arbres répondant aux critères de l'expression. Par exemple :

Pour trouver tous les titres de chapitres dans un document livres.xml:
document("livres.xml")/chapitre/titre

b) Constructeurs d'éléments

Utilisés quand une requête doit créer de nouveaux éléments. Les accolades { } permettent de placer une expression au sein d'un constructeur. Par exemple :
Génère un élément <livre> dont un sous élément est <titre> et qui contient l'attribut "année":

<livre>
{ $b/@annee }
{ $b/titre }
</livre>

La variable $b est rattachée à une autre partie de la requête. Quand l'ensemble de la requête est lancée, le constructeur ci-dessus donnera un résultat sous la forme suivante:

<livre annee="2005">
<titre> An XML Query Language </titre>
</livre>

c) Les expressions FLWR (prononcé flower)

Est similaire à la construction Select-From-Where de SQL, et compose le squelette de l'expression XQuery. Le nom provient de for, let, where et return.

- for : Fourni un mécanisme d'itération.
- let : Permet l'assignation de variable.
- where : Les clauses for et let génèrent un ensemble de noeuds qui peuvent être filtré par un ou plusieurs prédicats dans une clause where.
- return : génère le résultat de l'expression FLWR. Elle contient généralement un ou plusieurs éléments constructeurs et/ou des références à des variables, et est exécutée une fois pour chaque noeud renvoyé par les clauses For/Let/Where.

Un exemple simple d'une requête FLWR. Cette requête a pour but de trouver les titres des livres apparus en 2004 dans la librairie library1.

```
FOR $a IN document("library1.xml")//book
WHERE $a/years = ‘2004’
RETURN $a/title
```

d) Expressions utilisent des opérateurs et des fonctions

XQuery inclut un grand nombre de fonctions et d'opérateurs[W3C 01]. On retrouve dans XQuery les opérateurs arithmétiques, de comparaison, booléens et reliés à la séquence. Il existe des fonctions pour : chaînes de caractères, mathématiques, comparaison de dates, expressions régulières, noeuds XML, conversion de types, etc... Il est aussi possible de que les utilisateurs définir leurs propre fonctions.

e) Expressions conditionnelles et expressions quantifiées

Comme dans la majorité des langages de programmation, XQuery autorise l'utilisation des expressions IF / THEN / ELSE. La syntaxe de cette expression est[W3C 01]: IfExpr ::= "if" "(" Expr ")" “then” Expr “else” Expr .

Une expression quantifiée dans XQuery est exprimée par SOME et EVERY.

- En utilisant l’expression SOME, il est possible de déterminer si au moins un noeud d'un bloc de nœuds correspond à un prédicat. SOME possède la syntaxe, SomeExpr ::= "some" Variable "in" Expr "satisfies" Expr,
- L'expression EVERY permet quant à elle de tester si tous les noeuds d'un bloc correspondent à un prédicat. EVERY possède la syntaxe, EveryExpr ::= "every" Variable "in" Expr "satisfies" Expr

f) Expressions manipule les types de données

XQuery est basés sur le système de typage de XML Schema (nombres, booléens, chaînes, dates, temps, durées...) et utilise aussi les types de données complexe qui sont définis par l’utilisateur. Les expressions INSTANCEOF et TYPESWITCH/CASE sont utilisées pour tester si une occurrence est d’un type donné[W3C 01].

2.4 Les outils pour la représentation de la sémantique des informations

Dans la littérature [SHET 98][CULL 03], on trouve trois types d'outils utilisés pour la représentation de la sémantique des informations : les méta-données, les ontologies et le contexte (considéré comme combinaison des méta-données et d'ontologies).

2.4.1 Les méta-données

Les méta-données sont des "données sur les données", Elles sont décrites soit de façon littérale soit par des attributs qui décrivent les propriétés intrinsèques des entités considérées[CULL 03]. Une méta-donnée permet de donner du sens au contenu des ressources de manière à ce que leur localisation et interrogation soient plus aisées et plus pertinentes. Pour représenter explicitement la sémantique des informations et aider à la détection et à la résolution des conflits sémantiques, une solution potentielle importante aux problèmes d'hétérogénéité sémantiques, qui est l'ontologie[HAKI 03].

2.4.2 Les ontologies

Le terme ontologie est issue du domaine de la philosophie où il signifie « explication systématique de l'existence »[TIXI 01]. Dans le cadre de l'ingénierie des connaissances, différentes définitions ont été établies. Une définition très générale a été formulée par Gruber « une ontologie est une spécification explicite d'une conceptualisation » [TIXI 01] . Une conceptualisation étant définie comme étant « une structure sémantique intensionnelle qui capture les règles implicites contraignant la structure d'un morceau de réalité ».

On trouve que d'autres auteurs apportent des définitions fondées sur l'approche qu'ils ont adoptée pour construire leurs ontologies. Il peut s'agir d'une approche linguistique pour Swartout: « une ontologie est un ensemble de termes structuré de façon hiérarchique, conçu afin de décrire un domaine et qui peut servir de charpente à une base de connaissances ». Il peut également s'agir d'une approche formelle : Guarino et Giaretta ont défini l'ontologie comme étant « une théorie logique qui rend compte partiellement mais explicitement d'une conceptualisation »[TIXI 01].

En effet, une ontologie définit un ensemble organisé (souvent sous forme de taxonomie) de concepts utilisables pour formuler des connaissances.

Un concept représentent un constituant de la pensée, né de l'esprit pour lequel est explicitée une sémantique évaluable et communicable[TIXI 01].

2.4.2.1 Le rôle des ontologies

Les ontologies permettent de rendre différents services, tels que la définition d'un vocabulaire permettant de formuler des savoirs, la spécification plus ou moins formelle du sens des termes et des relations entre les termes.

Les ontologies sont utilisées dans différents domaines: représentation d'informations et connaissances, intégration de systèmes d'informations, spécification de systèmes, etc. Selon [JASP 99] l'ontologie peut servir les objectifs de :

Communication (humains et organisations) : Le bénéfice d'usage d'ontologie ici est d'éliminer l'ambiguïté. Dans l'ontologie, il n'y a jamais deux termes ayant la même sémantique, au contraire si l'on utilise un langage naturel pour la communication.

Interopérabilité (machines et systèmes) : L'ontologie sert à définir le format d'échange entre les systèmes. L'ontologie peut être utilisée comme un modèle intermédiaire pour la traduction entre les modélisations différentes collections d'objets.

Ingénierie des systèmes : L'ontologie peut servir divers aspects du développement des systèmes d'informations. Elle peut assister le processus de construction de spécification de système. L'usage d'ontologie rend les documents du processus plus compréhensible, évite l'ambiguïté dans la spécification. En outre, une représentation formelle d'ontologie permet d'un traitement automatique du développement. Elle soutient également l'automatisation du processus de vérification de fiabilité de systèmes.

Les ontologies ont à l'heure actuelle un rôle clé dans les travaux du web sémantique, et donc paraissent essentielles pour la problématique bien spécifique de la gestion sémantique des ressources documentaires pour une entreprise. En effet, elles peuvent servir

de ressources pour représenter le sens de différents contenus échangés dans des systèmes d'informations[TIXI 01].

Pour pouvoir utiliser puis exploiter une ontologie, deux problèmes essentiels se posent : (1) L'élaboration de l'ontologie, il est nécessaire de trouver une méthodologie permettant sa construction, car il s'agit d'une tâche complexe surtout si l'ontologie est de grande taille. (2) Pour pouvoir exploiter une ontologie dans un système informatique opérationnel, il est nécessaire de la représenter au moyen de langages plus ou moins formels.

2.4.2.2 Modèles de représentation des ontologies

Il existe plusieurs modèles et langages de représentation pour la modélisation d'ontologie. [JENH 03] a classé les différents langages formels en fonction des modèles sur lesquels ils reposent. Parmi les langages de représentation développés au niveau conceptuel, trois grands modèles sont distingués: les logiques de description, les graphes conceptuels et les langages de frames.

1- Les logiques de description

Les logiques de description permettent de représenter les connaissances sous forme de concepts, de rôles et d'individus. Les rôles sont des relations binaires entre concepts et les individus sont les instances des concepts. Les propriétés des concepts, rôles et individus sont exprimées en logique des prédicats, en particulier les propriétés de subsomption[JENH 03]. Les concepts, les rôles et leurs propriétés sont définis au niveau terminologique. Au niveau factuel sont exprimés les faits portant sur des individus. LOOM et KL-ONE sont des exemples de systèmes implémentant ce modèle. D'autres langages appartenant à la même famille que celle de KL-ONE par exemple: CLASSIC et DAML+OIL. Tous les langages de la famille KL-ONE sont des langages hybrides: Les buts des langages hybrides est de donner une généralisation des réseaux sémantiques, langage de frames et permettre à un système de raisonner sur son état de connaissance[JENH 03].

2- Les graphes conceptuels

Les graphes conceptuels sont décomposés en deux niveaux: le premier niveau est le niveau terminologique où sont décrit les concepts, les relations et les instances de concepts, ainsi que les liens de subsomption entre concepts et entre relations. Le deuxième niveau est le niveau assertionnel où sont représentés les faits, les règles et les contraintes sous forme de graphes où les sommets représentent des instances de concepts et les arcs représentent des relations.

3- Langages de frame

Introduit dés les années 70 par Minsky comme une modélisation de base pour la représentation de connaissances dans le domaine d'Intelligence Artificielle (AI)[JENH 03]. Une « frame » est un objet nommé, qui est utilisé pour représenter un certain concept dans un domaine et est dotée d'attributs (slots), qui peuvent porter différentes valeurs (facets), et d'instances. Le fondement logique pour les langages de frame est le formalisme F-logic. Il permet de comprendre le modèle sémantique dans tous les langages de frame et d'aider à la construction d'une base de connaissances. Parmi les langages de frame on peut citer : Ontolingua, RDF, RDF schéma et DAML.

2.5 Conclusion

Dans ce chapitre nous avons présenté les différents conflits de données et on a vu la nécessité d'utilisation d'un modèle de données commun pour homogénéiser la représentation des données hétérogènes.

La plupart des études sur les systèmes de gestion de données distribuées et hétérogènes utilisent le modèle relationnel ou le modèle objet comme modèle de données d'intégration[DANG 03]. Notre approche est d'utiliser le XML Schema comme modèle de données commun pour éliminer les conflits syntaxiques.

Les avantages de l'utilisation de XML Schema comme modèle commun tiennent à la richesse de ce modèle: clarté, et extensibilité, ainsi son caractère général rend aisée la traduction des modèles de données existants. Le langage XQuery offre l'interrogation

efficace des requêtes sur XML. Nous avons considéré XQuery comme langage commun d'interrogation.

On a vu aussi les outils utilisés pour la représentation de la sémantique des informations. La sémantique intervient dans la construction d'un système d'interopération de systèmes d'information et dans son interrogation. Les informations sémantiques doivent permettre de comprendre sans ambiguïtés la signification des informations partagées et échangées. Les ontologies sont largement répondu notamment dans le Web sémantique pour éliminer l'hétérogénéité sémantique. Pour capturer la sémantiques des informations nous avons adopté d'utiliser les ontologies qui ont un pouvoir d'expression très puissant pour expliciter la sémantique des informations.

Le choix d'un modèle de représentation d'une ontologie est guidé par le type des informations à représenter. Pour cela on a adopté les logiques de description afin de représenter les ontologies de notre solution.

Dans le chapitre suivant nous concentrerons sur les problèmes de traitement (l'évaluation) des requêtes sur les systèmes à base d'un médiateur.

Chapitre 3 : Traitement des requêtes sur les systèmes à base d'un médiateur

3.1 Introduction

L'exécution des requêtes dans des bases de données hétérogènes et distribuées conduit à de nombreux problèmes qui peuvent se classer en deux catégories: l'intégration des données hétérogènes et le traitement des requêtes. L'intégration des données hétérogènes est confrontée à de nombreux conflits de données (conflits syntaxiques, schématiques et sémantiques). Le traitement d'une requête posée indépendamment de la localisation des différentes données intervenant pour calculer le résultat, cela introduit les difficultés : de la reformulation d'une requête, la décomposition d'une requête et la recomposition des résultats et le niveau de l'optimisation.

L'objectif principal de ce chapitre consiste à présenter le problème de traitement des requêtes dans les systèmes à base d'un médiateur et étudier les différentes approches proposées.

3.2 Traitement d'une requête

Le rôle d'un système de médiation consiste à reformuler la requête d'un utilisateur en termes de schémas sources, puis à la décomposer en sous requêtes, qui seront envoyées aux différentes sources. Les résultats sont ensuite renvoyés au médiateur, qui les intègre avant de les renvoyer à l'utilisateur.

3.2.1 La reformulation d'une requête

Une des différences principales entre un système d'intégration de données à base d'un médiateur et un système traditionnel de base de données est que les utilisateurs posent leurs requêtes en termes du schéma global. Les données sont stockées dans les sources de données, organisées sous des schémas sources. Par conséquent, pour répondre aux requêtes, il est nécessaire d'établir une correspondance entre le schéma global et les schémas des sources de données.

Le processeur de requête doit être capable de reformuler une requête écrite sur le schéma global en une requête écrite sur les schémas des sources de données.

On distingue dans la littérature trois approches [LENZ 03][LEVY 00] pour établir la correspondance entre le schéma global et les schémas des sources de données à intégrer (règles de mapping).

3.2.1.1 L'approche GAV

Un schéma global est considéré comme une vue sur des schémas sources. GAV (Global As View), a été la première à être proposée pour l'intégration de données.

Dans l'approche GAV, pour chaque relation utilisée dans le schéma global, on définit une vue composée des termes des relations des schémas sources. La transformation d'une requête écrite en utilisant le schéma global en une requête écrite en utilisant des schémas sources est une opération simple, cette simplicité est considérée comme avantage de l'approche GAV.

Les inconvénients de cette approche est qu'il suppose que les sources à intégrer soient connues à l'avance, ainsi une modification sur l'ensemble des sources ou sur leurs schémas entraîne une reconsidération complète du schéma global.

Nous donnons un exemple de l'approche GAV. Cet exemple est simple qui évite les problèmes d'hétérogénéités des sources.

Exemple :

Considérant le schéma global[LENZ 01]:

Schéma global: Film(*Titre, Année, Directeur*)
Européen(*Directeur*)
Révision(*Titre; Critique*)

Les schémas sources de deux sources sont :

Source 1: r1(*Titre, Année, Directeur*) de puis 1960, européen directeurs

Source 2: r2(Titre, Critique) depuis 1990.

Suivant l'approche GAV les règles de mapping consiste à définit chaque relation du schéma global par une requête (vue) sur les schémas sources. Les règles de mapping sont:

$\{ (T, A,D) \mid r1(T, A,D) \} \subseteq$ Film(T, A,D)

$\{ (D) \mid r1(T, A,D)\} \subseteq$européen(D)

$\{ (T,R) \mid r2(T,R)\} \subseteq$ révision(T,R)

Soit la requête posée par un utilisateur sur le schéma global :

Requête : Titre et critique du filmes en 1998. Elle est écrite comme suit:
$\{ (T,R) \mid$ Il existe D, film(T, 1998,D) $\wedge$ révision(T,R)$\}$; et représente l'ensemble :
$\{ (T,R) \mid$ film(T, 1998,D) $\wedge$ révision(T,R) $\}$

La reformulation de la requête dans GAV est une opération simple, puisque les relations du schéma global sont définies en termes des relations des schémas sources, nous devons seulement substituer les relations du schéma global par leurs définitions dans les règles de mapping, donc la requête $\{ (T,R) \mid$ film(T, 1998,D) $\wedge$ révision(T,R) $\}$ est reformulée en $\{ (T,R) \mid$ r1(T, 1998,D) $\wedge$ r2(T,R) $\}$

3.2.1.2 L'approche LAV

LAV (Local As View) Chaque relation d'un schéma source est définie comme vue sur le schéma global. Dans le cas d'une approche LAV, la requête sur le schéma global doit être reformulée suivant les schémas sources. Dans cette approche, chaque source est spécifiée de manière indépendante.

Cette approche a l'avantage qu'elle est très flexible par rapport à l'ajout (ou la suppression) de sources de données à intégrer, cela n'a aucun effet sur le schéma global, seules des vues doivent être ajoutées (ou supprimées). De l'autre coté, le prix à payer pour cette flexibilité et cette simplicité de mise à jour est la complexité de la construction des réponses à une requête dans un médiateur conçu selon l'approche LAV. La reformulation de requêtes en termes de vues est la seule possibilité pour obtenir des requêtes exprimées en termes de vues, que l'on doit ensuite exécuter pour interroger les sources de données[LEVY 00]

Exemple :

Le schéma global, les schémas sources et la requête sont les mêmes de l'exemple précédant. Suivant l'approche LAV les règles de mapping consiste à définir chaque relation d'un schéma source par une requête (vue) sur le schéma global. Soit les règles de mapping:

$r1(T, A, D) \subseteq \{ (T, A, D) \mid \text{film}(T, A, D) \wedge \text{européen}(D) \wedge A \geq 1960 \}$

$r2(T, R) \subseteq \{ f(T, R) \mid \text{film}(T, A, D) \wedge \text{révision}(T, R) \wedge A \geq 1990 \}$

La reformulation de la requête dans l'approche LAV est plus complexe que GAV, parce qu'il faut reformuler la requête $\{ (T,R) \mid \text{film}(T, 1998, D) \wedge \text{révision}(T,R) \}$ par un mécanisme d'inférence qui réécrit les relations du schéma global en relations des schémas sources.

La reformulation est :$\{ (T,R) \mid r2(T,R) \wedge r1(T, 1998, D) \}$

3.2.1.3 L'approche GLAV

GLAV (Global and Local As View) est une combinaison entre GAV et LAV prend les avantages des deux approches prétendantes[LEVY 00].

GLAV offre plus de flexibilité au mise à jour par les utilisateurs ou les sources locales. GLAV associe une requête conjonctive écrite sur le schéma global à une requête conjonctive écrite sur des schémas sources.

Exemple :

Considérant l'exemple suivant :

Le schéma global est[LENZ 03]:

Schéma global : Travail(Personne, Projet),
Région(Projet, Domaine)

Les schémas sources sont:

Source 1: A-un-Emploi(Personne, Domaine)

Source 2: Enseigner(Professeur, Course), Dans(Cours, Domaine)

Source 3: Obtenir(Chercheur, Allocation), Pour(Allocation, Project)

Les règles de mapping de l'approche GLAV consiste à associe une requête conjonctive écrite sur le schéma global à une requête conjonctive écrite sur des schémas sources.

Soit les règles :

$\{ (r, d) \mid \text{A-un-Emploi}(r, d) \} \subseteq \{ (r, d) \mid \text{Travail}(r, p) \wedge \text{Région}(p, d) \}$

$\{ (r, d) \mid \text{Enseigner}(r, c) \wedge \text{Dans}(c, d) \} \subseteq \{ (r, d) \mid \text{Travail}(r, p) \wedge \text{Région}(p, d) \}$

$\{ (h, p) \mid \text{Obtenir}(h, a) \wedge \text{Pour}(a, p) \} \subseteq \{ (h, p) \mid \text{Travail}(h, p) \}$

Soit la requête posée par un utilisateur sur le schéma global :

Requête : trouvez toutes les personnes travaillent sur le projet, *projet10.*

La requête est représentée par : $\{ h \mid \text{Travail}(h, \textit{pojet10}) \}$

Pour reformuler la requête il faut utiliser un mécanisme d'inférence qui réécrit les relations du schéma global en relations des schémas sources.

La reformulation de la requête est : $\{ h \mid \text{Obtenir}(h, a) \wedge \text{Pour}(a, \textit{projet10}) \}$

Répondre à une requête en utilisant des vues

L'utilisation des vues pour répondre à des requêtes est une technique utilisée pour résoudre de nombreux problèmes de bases de données : l'optimisation de requêtes, la gestion de l'indépendance physique des données, l'intégration de données et la conception d'entrepôts de données[HALE 01]. Le problème de reformulation d'une requête dans l'approche LAV peut être expliqué comme suit.

En raison de la forme des règles de mapping en LAV, chacune des sources peut être considérée comme elle contient une réponse à une requête écrite sur le schéma global. Par conséquent, les sources représentent des réponses matérialisées aux requêtes écrites sur le schéma global virtuel. Le problème est de trouver une méthode pour répondre à une requête d'un utilisateur (écrite sur le schéma global) en utilisant seulement les réponses des requêtes qui décrivent les sources.

Formellement, Soit la requête *Q* écrite sur un schéma de base de données, et soit l'ensemble des vues $V = \{ V1,\ldots,Vi,\ldots Vn \}$ chaque vue est définie sur le même schéma. La reformulation de la requête en utilisant seulement les vues est l'expression de la requête *Q'* où elle utilise seulement des vues dans sa expression. Il existe deux types de reformulations [LEVY 00] : reformulation équivalente (equivalent rewritings) et reformulation de contenance maximal (Maximal-containing rewritings).

Les notions de contenance de requête et d'équivalence de requête sont importantes afin de comprendre les définitions des types de reformulation.

Contenance et équivalence de requête, une requête *Q1* est dite contenue dans une requête *Q2*, notée par : $Q1 \subseteq Q2$, si le résultat de *Q1* est un sous ensemble de résultat du *Q2* pour chaque instance d'une base de données. Dans le cas où : $Q1 \subseteq Q2$ et $Q2 \subseteq Q1$, on dit que *Q1* et *Q2* sont équivalentes.

Reformulation équivalente, Soit *Q* est une requête et *V ={ V1,…,Vi,…Vn}* est un ensemble de vues définies. La requête *Q'* est une reformulation équivalente de *Q* en utilisant *V* si :

2- *Q'* référence seulement aux vues dans *V* ,
3- *Q'* est équivalente à *Q*

Reformulation de contenance maximal, Soit *Q* est une requête et *V ={ V1,…,Vi,…Vn}* est un ensemble de vues définies par le langage de requêtes *L*. la requête *Q'* est la reformulation maximale contenue dans *Q* en utilisant *V, si :*

1- *Q'* référence seulement aux vues dans *V* ,
2- *Q' est contenue dans Q,* Et
3- Il n'existe pas une reformulation *Q1*, tel que $Q' \subseteq Q1 \subseteq Q$ et *Q1* n'est pas équivalente à *Q'*.

La reformulation équivalente est nécessaire lorsque des vues matérialisées sont utilisées pour l'optimisation des requêtes (au sens des SGBD) et l'indépendance physique de données[HALE 01].

La reformulation de contenance maximal est utilisée dans le contexte d'intégration de données à base d'un médiateur, parce que les sources ne contiennent pas nécessairement toutes les informations requises pour fournir une réponse équivalente à la requête. De ce fait on cherche la reformulation de contenance maximal possible. Dans [PENS 02], une amélioration de cette méthode consiste à prendre en compte le cas où elle existe une vue contienne la requête, dans ce cas les sources ont toutes les informations nécessaires pour répondre à la requête.

3.2.2 La décomposition d'une requête et recomposition des résultats

Dans un système de médiation la requête est décomposée en sous requêtes destinées aux sources de données. Une fois les sous-requêtes soumises à chacune des sources, il s'agit de savoir recomposer les différents résultats entre eux pour obtenir un résultat final. Le médiateur doit décider quelle est la part du travail qui doit être effectuée par les sources de données et quelle est la part du travail qui doit être effectuée par lui.

3.2.3 Le niveau de l'optimisation

Pour une requête donnée dans un système traditionnel de gestion de base de données, il est possible de considérer plusieurs plans d'exécution équivalents donnant le même résultat. Ces plans non pas le même coût, en termes de temps d'exécution. L'optimiseur de requêtes a pour rôle d'examiner toutes les alternatives d'exécution possibles afin de choisir la meilleure (le plan d'exécution optimal).

Dans le contexte des systèmes à base d'un médiateur, le médiateur a rarement une vision sur la façon dont sont traitées les sous requêtes au niveau des sources (placement des données, type de stockage, stratégie d'évaluation…)[LEVY 00][DANG 03].

De plus la distribution des données sur des sources disjointes ne permet pas d'utiliser directement les algorithmes employés normalement dans le cas d'un SGBD centralisé.
Le problème consiste comment le médiateur peut-il estimer le coût d'un plan d'exécution sachant que[NAAK 99] :

1- Les opérations du plan sont distribuées sur les wrappers et le médiateur;
2- Le médiateur ne connaît pas l'implémentation des opérateurs traités sur les sources par l'intermédiaire des wrappers;
3- Le degré d'autonomie des sources peut être différent d'une source à l'autre.

Dans la plupart des techniques d'optimisation proposées (parallelisation d'une requête, ajout des vues transitoires et capacité réduite des sources)[TAME 99][NAAK 99]. On trouve que le processus d'optimisation précède l'exécution de la requête. Elles sont de nature statique. Ces techniques reposent sur une estimation des caractéristiques des sources

(modèle de coût). Par conséquent, si les estimations manquent de précision (des environnements hétérogènes et distribués), le plan d'exécution choisi est loin d'être optimal[NAAK 99].

Pour éviter ce problème, des techniques d'optimisation dynamiques, pendant l'exécution du plan, ont été proposées.

3.2.3.1 Optimisation dynamique des requêtes sur de bases de données hétérogènes

Au contraire de l'optimisation statique qui précède l'exécution de la requête, l'optimisation dynamique a lieu pendant l'exécution de la requête. Nous distinguons dans la littérature[NAAK 99] quatre approches : Réécriture dynamique du plan d'exécution, Prise en compte des sources indisponibles, Ordonnancement dynamique des jointures et Optimisation adaptative.

a- Réécriture dynamique du plan d'exécution

L'objectif est de réduire l'impact des sources les plus lentes sur le temps de réponse total de la requête[NAAK 99]. Cette approche [AMSA 96] propose d'éviter les situations de blocage en modifiant dynamiquement le plan d'exécution pour permettre au médiateur d'effectuer du travail utile en attendant la fin du blocage. Le travail utile est :

1- La matérialisation des résultats intermédiaires, et
2- La construction de nouveaux opérateurs pour manipuler les résultats intermédiaires disponibles. Lorsque le travail utile est réalisé, le médiateur attend que les sources bloquées reprennent la production des résultats.

Cette technique fait l'hypothèse que toutes les sources de données répondent dans un délai raisonnable. Mais, certaines sources de données peuvent être temporairement indisponibles, ou fournir leur réponse dans un délai qui n'est pas raisonnable[BONN 99].

b- Le traitement incrémental des requêtes

Le problème est de traiter la requête en tenant compte des sources indisponibles sans attendre que toutes les sources soient disponibles pour débuter le traitement de la requête[NAAK 99]. Cette approche [BONN 99] vise à obtenir efficacement le résultat d'une requête en commençant le traitement avant que toutes les sources soient disponibles. La

requête est traité partiellement et les résultats partiels sont matérialisés dans le médiateur. La requête initiale est réécrite pour prendre en compte les résultats partiels (nommée requête incrémentale). La requête incrémentale est exécutée lorsque les sources indisponibles devient accessibles. Le résultat est équivalent à celui de la requête initiale mais plus rapide.

Un des avantages de cette méthode est de pouvoir traiter une requête bien qu'à aucun instant l'ensemble des sources ne soient disponibles simultanément[NAAK 99].

c- Ordonnancement dynamique des jointures.

L'idée est ici de traiter une jointure inter-site le plutôt possible, dès que deux sous-requêtes ont produit leur résultat. La méthode se déroule en quatre étapes [EVRE 95] [NAAK 99]. En préliminaire à l'exécution, le coût des jointures entre deux sites est estimé au moyen des formules traditionnelles basées sur la sélectivité des jointures. (2) Toutes les sous-requêtes sont envoyées en parallèle aux wrappers, pour être exécutées. (3) Dès qu'une sous-requête a produit son résultat, une valeur de seuil est calculée en fonction du coût des jointures qui sont reliées à cette sous-requête. Parmi toutes les jointures impliquant les résultats des sous-requêtes, celles qui ont un coût supérieur au seuil sont éliminées; la jointure de moindre coût est exécutée. Reprendre l'étape (3) jusqu'à ce que toutes les jointures soient exécutées.

Cette approche a deux avantages: elle remplace la stratégie d'optimisation statique qui détermine un plan d'exécution à partir d'un modèle de coût, et elle prend en considération le temps de réponse réel des sources, et ainsi offre une amélioration du temps de réponse total de la requête[EVRE 95] [NAAK 99].

d- Optimisation adaptative

L'approche suppose que l'exécution d'une requête se déroule selon trois phases[NAAK 99]: (1) la phase de contrôle (monitoring) pendant laquelle l'état d'avancement de l'exécution est mesuré; (2) la phase de prise de décision au cours de laquelle l'optimiseur décide éventuellement de corriger le plan courant s'il n'est plus optimal à cause de la perte de précision des estimations utilisées pour son choix; (3) une phase de correction pendant laquelle le plan courant est abandonné au profit d'un nouveau plan.

Le problème de cette approche est la phase de prise de décision qui est basé sur un modèle de coût.

3.3 Conclusion

Nous avons étudié dans ce chapitre les problèmes du traitement des requêtes sur les systèmes basés médiateur. D'après cette étude nous concluons que l'approche GLAV est la meilleure par rapport aux autres (GAV et LAV). En effet, les deux avantages principaux de cette approche sont :

1- Elle est flexible aux changements des sources de données (comme LAV et au contraire que GAV);

2- Dans la reformulation des requêtes suivant cette approche, chaque règle de mapping est représentée par une requête conjonctive écrite sur le schéma global associée à une requête conjonctive écrite sur des schémas sources. Ces requêtes sont des vues virtuelles ne représentent pas des résultats stockés sur des sources, au contraire que LAV (dans LAV chacune des sources peut être regardée comme elle contient une réponse à une requête écrite sur le schéma global. Par conséquent, les sources représentent des réponses matérialisées aux requêtes écrites sur le schéma global). Ces règles permettent de reformuler la requête d'une manière plus efficace.

Dans notre solution nous avons adapté l'approche GLAV pour définir les règles de mapping mais nous utilisons le langage XQuery (chapitre 2) pour exprimer les règles de mapping. XQuery est plus expressive que celle utilisé dans GLAV.

L'optimisation des requêtes dans un environnement hétérogènes et distribuées est une tâche difficile, à cause l'autonomie des sources. Dans notre travail nous considérons que les sources sont autonomes elles n'exports pas leurs modèles de coûts. De ce fait l'utilisation d'une méthode d'optimisation statique s'apparaît inutile. Nous adoptons d'utiliser une méthode d'optimisation dynamique qui remplace les méthodes d'optimisations statiques (qui utilisent un modèle de coût). Nous adaptons la méthode d'ordonnancement de jointure inter-sites dans la deuxième stratégie de coopération entre les

agents. Cette approche elle prend en considération le temps de réponse réel des sources, et ainsi elle offre une amélioration du temps de réponse total de la requête.

Chapitre 4 : Les systèmes multi agents

4.1 Introduction

L'apparition du paradigme agent donne une nouvelle vue pour le développement des systèmes de nature complexe, hétérogène, distribué et/ou autonome. L'objectif ce chapitre est de donner un aperçu sur les systèmes multi agents, qui est en relation avec notre travail, ensuite nous expliquerons l'utilisation des SMA pour l'intégration d'informations.

4.2 Généralité sur les SMA

Depuis quelques années les systèmes multi agents (SMA) ont pris une place importante dans le domaine de l'informatique en général et dans le domaine de l'intelligence artificielle et des systèmes distribués en particulier. En effet, la programmation orienté objet n'est pas toujours adaptée aux besoins des applications d'aujourd'hui.

Les systèmes sont souvent distribués sur plusieurs machines. Ils interagissent et communiquent entre eux et doivent aussi s'exécuter indépendamment les uns des autres. Ils sont souvent divisés en sous-systèmes (indépendants) qui exécutent chacun une partie du travail dans un but commun. La programmation orientée-agent offre une façon beaucoup plus naturelle d'analyser, de concevoir et d'implémenter ce type de système[GIRA 01].

Les SMA sont des systèmes idéaux pour représenter des problèmes de nature hétérogènes et distribués [JARR 02]. Ils ont l'avantage de faire intervenir des schémas d'interaction sophistiqués. Les SMA sont à l'intersection de plusieurs domaines scientifiques: informatique répartie, génie logiciel, intelligence artificielle et vie artificielle. Ils s'inspirent également d'études issues d'autres disciplines connexes notamment la sociologie, la psychologie sociale, les sciences cognitives et bien d'autres[JARR 02][CHAI 99].

La technologie multi agents offrent de nombreux avantages potentiels et appliquée dans des domaines très variés où ils présentent une alternative intéressante aux approches classiques[CHAI 99]. Parmi ces domaines on trouve : systèmes distribués, interface hommes-machines, bases de données et bases de connaissances distribuées coopératives, systèmes pour la compréhension du langage naturel, protocoles de communication et

réseaux de télécommunications, génie logiciel, robotique cognitive et coopération entre robots, applications distribuées comme le web, l'Internet, le contrôle de trafic routier, le contrôle aérien, etc.
Dans ce qui suit nous donnons la notion d'agent.

4.3 Notion d'agent

Le concept d'agent a été l'objet d'études pour plusieurs décennies dans différentes disciplines. Toutefois, il est possible d'identifier quelques caractéristiques qu'un agent doit posséder pour qu'il puisse être qualifié d'agent[JARR 02].

Dans la littérature, on trouve une multitude de définitions d'agents. Elles se ressemblent toutes, mais diffèrent selon le type d'application pour la quelle est conçu l'agent. L'une des premières définitions de l'agent dûe à Ferber [FERB 95]: *Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi agent, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents.* Les propriétés clés d'un agent de cette définition sont : l'*autonomie*, l'*action*, la *perception* et la *communication.* D'autres propriétés peuvent être attribuées aux agents. En particulier la *réactivité*, la *rationalité*, l'*engagement* et l'*intention*.

Récemment, Jennings, Sycara et Wooldridge ont proposé la définition suivante pour un agent[JARR 02]: « Un agent est un système informatique, *situé* dans un environnement, et qui agit d'une façon *autonome* et *flexible* pour atteindre les objectifs pour lesquels il a été conçu ». Les notions "situé", "autonomie" et "flexible" sont définies comme suit : *Situé,* signifie que l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement. *Autonome,* signifie que l'agent est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne. *Flexible,* signifie que l'agent dans ce cas est :

- *Capable de répondre à temps* : l'agent doit être capable de percevoir son environnement et élaborer une réponse dans les temps requis;
- *Proactif* : l'agent doit exhiber un comportement proactif et opportuniste, tout en étant capable de prendre l'initiative au "bon" moment;

- *Social* : l'agent doit être capable d'interagir avec les autres agents (logiciels et humains) quand la situation l'exige afin de compléter ses tâches ou aider ces agents à accomplir les leurs.

4.4 Les différents types d'agent

La technologie agent est bouillonnante. De très nombreux acteurs interviennent sur le sujet : universités, centres de recherche, sociétés privées et leur classification des agents dépendent de leur utilisation. L'importance de cette classification est qu'elle permet de donner des attributs aux agents. Nous présentons une taxonomie pour les agents qui est plus complète que d'autres. Cette taxonomie est inspirée de celle présentée dans [GRAB 00][NWAN 96] [ANGL ??].

4.4.1 Agents collaboratifs

Les agents collaboratifs sont utilisés dans l'accomplissement des tâches pour résoudre des problèmes. Ces agents exécutent diverses tâches et agissent l'un sur l'autre pour diffuser l'information et manipuler des conflits dans des environnements multi-agents ouverts. Ils peuvent apprendre, mais cet aspect n'est pas typiquement leur opération principal. Pour coordonner leurs activités, ils négocient afin de conclure des accords mutuellement acceptables.

Ce type d'agents permettent de[GRAB 00]:

- Résoudrent des problèmes qui sont trop vaste pour un agent simple centralisé, en raison des limitations de ressource ou le risque d'avoir un système centralisé;
- L'intercommunication et l'interopération des systèmes anciens (legacy system) existants,
- Fournissent des solutions aux problèmes de nature distribués.

4.4.2 Agents d'interface

Les agents d'interface provenant des domaines de l'interface homme-machine et de l'intelligence artificielle qui essayent de simplifier la vie de l'utilisateur en automatisant des tâches réalisées couramment par l'observation des comportements répétitifs ou en surveillant des ressources comme le courrier électronique.

4.4.3 Agents Mobiles

Les agents mobiles provenant du domaine des réseaux dont l'objectif est la réalisation de tâches réparties sur un ensemble de machines interconnectées pour le compte d'un utilisateur ou d'une application. Un agent mobile est un agent capable de se déplacer dans son environnement, dispose donc de dispositifs assurant sa mobilité.

4.4.4 Agents d'Information/Internet

Ces agents sont utiles dans des situations de surcharge d'information. Ils exécutent le rôle de la gestion, manipulation ou rassemblant l'information de plusieurs sources distribuées. Les agents d'information sont capables de filtrer de grandes quantités d'information et de choisir des données appropriées[GRAB 00]. Nous décrivons brièvement le rôle pour chaque type d'agents d'information :

Les Agents de recherche d'information

La quantité d'information disponible restreint la capacité de la plupart des utilisateurs de retrouver l'information utile. Les agents de recherche disposent de la connaissance de diverses sources d'information. Cette connaissance inclut le type des informations disponibles à chaque source, comment accéder à ces données et leur fiabilité.

Les Agents de filtrage d'information

Les agents de filtrage d'information essaient de résoudre le problème de la surcharge d'information en limitant et en triant les informations arrivant à un utilisateur. Les agents de filtrage d'information comportent des mécanismes d'apprentissage. Cela leur permet de s'adapter aux besoins de leur utilisateur.

Les Agents de suivi de l'information

Des tâches sont dépendantes de la notification du changement de certaines données. La veille technologique utilise ce genre d'outils pour suivre les changements sur les sites WEB choisis. Les agents de suivi de l'information permettent de consulter des sources de données diverses et de faire un suivi permanent des changements apportés[ANGL ??].

Les Agents de médiation de sources de données

Les agents peuvent être employés comme des médiateurs entre des sources de données, et fournissant les mécanismes permettant d'interopérer.

4.4.5 Agents réactifs

Un agent est réactif s'il répond de manière opportune aux changements de son environnement. Un tel type d'agent n'a pas besoin d'une représentation symbolique élevée de la perception de son environnement. On trouve de nombreux agents réactifs dans la littérature, mais peu le sont purement. Pour la plupart, la réactivité n'est qu'une caractéristique et n'interdit pas l'action intentionnelle. La structure des agents purement réactifs tend à la simplicité, mais ces derniers peuvent être capables d'actions de groupe complexes et coordonnées. C'est le cas d'une société de fourmis.

4.4.6 Agents hybrides

Les architectures hybrides mélangent des composants réactifs et délibératifs pour produire un modèle plus puissant.

4.4.7 Agents hétérogènes

Les systèmes d'agents hétérogènes référence à une organisation contient aux moins deux ou plus d'agents appartiennent à deux ou plus classes différentes. Un système d'agents hétérogènes peut contenir aussi des agents hybrides. Les différents agents peuvent communiquent entre eux en utilisant un langage de communication d'agent (ACL).

4.4.8 Agents réellement intelligents

Les agents réellement intelligents représentent une forme idéale d'un agent intelligent et autonome avec un modèle interne de raisonnement. Un agent réellement intelligent est un agent, ce qui inclut toutes les formes d'agents mentionnées au-dessus[GRAB 00].

4.5 Les systèmes multi agents

Un système multi agents est un système distribué composé d'un ensemble d'agents. Les SMA sont conçus et implémentés idéalement comme un ensemble d'agents interagissants, le plus souvent, selon des modes de *coopération*, de *concurrence* ou de *coexistence*[CHAI 99]. Selon [JARR 02], un SMA est généralement caractérisé par :

1- Chaque agent a des informations ou des capacités de résolution de problèmes limitées, ainsi chaque agent a un point de vue partiel;
2- Il n'y a aucun contrôle global du système multi agents;
3- Les donnés sont décentralisées;
4- Le calcul est asynchrone.

Les SMA sont des systèmes idéaux pour représenter des problèmes possédant de multiples méthodes de résolution, de multiples perspectives et/ou de multiples résolveurs. Ces systèmes possèdent les avantages traditionnels de la résolution distribuée et concurrente de problèmes comme la modularité, la vitesse, et la fiabilité[CHAI 99]. Ils héritent aussi des bénéfices envisageables de l'Intelligence Artificielle comme le traitement symbolique, la facilité de maintenance, la réutilisation et la portabilité mais surtout, ils ont l'avantage de faire intervenir des schémas d'interaction sophistiqués[JARR 02].
Les types courants d'interaction incluent[CHAI 99] :

- La coopération : travailler ensemble à la résolution d'un but commun,
- La coordination : organiser la résolution d'un problème d'une manière à éviter les interactions nuisibles ou les interactions bénéfiques soient exploitées,
- La négociation : parvenir à un accord acceptable pour toutes les parties concernées.

4.6 La coopération entre agents

La coopération est une caractéristique très importante dans les SMA. Une résolution distribuée d'un problème est le résultat de l'interaction coopérative entre les différents agents. Les méthodes de coopération tels que nous les trouvons dans la littérature [FERB 95][SAID 03] :

- *Le regroupement et la multiplication* : Les agents forment un bloc composé de la totalité des individus, ce qui leur permet d'effectuer des actions impossibles à un agent seul.
- *La communication :* Les agents coopèrent par la communication en définissant des protocoles de communication pour que les agents puissent se comprendre et échanger des informations.
- *La spécialisation :* Elle repose sur le fait que chaque agent se spécialise à un seul type de tâche spécifique.

- *La collaboration par partage de tâches et de ressources :* Il existe plusieurs mécanismes pour distribuer les tâches. On cite les mécanismes d'élection où les tâches sont attribuées à des agents suite à un accord ou un vote, les réseaux contractuels où des tâches sont attribuées aux agents suite à des cycles d'appels d'offres ou de propositions. La planification multi agents attribue aux agents planificateurs la responsabilité de la distribution des tâches et la structure organisationnelle où les agents ont des responsabilités pour des tâches particulières.
- *Coordination d'action :* Coordonner les actions pour assurer le fonctionnement cohérent de système.
- *La résolution de conflit par arbitrage et négociation :* les agents coopèrent efficacement s'ils sont capables de résoudre le conflit et empêcher que des désaccords entre individus ne se transforment en lutte ouvertes et que le système dans son entier ne dégrade ses performances.

4.7 Communication entre agents

Les agents peuvent interagir soit en accomplissant des actions explicites de communication entre eux, soit en accomplissant des actions du type signaux qui modifient leur environnement[JARR 02]. En communiquant les agents peuvent échanger des informations et coordonner leurs activités. Les communications sont à la base des interactions et de l'organisation sociale d'un SMA. Dans les SMA deux stratégies principales ont été utilisées pour supporter la communication « explicite » [CHAI 99] : les agents peuvent échanger des messages directement ou ils peuvent accéder à une base de données partagées (appelée tableau noir ou « blackboard », figure 4.1).

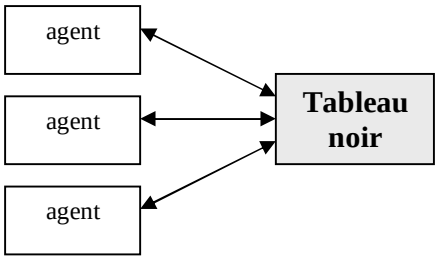


Figure 4.1 : Communication par partage d'information

4.7.1 Communication par échange directe des messages entre agents

Dans la littérature [GRAB 00] on trouve plusieurs stratégies utilisées pour l'échange direct des messages entre agents. Nous décrivons ces stratégies:

- ***Appel de procédure à distant*** (Remote Procedure Call « RPC ») : RPC est l'utilisation des services d'autres systèmes. L'inconvénient majeur du RPC est le couplage forte de la communication et une liaison forte entre la procédure d'appel et l'appelé.
- ***Bus logiciel*** (Object Request Brocker « ORB ») [GEIB 98] : Le concept ORB est l'extension orientée objet du RPC. Dans cette architecture, les services sont offres par les objets. Principalement, il prend en charge la communication par des courtiers (brockers). Un courtier est un composant de système utilisé pour la localisation de service-offre, des service-consommateurs et pour le transport de message. CORBA et DCOM sont considérés comme des exemples de cette stratégie.
- ***Groupe de communication*** : Dans un groupe de communication, un message est envoyé à plus d'un récepteur. Selon le nombre des récepteurs, on distingue : *Multicast,* le message est reçu par un sous ensemble d'agents parmi l'ensemble qui écoute le canal de communication. *Broadcast,* le message est reçu par tous les agents qui écoutent le canal de communication.
- ***Stratégies basées sur la théorie des actes de langage*** : c'est l'approche de communication la plus couramment utilisée. La théorie des actes de langage est une théorie majeure de la philosophie du langage qui est d'un grand intérêt pour l'analyse des communications symboliques dans les SMA[JARR 02]. Cette théorie fait appel aux blocs primaires du langage naturel, appelés actes de langage, dont la signification est claire et la structure bien formée[CHAI 99]. Récemment la théorie des actes de langage a été proposée comme une base pour les communications entre agents[JARR 02]. Les communications basées sur les actes du langage naturel nécessitent, des actes primitifs, des ontologies de services et des protocoles. De ce fait un SMA doit inclut les éléments suivants[PASQ 01]: 1. les actes du langage qui servent de briques de bases de la structure de communications entre agents. 2. l'ontologie des services permettre aux agents de se comprendre et offre une interprétation commune des contenus propositionnels des actes de langage. 3. des

protocoles ou règles de conversation communes pour structurer le dialogue entre les agents : en effet chaque acte de langage est à interpréter via son contexte et plus particulièrement en rapport aux autres actes de langages qui l'entourent temporellement. L'acte de langage prend son sens comme élément d'un contexte et plus spécifiquement d'une conversation. Les ACL (Agent Communication Langage) prennent donc place dans une couche logiquement supérieure à celle des protocoles de transfert (TCP/IP, HTTP, IIOP) et adressent le niveau intentionnel et social des agents[DENI 03]. Les deux ACL les plus connus sont KQML[FINI 92] et FIPA-ACL[FIPA 00].

4.7.2 Le langage KQML

Knowledge Query and Manipulation Language, Ce langage a été proposé dans un premier article en 1992, puis il a été plus complètement spécifié en 1993[YANN 03]. KQML fourni un ensemble d'actes de langage standards et utiles. Le langage est structuré selon trois niveaux (figure 4.2):

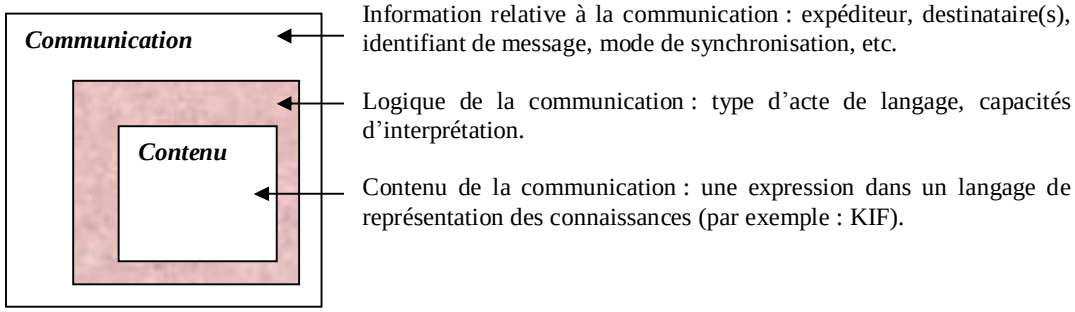


Figure 4.2 : Les trois niveaux d'encapsulation d'un message KQML[YANN 03]

2- La couche de communication : renseigne la communication (identité du récepteur, de l'émetteur et nature de la communication).
3- La couche message : donne des indications sur le contenu du message (langage et ontologie utilisé pour le contenu, type d'acte de langage attaché au contenu). C'est la couche centrale de KQML qui définit le type d'interaction que des agents-KQML pourront avoir.
4- La couche de contenu : contenu du message exprimé en KIF [Knowledge Interchange Format], Prolog, KQML ou autre[CHAI 99]. KQML est muni d'une seule règle de conversation. Elle est simple, même si de nombreuses variantes sont permises. La conversation commence lorsque qu'un agent envoie un message

KQML à un autre et se termine lorsque ce dernier répond. Les actes de langage autorisés par KQML sont restreints aux performatives. Ces performatives sont divisées en trois catégories[CHAI 99][PASQ 01] :

a- 7 performatives de régulation de conversation traitent quelques cas particuliers (sorry, error) et permettent quelques variantes de la règle de conversation (standby, ready, next, rest, discard.) ;

b- 17 performatives de discours permettent l'échange d'informations et de connaissances (ask-if, tell. . .) ;

c- 11 performatives d'assistance et de réseau pour étendre la conversation à plus de deux agents (forward, broker-all. . .).

La principale faiblesse de KQML est que sa sémantique n'est pas fortement formalisée, ce qui provoque des différences d'interprétations. De nouveaux efforts sur la base de ce langage ayant abouti à de nouveaux langages plus fiables, en particulier le FIPA-ACL que nous présentons dans ce qui suit.

4.7.3 Le langage FIPA-ACL

La *Foundation for Intelligent Physical Agents* (FIPA), une autre initiative importante débutée en 1996, est une organisation à but non lucratif regroupant essentiellement des entités commerciales. Elle a pour but d'établir des spécifications standardisées. Parmi ces spécifications, elle propose un langage de communication pour agents (FIPA-ACL)[FIPA 00]. Ce langage est composé de quatre primitives: *confirm*, *disconfirm*, *inform*, *request*. Ces primitives représentent des actions atomiques, c'est-à-dire non-décomposables. Tous les autres actes de langage sont construits sur la base des ces quatre primitives. Si ceux proposés par le standard ne suffisent pas (20 à 30 actes de langages sont proposés), il est possible d'étendre cet ensemble de performatives. FIPA-ACL possède 21 actes de langage[FIPA 00], exprimés par des performative, qui peuvent être groupés selon leur fonctionnalité de la façon suivante :

- Passage d'information: *inform*, *inform-if*, *inform-ref*, *confirm*, *disconfirm*
- Réquisition d'information : *query-if*, *query-ref*, *subscribe*
- Négociation : accept-proposal, *cfp*, *propose*, *reject-proposal*
- Distribution de tâches (ou exécution d'une action) : *request*, *request-when*, *request-whenever*, *agree*, *cancel*, *refuse*

- Manipulation des erreurs : *failure, not-understood*

La sémantique de FIPA-ACL beaucoup plus formalisée que KQML. FIPA-ACL, contrairement à KQML, fournit un certain nombre de protocoles de communication impliquant chacun plusieurs actes de langage[DENI 03]. Par exemple : demande d'action, contract net, …

Un message comprend plusieurs éléments qui sont présentés dans le tableau[FIPA 00].

Elément	Description
Performative	Type de l'acte de langage du message
Sender	L'émetteur du message
Receiver	Le destinataire du message
Reply-to	Participant à l'acte de langage
Content	Contenu du message
Language	Le langage dans lequel le contenu est représenté
Encoding	Décrit le mode d'encodage du contenu du message
Ontology	Le nom de l'ontologie utilisé pour donner un sens aux termes utilisés dans le *content*
Protocol	*Contrôle la conversation*
Conversation-id	Identificateur de la conversation
Reply-with	Identificateur unique du message, en vue d'une référence ultérieure
In-reply-to	Référence à un message auquel l'agent est entrain de répondre (précisé par l'attribut reply-with de l'émetteur)
Reply-by	Impose un délai pour la réponse

4.8 L'utilisation des SMA pour l'intégration d'informations

Plusieurs projets d'intégration d'information utilisent le paradigme agent [PURV 00][BAYA 97] . Ces systèmes sont généralement doivent être capables d'offrir les fonctions suivantes : *La découverte des sources,* trouver la bonne source de données pour l'interroger. *La recherche d'informations,* identifier les informations non structurées et semi-structurées. *Le filtrage des informations,* analyser les données et éliminer celle qui sont inutiles.
La fusion des informations, regrouper les informations d'une manière significative. Parmi les travaux réalisés :

Le projet **Infosleuth** [BAYA 97] a pour but d'implémenter un ensemble d'agents coopératifs qui *découvrent, intègrent et présentent l'information* en fonction des besoins d'un utilisateur ou d'une application pour lesquels ils produisent une interface simple et cohérente. L'architecture du projet Infosleuth est constituée d'un ensemble d'agents collaboratives communiquent par l'ACL KQML. Les utilisateurs expriment ses requêtes sur une ontologie spécifique. KIF et SQL sont utilisés pour représenter les requêtes. Les requêtes sont expédier aux agents spécialisés(agent broker, ontologique, planner,…) pour la recherche des données sur des sources distribuées et pour l'intégration. La résolution de nombreux conflits sémantiques reste guider par l'utilisateur.

Dans ces systèmes malgré la distribution des services, certaines informations doivent être centralisées dans des agents particuliers. Ainsi ils utilisent des agents spécialisés (brokers, planners,…) qui peuvent considérer comme des threads sont plus loin de la définition de l'agent cognitive comme défini dans l'intelligence artificielle distribuée[SERA 98][JOUA 01].

[LARR 99] propose un système d'intégration des données basé médiateur sous une approche multi agents. Ce système constitué des agents : un agent médiateur, agent ontologie, agents ressources, agent d'interface, agent routeur. Chaque agent est considéré comme un thread Java. Les agents communiquent en utilisant l'ACL KQML. La communication entre agents est assurée par un agent spécial « agent routeur ». Le langage pour interroger les sources est SQL et les sources supportées sont relationnelles. Ce système est basé sur un agent médiateur centralisé.

4.9 Conclusion

Dans ce chapitre nous avons expliqué les notions : d'agent, de système multi agents, de coopération et de communication entre agents. Ces notions sont nécessaires afin de comprendre le paradigme agent et de sa grande puissance. Ensuite nous avons étudié l'utilisation des SMAs pour l'intégration d'information. En effet les systèmes multi agents sont actuellement très utilisés, particulièrement pour les applications complexes exigeant l'interaction entre plusieurs entités.

Dans le chapitre suivant nous proposerons notre solution pour résoudre le problème de l'intégration de données hétérogènes et distribuées, est une architecture multi agents

décentralisée qui évite les points faibles des architectures centralisées. Cette architecture combine l'approche de type médiation suivant la première méthode (section 1.5.3) et l'utilisation du paradigme agent, pour assurer la transparence (la transparence à la localisation, aux schémas sources et aux langages de requêtes des données sources) et prendre en compte les conflits sémantiques et aussi assure un traitement efficace des requêtes en utilisant la coopération multi agents.
Nous motivons le choix d'utiliser le paradigme agent par :

1- Les SMA représentent une nouvelle façon d'analyser, de concevoir et d'implémenter des systèmes informatiques complexes, ainsi sont des systèmes idéaux pour représenter des problèmes de nature hétérogènes et distribués;
2- Ils ont l'avantage de faire intervenir des schémas d'interaction sophistiqués;
3- Le problème de l'évaluation des requêtes peut être considéré comme un problème de partage de tâche;
4- Le rendement global du système est amélioré en utilisant la coopération entre les agents;

Nous avons adopté l'ACL de la FIPA comme un langage de communication entre agents. FIPA-ACL présente sur KQML l'avantage d'une sémantique plus rigoureuse, ainsi qu'un effort important de standardisation et permet aussi d'expliciter les conversations.

Chapitre 5 : Le système proposé

« Approche multi agents pour l'intégration et l'interrogation des sources de données hétérogènes et distribuées »

5.1 Introduction

L'intérêt de faire collaborer des systèmes ne cesse de croître. L'évolution constante en matière de réseaux et de bases de données poussent à la réalisation de systèmes d'intégration de données de grande envergure. Ces systèmes fournissent aux utilisateurs une vue uniforme sur un ensemble de sources de données hétérogènes, autonomes et distribuées.

Nous avons montré dans les chapitres précédents que le problème d'intégration de données hétérogènes, autonomes et distribuées est confronté à plusieurs obstacles dûs aux conflits de données et à la complexité du processus d'intégration et de traitement des requêtes d'utilisateurs.

En effet, les solutions doivent assurer : Un processeur de requêtes très puissant s'adapte aux changements d'environnement, la gestion des conflits de données (les conflits syntaxiques, schématiques et sémantiques), l'autonomie des systèmes, l'extensibilité de l'architecture pour maîtriser l'ajout et le retrait de sources de données, la scalabilité du système face à l'évolution du nombre de participants, et la transparence d'accès à la localisation et au format des données.

Comme nous l'avons évoqué dans les chapitres précédents, différentes solutions (ou approches) ont été proposées et que chaque solution fournit quelques avantages par rapport aux d'autres. En plus, nous avons expliqué quelques caractéristiques de notre solution. Dans ce chapitre nous décrirons le système proposé. Pour le présenter, ce chapitre sera divisé en deux parties : la première en donnera un aperçu général et la seconde en sera une présentation détaillée.

5.2 Partie I : Architecture générale

5.2.1 Caractéristiques de la solution

Notre solution, pour résoudre le problème de l'intégration de données hétérogènes et distribuées, est une architecture multi agents décentralisée qui évite les points faibles des architectures centralisées. Cette architecture permet d'offrir aux utilisateurs les moyens de manipuler de manière transparente des données issues d'un ensemble de sources de données hétérogènes, autonomes et distribuées. Elle combine entre l'approche de type médiation et l'utilisation du paradigme agent. En résumé, notre solution est caractérisée par :

1- Architecture multi agents décentralisée qui évite les points faibles des architectures centralisées. Offre une transparence d'accès à la localisation, aux schémas sources et aux langages de requêtes des données sources. Nous avons défini des stratégies de coopération entre agents pour l'efficacité du traitement des requêtes et l'amélioration du rendement global du système et assurer la scalabilité du système. Chaque agent décide son comportement pour satisfaire les besoins définis au niveau global;

2- Offre une intégration sémantique des données en utilisant les nouvelles technologies pour la représentation de la sémantique des informations;

3- Les données hétérogènes supportées sont les données relationnelles, les objets et XML;

4- Utilise le langage d'interrogation expressive XQuery comme langage d'interrogation commun et le modèle XML Schéma comme modèle de données commun;

5- Approche d'intégration descendante et une adaptation des règles de mapping GLAV offrent une flexibilité aux changements;

6- Traitement des sous requêtes destinées aux sources de données permettant de supporter des systèmes anciens (legacy system);

7- Communication de haut niveau entre les agents en utilisant le FIPA-ACL, dans lequel les agents communiquent en terme de transfert de connaissances plutôt qu'en terme de transfert de données.

5.2.2 Vue générale de l'architecture

Nous présentons une architecture multi agents pour résoudre le problème de l'intégration de données hétérogènes et distribuées en prenant en compte les conflits des données (syntaxiques, schématiques, sémantiques) dans l'intégration. Cette architecture permet d'offrir à des utilisateurs les moyens de manipuler de manière transparente des données issues d'un ensemble de sources de données hétérogènes, autonomes et distribuées. Aussi elle offre une optimisation des requêtes en utilisant la coopération entre les agents. Cette architecture est décrite dans [BENH 05]. Les sources de données hétérogènes supportées par notre solution sont des données Relationnelles, Objets et XML.

Nous nous inspirons l'aspect de décentralisation de l'architecture décentralisée générale (faiblement couplé) défini dans [TURK 97]. Dans cette architecture les auteurs n'ont pas donné des informations sur l'architecture interne d'un agent, ni sur le processus d'intégration ou le traitement des requêtes.

L'architecture générale de notre solution est présentée dans la figure (figure 5.1).

Dans cette solution chaque source de données locale est représentée par un agent. Un agent joue le rôle d'un médiateur ou d'un wrapper selon ses besoins pour accomplir ses buts. L'ensemble des agents constitue un système multi agents. Les agents coopèrent entre eux pour exécuter efficacement des requêtes.

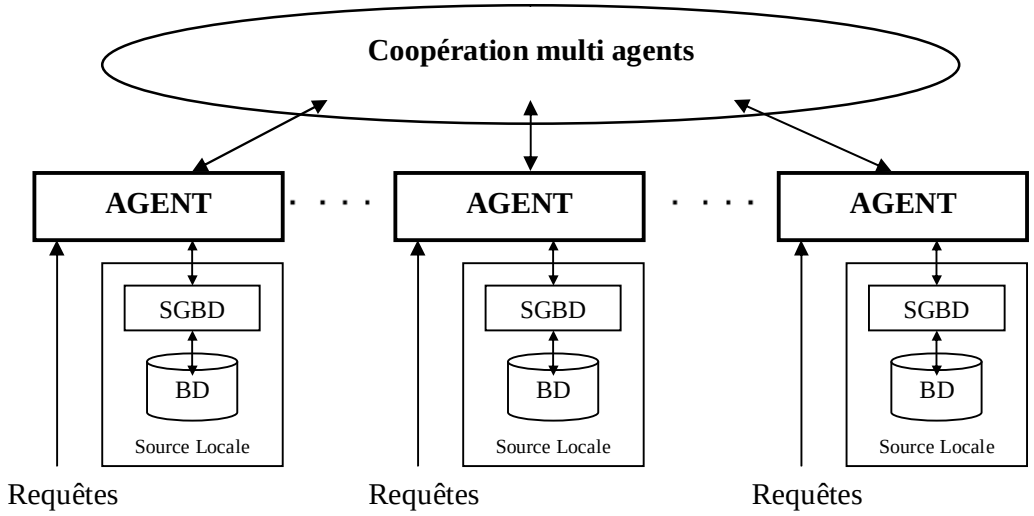


Figure 5.1 : l'architecture générale de notre solution

Généralement lorsqu'un agent reçoit un message de l'environnement, il l'analyse et interagit avec l'environnement. Si le message est une requête d'un utilisateur, l'agent décide à partir de ses connaissances la meilleure stratégie de coopération à adopter pour exécuter

efficacement cette requête. Durant l'exécution d'une requête selon une stratégie de coopération, les agents éliminent les deux problèmes : des conflits d'hétérogénéité sémantiques, schématiques, syntaxiques des données et le problème de distribution de données.

La figure 5.2 représente l'architecture interne de notre agents.

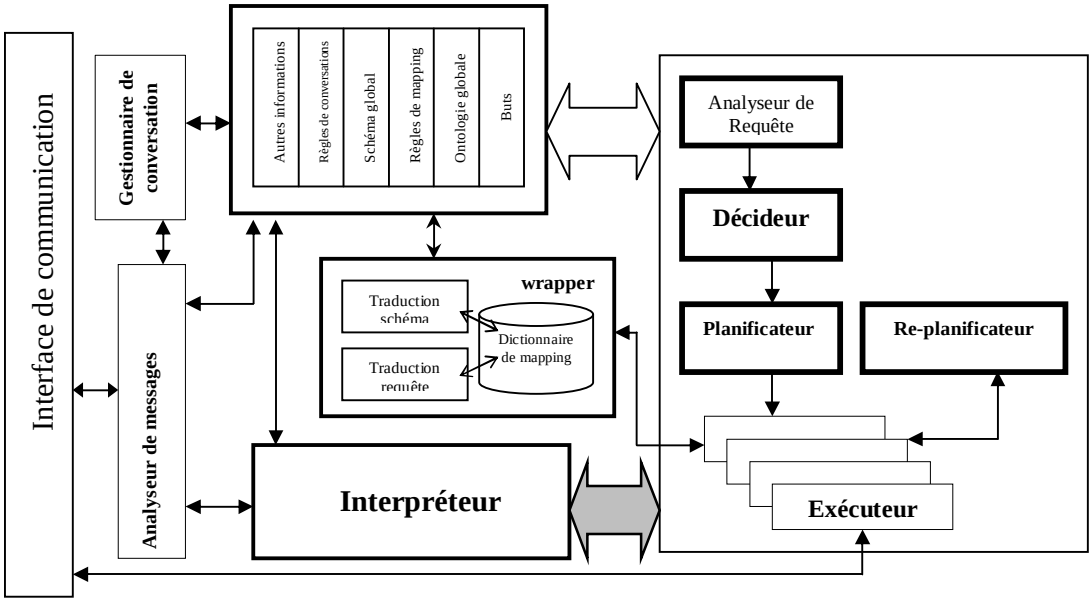


Figure 5.2 : L'architecture interne d'un agent de notre solution

Chaque agent possède:

1- Des informations pour résoudre les conflits syntaxiques, schématiques, sémantiques des données ;
2- Des informations pour exécuter efficacement les requêtes d'utilisateurs.

5.2.2.1 Description des composants

Interface de communication : permet à un agent de communiquer avec l'environnement : Il envoie des requêtes à la source locale qui lui est associée et reçoit des réponses, répond à des requêtes des utilisateurs. Coopère avec les autres agents du système multi agent.

Analyseur de messages : analyse les messages reçus, et détermine leurs types.

Gestionnaire de conversations : s'assure de la bonne compréhension du message en vérifiant que le contexte du message est connu. Il identifie le protocole (stratégie) de coopération utilisé pour le dialogue entre l'émetteur et le récepteur et il contrôle l'état d'avancement de la conversation.

Interpréteur : fait la gestion de synchronisation entre les différents threads. Ainsi selon le type du message reçu, il active et appel le composant nécessaire pour le traitement des requêtes suivant les différentes stratégies appliquées.

Analyseur de requêtes : vérifier la validité d'une requête.

Décideur : décide la meilleure stratégie de coopération qu'il faut appliquer pour traiter efficacement une requête.

Planificateur : son rôle principal est la décomposition d'une requête en des sous-requêtes.

Replanificateur : à partir des informations reçus et des sous requêtes, il génère des nouveaux messages pour optimiser l'exécution d'une requête.

Exécuteur : le rôle d'un exécuteur est multiple selon la mission pour laquelle il est crée. Parmi ces rôles: Un Exécuteur est crée par le Planificateur pour exécuter une requête de recomposition des résultats ou contrôler l'exécution d'une sous-requête de son agent. Un Exécuteur peut aussi être crée par l'Interpréteur afin d'exécuter une sous requête d'un autre agent ou pour décider d'accepter ou non l'exécution d'une requête d'un autre agent. Comme il peut être crée par le Planificateur pour évaluer les capacités des agents qui acceptent l'exécution d'une requête.

Wrapper : joue principalement trois rôles :

1- La traduction des sous requêtes écrites en utilisant le langage commun vers le langage propriétaire à la source locale;
2- La traduction du schéma de la source locale vers un schéma écrit en utilisant le modèle de données commun (XML Schema);
3- La traduction des résultats obtenus de la source locale vers le format commun (XML).

5.2.2.2 L'ajout et le retrait d'un agent

Chaque agent voulant intégrer ou quitter le système multi agent doit informer *l'agent administrateur* qui joue le rôle de la perception de l'environnement et informe les autres agents aux changements de l'environnement (l'explication en section 5.3.1.2).

5.2.2.3 La coopération entre agents

Le mécanisme de coopération entre agents ne suive pas une stratégie fixe mais il est en fonction de la requête traitée, l'état interne de l'agent et de l'environnement. Chaque agent décide son comportement pour satisfaire le but global (amélioration du rendement global du système). Les stratégies de coopération adoptées sont expliquées en section 5.3.2.1

Au niveau de la communication, les agents communiquent en utilisant le langage FIPA-ACL. Ils communiquent en terme de transfert de connaissance. Les agents échangent d'une manière implicite des informations sur l'environnement afin d'améliorer le rendement du SMA (elles aident l'agent de choisir la stratégie la plus efficace) et évitent les interactions nuisibles (expliquée en section 5.3.3.1).

5.3 Partie II : Architecture détaillée

Dans cette section, nous décrivons les caractéristiques intéressantes de notre système : la modélisation des informations, le traitement des requêtes, les composants décideur, planificateur, replanificateur, exécuteur et wrapper. Et la communication entre les agents.

5.3.1 Intégration de données hétérogènes

5.3.1.1 Modélisation des informations

i) Modèle de données commun

L'utilisation d'un modèle de données commun facilite la compréhension du schéma des données hétérogènes et assure un accès uniforme à l'ensemble des données distribuées. Il permet d'éliminer les conflits syntaxiques. Comme le montre le chapitre 2, nous avons adopté le modèle XML Schéma comme modèle de données commun pour éliminer les conflits syntaxiques et le langage XQuery comme langage commun d'interrogation.

ii) Représentation de la sémantique des informations

La sémantique entre dans deux niveaux :

1. La construction du système d'intégration de données, et
2. L'interrogation de ce système.

Les informations sémantiques doivent permettre de comprendre sans ambiguïtés la signification des informations partagées et échangées. Dans notre solution, nous définissons trois niveaux de connaissance à savoir: Le domaine général, le contexte source et les règles de mapping.

A) Domaine général

C'est le domaine général dans lequel les données trouvent une signification. Nous utilisons un schéma global décrit par le modèle de données commun XML Schéma auquel on associe une ontologie globale représentée par la logique de description pour expliciter les concepts du domaine général.

A-1) Le schéma global

Le schéma global est construit suivant l'approche descendante (Top down), c'est à dire à partir des buts globaux (besoins communs) on construit un schéma global. Ce schéma est réalisé par des spécialistes dans le domaine d'application de l'intégration des données. Il est écrit en utilisant le modèle commun XML Schéma.

En effet, les approches basées sur un schéma global sont considérées comme des approches fortement fédérées (chapitre 1), mais l'approche descendante est considérée moins fortement fédéré que l'approche ascendante (Bottom up)[BUSS 99].

A-2) Ontologie Globale

Cette ontologie est utilisée pour expliciter les concepts du domaine général et ainsi pour faciliter l'intégration des sources de données. Cette ontologie est un thesaurus qui décrit tous les concepts du schéma global. Cette ontologie permet de préciser d'une manière claire et explicite tous les concepts du schéma global, afin de faciliter l'intégration des données. La construction de cette ontologie en utilisant la logique de description est réalisée par un spécialiste du domaine d'application d'intégration de données.

B) Contexte source

Chaque source locale (un agent fait l'abstraction d'une source de données) dans notre solution est caractérisée par :

1. Son schéma décrit en modèle de données commun XML Schéma;
2. Une ontologie locale représentée par la logique de description explicite les concepts du domaine local;
3. Une bibliothèque de fonctions permettant de résoudre quelques types des conflits sémantiques (conflits sémantiques de valeurs) et des conflits schématiques (cette idée est inspirée de celle présentée dans [JOUA 01]).

B-1) Le passage vers XML schéma

Les données supportées par notre solutions sont de types relationnels, objets et XML. Pour cela nous définissons des règles du passage d'un schéma relationnel vers XML schéma et d'un schéma objet vers XML schéma.

a- Passage d'un schéma relationnel vers XML schéma

Ce passage d'un schéma relationnel vers XML schéma prend en compte : les contraintes d'unicité des clés primaires et les contraintes liées aux clés étrangères. XML schéma peut supporter facilement un schéma relationnel[CHAZ 01]. Nous nous inspirons quelques étapes du passage de celle présentées dans[BOUZ 03]. Les étapes du passage sont:

1- Chaque table est représenté par un élément de type ComplextType,
2- Les attributs d'une table (sauf la clé primaire) sont représentés par des sous éléments de type SimpleType,
3- L'attribut clé primaire est représenté par un attribut dans XML schéma,
4- On ajoute les contraintes d'unicité et référentielles.

Notre algorithme de passage est présenté dans la figure (Algorithme :Passage de relationnel vers XMLSchema).

On note que la position de la déclaration *key* donne le contexte à partir duquel le contrôle sera effectué. Une clé primaire sera unique et doit être non nulle à l'intérieur de sa table.

Les deux chemins XPath spécifiés dans la déclaration *Key* seront évalués par rapport à ce noeud contexte. Le premier est défini dans l'élément *selector* et, comme son nom l'indique, il sélectionne l'élément qui doit être unique.

Le deuxième chemin XPath est évalué par rapport au noeud sélectionné et spécifie le noeud identifiant l'attribut NomDeLaCléPrimaire qui doit être unique. C'est ce noeud dont l'unicité va être testée.

La déclaration, *Keyref* définit une référence à une clé primaire. Nous allons ainsi pouvoir tester qu'une clé étrangère correspond à une clé primaire d'une table.

Exemple :

L'exemple suivant montre le passage d'un schéma relationnel vers un schéma XML. Le schéma relationnel représente une base de données « employés et départements »

Algorithme :Passage de relationnel vers XMLSchema

Entrée : un schéma relationnel(ensemble des tableaux)

Sortie : document XML Schéma

Début

- Initialiser un document XML Schéma : créer l'élément schéma,
  <schema xmlns=http://www.w3.org/2000/10/XMLSchema >
- Créer l'élément racine et déclarer tous ses sous éléments nécessaires,

```
<element name="racine"  type ="racineType" />
< complexType   name= "racineType" >
  <sequence>
    <element name="NomDeLaTable-1"  type ="NonDeLaTableType-1" minOccurs="1"
maxOccurs="unbounded" />
    <element name="NomDeLaTable-2"  type ="NonDeLaTableType-2" minOccurs="1"
maxOccurs="unbounded"/>
          …….
          …….
   <element name="NomDeLaTable-N"  type ="NonDeLaTableType-N" minOccurs="1"
maxOccurs="unbounded"/>
  </sequence>
< /complexType >
```

- Pour chaque table du schéma relationnel alors : ajouter au document la définition de la table et ses attributs : la déclaration d'une table et ses attributs suivent la syntaxe :

```
- Ajouter :      < complexType   name= "NonDeLaTableType" >
                  <sequence>
- Pour tous les attributs i sauf la clé primaire alors :
      Ajouter : <element  name= "attr-i"   type= "attr-i-Type" />
- Ajouter :     </sequence>
< !…  ajouter la clé primaire de la table……/>
- Ajouter       <attribute name = "NomDeLaCléPrimaire" Type ="typeDeCléPrimaire" />
- Ajouter        </complexType>

< !…  ajouter la contrainte d'unicité de la clé primaire de la table……/>
- Ajouter:       <Key name = "Key"+"NomDeLaCléPrimaire">
                  <selector xpath= "NomDeLaRacine/NomDeLaTable"/>
                 <field xpath ="@NomDeLaCléPrimaire"/>
                 </Key>

< !…  ajouter les contraintes référentielles des clés étrangères de la table……/>
- Pour toute clé  étrangère j alors :
    Ajouter :
          <Keyref name = "Key"+"NomDeLaCléEtrangère-j" refer=
               "Key"+"NomDeLaCléEtrangère-j-DansSaTable " >
          <selector xpath= "NomDeLaRacine/NomDeLaTable"/>
          <field xpath ="NomDeLaCléEtrangère-j"/>
          </Keyref>
```

- Ajouter : </schema>

Dept(DeptClé, Dnom, Budget) ;

Emp(EmpClé, Enom, DeptCléEtr, Salarie) ; tel que :

DeptClé, EmpClé, DeptCléEtr, Budget et Salaire sont de type positiveInteger ,

Dnom, Enom sont de type String,

Nous obtenons le document XML schema en appliquant l'algorithme précédent :

```
<schema xmlns=http://www.w3.org/2000/10/XMLSchema >
<element name="DBR"  type ="DBRType"/ >
< complexType   name= "DBRType" >
    <sequence>
        <element  name= "Dept"   type= "DeptType" minOccurs="1" maxOccurs="unbounded" />
        <element  name= "Emp"   type= "EmpType" minOccurs="1" maxOccurs="unbounded" />
    </sequence>
</complexType>

< complexType   name= "DeptType" >
    <sequence>
        <element  name= "Dnom"   type= "string" />
        <element  name= "Budget"   type= "positiveInteger" />
    </sequence>
   <attribute name = "DeptClé" type ="positiveInteger" />
</complexType>
<Key name = "KeyDeptClé">
  <selector xpath= "BDR/Dept"/>
 <field xpath ="@DeptClé"/>
 </Key>
 < complexType   name= "EmpType" >
    <sequence>
        <element  name= "Enom"   type= "string" />
        <element  name= "Salaire"   type= "positiveInteger" />
        <element  name= "DeptCléEtr"   type= "positiveInteger" />
    </sequence>
   <attribute name = "EmpClé" Type ="positiveInteger" />
</complexType>
<Key name = "KeyEmpClé">
  <selector xpath= "BDR/Emp"/>
 <field xpath ="@EmpClé"/>
 </Key>
<Keyref name = "KeyDeptCléEtr" refer= "KeyDeptClé" >
<selector xpath= "DBR/Emp"/>
<field xpath ="DeptCléEtr"/>
</Keyref>
</schema>
```

b- Passage d'un schéma objet vers XML schéma

Le schéma d'une base de données objets est un ensemble de classes[FLOR 95]. Pour permettre le passage d'un schéma objet vers XML schéma nous ne prenons en compte que l'aspect statique des données. Nous nous inspirons du travail[BOUR 01] qui fait le passage d'un schéma objet vers un schéma relationnel et on lui ajoute les concepts du modèle XML schéma qu'il ne possède pas (par exemple le concept d'héritage est supporté dans XML schema et n'est pas supporté dans le modèle relationnel). Un schéma objet est caractérisé par[BOUR 01] :

1- la représentation d'une classe,
2- l'héritage entre les classes,
3- attribut monovalué de type simple,
4- attribut multivalué de type simple,
5- attribut monovalué de type classe,
6- attribut multivalué de type classe.

Ce passage d'un schéma objet vers XML schéma prend en compte : les contraintes d'unicité des clés primaires et les contraintes référentielles liées aux clés étrangères. Les règles du passage sont :

1- Initialiser un document XML schéma et créer l'élément schema et l'élément racine du schéma,
2- Chaque classe est représentée par un element de type complexType,
3- Le concept d'héritage est représenté par la définition de type dérivé par extension en utilisant : <complexContent> et <extension base="superclasse">,
4- Les attributs « attribut monovalué de type simple » sont représentés par des sous elements de type simple. Pour la clé primaire est représentée par un attribut,
5- Les attributs multivalués de type simple sont représentés par la structure adéquate utilisée dans XML schéma,
6- Les attributs monovalués de type classe, sont représentés par des clés étrangères,
7- Les attributs multivalués de type classe sont représentés par la structure adéquate utilisée dans XML schéma et utilisant des clés étrangères.

Exemple :

Considérant le schéma objet ci-dessous: nous prenons des noms symboliques pour représenter plusieurs concepts :

```
Classe A {
        Key intpositive Ka
        int x ;
        E e ; // attribut de type classe
        }

Classe B : A { // la classe B hérite de la classe A
                int y ;
                Set D d ; // attribut multivalué de type classe
            }

Classe C : A { // la classe B hérite de la classe A
              string s ;
              Set int n; // attribut multivalué de type simple
             }
Classe D {
        Key intpositive Kd
        string ss ;
        int t;
        }
Classe E {
        Key intpositive Ke
        Int z ;
        String se;
        }
```

Le schema XML correspondant au schéma objet en utilisant les règles précédentes est :

```
<schema xmlns=http://www.w3.org/2000/10/XMLSchema >

<element name="DBO"  type ="DBOType"/ >
< complexType   name= "DBOType" >
    <sequence>
        <element  name= "A"   type= "AType" minOccurs="1" maxOccurs="unbounded"/>
        <element  name= "B"   type= "BType" minOccurs="1" maxOccurs="unbounded"/>
```

```
            <element  name= "C"   type= "CType" minOccurs="1" maxOccurs="unbounded"/>
            <element  name= "D"   type= "DType" minOccurs="1" maxOccurs="unbounded"/>
            <element  name= "E"   type= "EType" minOccurs="1" maxOccurs="unbounded"/>
      </sequence>
</complexType>

<!…….la définition du type Atype…….>
< complexType   name= "AType" >
      <sequence>
            <element  name= "x"   type= "integer" />
            <element  name= "e"   type= "positiveInteger" />
     </sequence>
    <!……clé primaire……>
    <attribute name = "Ka"  type="integer" />
</complexType>
<!…….la contrainte d'unicité de la clé primaire Ka…….>
<Key name = "KeyKa">
    <selector xpath= "DBO/A"/>
    <field xpath ="@Ka"/>
 </Key>
<!…… la contrainte référentielle sur la clé étrangère e …….  >
<Keyref name = "RefKe" refer= "KeyKe" >
    <selector xpath= "DBO/A"/>
    <field xpath ="e"/>
</Keyref>

<!…….la définition du type Btype…….>
< complexType   name= "BType" >
<complexContent>
<extension base= AType>
      <sequence>
            <element  name= "y"   type= "integer" />
            <element  name= "d"   type= "positiveInteger" maxOccurs ="unbounded" />
     </sequence>
</extension>
</complexContent>
</complexType>
<!…… la contrainte référentielle sur la clé étrangère d …….  >
<Keyref name = "RefKd" refer= "KeyKd" >
    <selector xpath= "DBO/B"/>
```

```
   <field xpath ="d"/>
</Keyref>

<!…….la définition du type Ctype…….>
< complexType   name= "CType" >
<complexContent>
<extension base= AType>
     <sequence>
          <element  name= "s"   type= "string" />
          <element  name= "n"   type= "integer" maxOccurs ="unbounded" />
    </sequence>
</extension>
</complexContent>
</complexType>
<!…….la définition du type Dtype…….>
< complexType   name= "DType" >
     <sequence>
          <element  name= "ss"   type= "string" />
          <element  name= "t"   type= "integer" />
    </sequence>
   <!……clé primaire……>
   <attribute name = "Kd"  type="positiveInteger" />
</complexType>
<!…….la contrainte d'unicité de la clé primaire Kd…….>
<Key name = "KeyKd">
   <selector xpath= "DBO/D"/>
   <field xpath ="@Kd"/>
 </Key>
<!…….la définition du type Etype…….>
< complexType   name= "EType" >
     <sequence>
          <element  name= "z"   type= "integer" />
          <element  name= "se"   type= "string" />
    </sequence>
   <!……clé primaire……>
   <attribute name = "Ke"  type="positiveInteger" />
</complexType>
<!…….la contrainte d'unicité de la clé primaire Ke…….>
<Key name = "KeyKe">
   <selector xpath= "DBO/E"/>
```

```
   <field xpath ="@Ke"/>
 </Key>
</schéma>
```

B-2) L'ontologie locale

La construction d'une ontologie locale explicite les concepts du schéma local. Dans notre solution, une ontologie locale est utilisée pour trois objectifs :

1- Simplifier l'intégration et identifier les points communs entre le schéma global et un schéma local afin de définir des règles de mapping GLAV.

2- Gérer les conflits sémantiques de valeurs: Permet d'éliminer les conflits sémantiques de valeurs dans le traitement des sous requêtes destinées à la source locale. Plusieurs travaux traitant la gestion de la sémantique n'ont pas pris en compte les conflits sémantiques de valeurs.

3- Permettre d'évaluer le taux de présence des données au niveau de la source locale pour une requête donnée (ce point est détaillé en section 5.3.2.1, définitions).

Cette ontologie est un thesaurus contenant les concepts du domaine local (schéma local) et ses synonymes, homonymes, polysémies, hypernymes et hyponymes (pour assurer l'objectif 1). Et ainsi on associe à chaque attribut du schéma local son type local et sa correspond au niveau global (schéma global) et une référence à une fonction de transformation de type de la bibliothèque de fonctions (s'il est nécessaire c-à-d, il existe des conflits sémantiques de valeurs : objectif 2).

B-3) Bibliothèque de fonctions

Utilisée pour résoudre :

- quelques conflits sémantiques : des fonctions font la conversion d'un type de données à l'autre afin d'éliminer des conflits sémantiques de valeurs;
- quelques types des conflits schématiques (même valeur est représentée par des types différents).

C) Règles de mapping

Après l'identification des points communs. On définit des règles de mapping pour faire la correspondance entre le schéma global et le(s) schéma(s) source(s). Le rôle de ces règles est :

1. Faite la liaison entre le schéma global virtuel et les schémas sources qui représentent des données physiques stockées sur des sources distribuées.
2. Intervenir dans la reformulation des requêtes des utilisateurs.
3. Masquer ou éliminer la majorité des conflits d'hétérogénéité schématiques et sémantiques.

Les règles de mapping de notre solution suivent l'approche GLAV, mais nous utilisons le langage XQuery pour les définir. Les règles sont de la forme :

$r_i : q_g \rightarrow q_s$.

où :

q_g : est une requête XQuery portant sur des éléments du schéma global et

q_s : est une requête XQuery portant sur des éléments des schémas sources (les schémas sources sont écrits en utilisant le modèle commun XML schéma). L'ensemble de toutes les règles de mapping est noté: $M=\{r_1,r_2,...,r_i,...,r_n\}$.

Exemple

Par exemple on a l'attribut *date* qui peut être représentée au niveau global par une chaîne de caractère et au niveau local par trois attributs entiers sont jour, mois et année. Ce conflit schématique est résolue au niveau des règles de mapping GLAV et en utilisant des fonctions de transformations. Soit la règle GLAV qui lie le schéma global à un schéma local d'un agent :

q_g	q_s
for $x in collection("SchemaGlobal")/ ElementGlobal where $x/date=$a return $x	for $x in collection("SchemaLocal de A")/ ElementLocal where $x/jour=JOUR($a) and $x/mois=MOIS($a) and $x/année =ANNEE($a) return $x

Donc, parmi les fonctions de la bibliothèques de fonctions de l'agent A, on trouve les trois fonctions : JOUR, MOIS et ANNEE.

5.3.1.2 L'architecture multi agent

La base de connaissance d'un agent est constituée principalement de :

1. Un schéma global, une ontologie globale et des règles de mapping GLAV : ces informations permettant à un agent de jouer le rôle d'un médiateur ;

2. Un schéma local écrit en modèle commun XML schema, une ontologie locale, une bibliothèque de fonctions de transformations des types et une table contenant des requêtes écrites en XQuery sur un schéma local (écrit en modèle commun) correspondant à des requêtes écrites sur le langage d'interrogation local: ces informations permettant à un agent de faire le rôle d'un wrapper;

3. Une table d'accointance dynamique sur les localisations des autres agents;

4. La croyance sur l'état du chargement des autres agents et aussi sur le coût de la communication. Ces valeurs sont mises à jour durant la communication entre les agents.

Pour que notre système soit flexible aux changements de l'environnement, nous utilisons un agent administrateur. Chaque source de données (représentée par un agent) voulant intégrer ou quitter le système multi agents, doit informer l'agent administrateur. Le rôle de l'agent administrateur est :

1- Lorsqu'un agent veut intégrer le SMA, il envoie sa localisation et ses règles de mapping (les règles lie le schéma global à son schéma local) à l'agent administrateur. Ce dernier envoie un message à touts les agents pour mettre à jours leurs connaissances et ajoutent les nouvelles règles de mapping. Chaque agent recevant le message de l'agent administrateur compare les nouvelles règles à ses règles de mapping GLAV pour trouver d'éventuelles similitudes entre les q_g afin de construire des unions de règles pour reformuler les requêtes d'une manière plus efficace.
2- Lorsqu'un agent veut quitter le SMA, il informe l'agent administrateur. Ce dernier envoie un message à touts les agents pour éliminer tout ce qui est lié à cet agent.

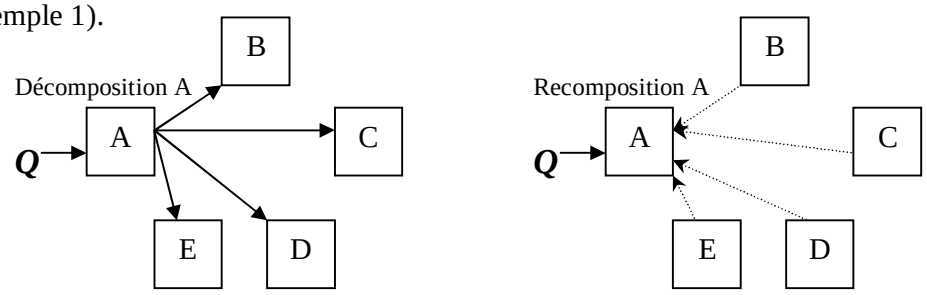
3- Lorsqu'un agent change son schéma local, il résulte de nouvelles règles de mapping. Cet agent informe l'agent administrateur de ses nouvelles règles. Ce dernier envoie un message à touts les agents pour éliminer les anciennes règles de mapping liées à cet agent et les remplace par les nouvelles règles de mapping.

5.3.2 Traitement des requêtes

5.3.2.1 Les stratégies de coopération

Lorsqu'un agent reçoit un message de l'environnement, il l'analyse, puis détermine son type. Dans le cas où le message est une requête d'un utilisateur. La requête est placée dans la liste des buts à accomplir. La requête est analysée pour valider sa correction, puis le composant Décideur de l'agent choisit la bonne stratégie de coopération. On utilise plusieurs stratégies de coopération pour l'exécution des requêtes, l'agent utilise ses connaissances pour en choisir une:

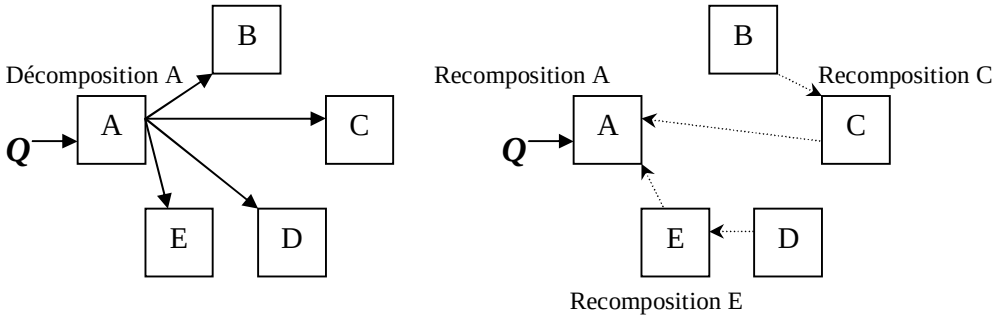
Stratégie 1 La requête est décomposée en sous-requêtes par le composant Planificateur de l'agent. Dans cette décomposition l'agent détermine les agents participant à l'exécution de la requête. Ensuite il envoie à chacun un message (ou des messages) contenant sa sous requête (ou ses sous requêtes). Et enfin un Exécuteur de l'agent recompose les résultats (exemple 1).



Exemple 1 : Décomposition et recomposition par l'agent A.

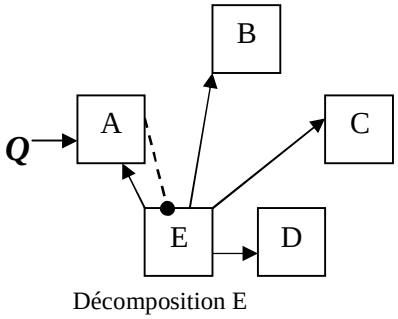
Stratégie 2 La requête est décomposée en sous-requêtes par le composant Planificateur de l'agent puis les envoie aux agents participant à l'exécution de cette requête. Le planificateur crée un Exécuteur pour attendre des informations sur les résultats de ces sous requêtes. Le Replanificateur analyse ces informations et regénère de nouveaux messages destinés aux agents participant à l'exécution de la requête. Ces messages contiennent des ordres de redirections des résultats vers d'autres agents ou des sous requêtes de recompositions des

résultats (exemple 2). Nous nous inspirons de la stratégie d'optimisation concernant l'ordonnancement des jointures inter-site[EVRE 95]. L'idée est de traiter une requête inter-site le plus tôt possible, dès que deux sous-requêtes ont produit leur résultat. Cette approche a deux avantages: elle remplace la stratégie d'optimisation statique qui détermine un plan d'exécution à partir d'un modèle de coût, et elle prend en considération le temps de réponse réel des sources.



Exemple 2 : Décomposition par l'agent A. des sous recompositions par l'agent C et E et une recomposition par A

Stratégie 3 : L'agent cherche un autre agent qui exécute la requête, puis envoie les résultats à l'agent émetteur. Cette stratégie ressemble au protocole Contract net. L'agent envoie un appel d'offres décrivant la requête. Les agents recevant l'appel fournissent leurs compétences pour exécuter la requête. Une fois que l'agent émetteur reçoit des réponses de tous les agents, il évalue ces capacités et fait son choix sur un agent pouvant exécuter efficacement la requête (l'agent ayant la plus grande capacité). L'agent choisi sera notifié par un message d'acceptation. Le résultat de la requête sera envoyée à l'agent émetteur une fois que l'agent choisi termine l'exécution de la requête. Dans cette stratégie la décomposition se fait par l'agent choisi (exemple 3), mais la recomposition des résultats se fait selon la stratégie appliquée par ce dernier (stratégie 1 ou stratégie 2).



Exemple 3 : Décomposition par l'agent E

Pour être plus clair, quant au processus de traitement des requêtes en utilisant les différentes stratégies, nous allons donner plus de détails sur les composants: Décideur, Planificateur, Replanificateur, Exécuteur et Wrapper d'un agent.

Définitions

Nous définissons l'état interne d'un agent par la formule défini par : $T_{ft}=(T_f+T_t)/2$;

où : T_f = taille de la file d'attente des requêtes restant à exécuter / taille totale pouvant être supportée. Et T_t = nombre de threads présents / nombre total pouvant être. Nous définissons ainsi les valeurs :

T_p = taux de présence des données au niveau de l'SGBD associé à l'agent pour une requête donnée. T_p est calculé en utilisant la requête donnée et l'ontologie locale de l'agent. T_p est égal au nombre d'éléments et des sous éléments (relations et attributs) présentés dans la requête et aussi présentés dans l'ontologie locale sur le nombre total d'éléments et des sous éléments présentés dans la requête.

Les deux valeurs a , b où : $a \in]0.5,1]$ et $b \in]0,0.5]$.

C_{com} = le coût de communication moyen d'une quantité de données de taille fixe entre deux agents.

Nous définissons la croyance Cr_e d'un agent A_i sur le taux du chargement d'un ensemble d'agents $A_1,...,A_j,...,A_p$ participent à l'exécution d'une requête Q , par la formule :

$Cr_e=\frac{\sum_{j=1}^{p} Cr_{A_i}^{A_j}}{p}$ où : $Cr_{A_i}^{A_j}$ est la croyance de A_i sur le taux du chargement de l'agent A_j est égale à 1 si l'agent A_i croit que l'agent A_j est chargé, et égale à 0 sinon.

La valeur d où : $d \in]0.5,1]$.

Nous définissons la croyance Cr d'un agent A_i sur le taux du chargement d'un ensemble d'agents $A_1,...,A_j,...,A_p$ par la formule :

$Cr=\frac{\sum_{j=1}^{p} Cr_{A_i}^{A_j}}{p}$ où : $Cr_{A_i}^{A_j}$ est la croyance de A_i sur le taux du chargement de l'agent A_j est égale à 1 si l'agent A_i croit que l'agent A_j est chargé, et égale à 0 sinon.

Nous définissons la capacité d'un agent A_i pour traiter une requête Q envoyée par l'agent A_j. Cette capacité est évaluée par l'agent A_j et calculée suivant la formule : $Cap_{A_i}^{Q} = xT'_{p_i} + yC'_{com_i} + zT'_{ft_i}$ où :

$T'_{p_i} = \frac{T_{p_i}}{\max(T_{p_1},...,T_{p_i},...,T_{p_m})}$, cette valeur représente le T_{p_i} normalisé de l'agent A_i,

$C'_{com_i} = 1 - \frac{C_{com_i}}{\max(C_{com_1},...,C_{com_i},...,C_{com_m})}$, cette valeur représente le C_{com_i} normalisé de l'agent A_i,

$T'_{ft_i} = 1 - \frac{T_{ft_i}}{\max(T_{ft_1},...,T_{ft_i},...,T_{ft_m})}$, cette valeur représente le T_{ft_i} normalisé de l'agent A_i.

Et x, y, z sont des nombres réels positifs chacun représente la priorité de la valeur qui lui est associé (x pour T'_{p_i}, y pour C'_{com_i} et z pour T'_{ft_i}).

5.3.2.2 Le Décideur

L'algorithme du Décideur est :

Décideur
Entrée: Q /* Q est une requête reçue d'un utilisateur ou d'un autre agent pour exécution */
Sortie : Stratégie i
{
  Calculer T_{ft}, T_p;
  Si ($T_{ft} > a$) alors
  {
    Si (($T_p < b$) AND ($Cr \neq 1$)) alors retourner Stratégie 3
    Sinon retourner stratégie 1
  }
  Sinon
  {
    Trouver les agents participant à l'exécution du Q ; // décompose Q//
    Calculer Cr_e ;
    Si ($Cr_e < d$) alors retourner stratégie 2
    Sinon retourner stratégie1* ;
  }
}

Le décideur choisit la stratégie 3 lorsque les trois conditions suivantes sont vérifiées :

1. le taux de chargement de l'agent est grand;
2. le taux présence des données est petit;
3. l'agent croit qu'il existe au moins un agent qui n'est pas chargé.

Le décideur choisit la stratégie 1 si l'une des conditions est vérifiée :

1- le taux du chargement de l'agent est grand et le taux de présence des données est grand ou l'agent croit qu'il existe des agents chargés.

2- le taux de chargement de l'agent est petit et il croit que le taux du chargement des agents participant à l'exécution de *Q* est grand.

Le décideur choisi la stratégie 2 lorsque le taux du chargement de l'agent est petit et croit que le taux du chargement des agents participant à l'exécution de *Q* est aussi petit.

5.3.2.3 Le Planificateur

Comme nous l'avons déjà décrit, le rôle du planificateur est la décomposition d'une requête en des sous requêtes. Le langage d'interrogation commun utilisé dans notre solution est une restriction de la grammaire de XQuery. Ce langage est défini dans[YU 04]. Cette grammaire est :

q ::= For $\$x_1$ in C_1,...,$\$x_n$ in C_m
Where B
Return r

r ::=$[A_1=r_1,...,A_k=r_k] \mid e \mid q$

e ::= $S \mid \$x \mid e/l$

C_i ::= $e \mid q$

où : $\$x$: est une variable, S : est la racine du schéma, l : est une étiquette, e/l : enregistrement de la projection

Définitions

Une substitution de l'ensemble de variables $X=\{x_1,..,x_i,...,x_n\}$est l'ensemble fini q de la forme $\{x_1/y_1,..,x_i/y_i,...,x_n/y_n\}$ où chaque y_i est une variable différente de x_i mais a le même type que x_i [FLOR 95].

Soit la substitution $q=\{x_1/y_1,...,x_i/y_i,...,x_n/y_n\}$ et Q une requête. Considérant les requêtes suivantes : $Q_1,...,Q_i,...,Q_n$ où : $Q_0=Q$ et Q_i est obtenue de Q_{i-1} par le remplacement de chaque occurrence de la variable x_i dans Q_{i-1} par y_i. Q_n est appelée l'instance de Q par la substitution q, et est notée par Qq .

Deux requêtes Q_1 et Q_2 sont dites logiquement équivalentes si et seulement si elles possèdent la même forme canonique (elles donnent les mêmes résultats)[FLOR 95].

Une requête est transformée en une forme dite simple en utilisant les règles suivantes :

- Transformer tous les prédicats dans la clause Where dans une forme normale conjonctive;
- Eliminer si elles existent, l'imbrication des requêtes présentées dans la clause For.

Deux requêtes Q_1 et Q_2 sont unifiables si Q_1 est l'instance de Q_2 par la substitution q. C'est à dire: $Q_1=Q_2 q$, on dit que Q_1 est logiquement équivalent à $Q_2 q$. Nous adaptons l'algorithme défini dans[FLOR 96] qui permet de vérifier l'unification de deux requêtes objet OQL. Notre algorithme est divisé en quatre étapes principales. L'algorithme est récursif (figure algorithme Unification) : en premier lieu il vérifie l'unification des collections ensuite il vérifie les prédicats et finalement les projections (return), si tout se passe bien, l'algorithme réussit et retourne la substitution q. La substitution q est calculée itérativement et on obtient une substitution q tel que : $Q_1=Q_2 q$.

Exemple :

Par exemple nous avons les deux requêtes *Q1*, *Q2*, et on veut savoir si *Q1* est unifiable avec *Q2* ou non :

```
Q1=     for $x in collection("SchemaGlobal")/livre
        where  $x/éditeur = "Edition EL FIKR" and $x/année = "2004"
        return  [ Titre = $x/titre]
```

Algorithme **Unification**(Q_1, Q_2)

Entrée : Q_1, Q_2 en forme simple.

Sortie : une substitution q tel que : Q_1 est logiquement équivalent à $Q_2 q$

Q_1 = For $\$x_{11}$ in C_{11},...,$\$x_{1n}$ in C_{1n}
Where p_{11} and p_{12} and ... and p_{1q}
Return $proj_1$

Q_2 = For $\$x_{21}$ in C_{21},...,$\$x_{2m}$ in C_{2m}
Where p_{21} and p_{22} and ... and p_{2k}
Return $proj_2$

{
soit $q = f$ (initialement vide),
soit UC= $\{C_{11},...,C_{1n}\}$,
soit UP= $\{p_{11},..., p_{1q}\}$,
// vérifier l'unification des collections
Pour chaque C_{2i} (i = 1,..,m)
{
s'il existe $C_{1j} \in UC$ tel que **Unification**(C_{1j}, $C_{2i} q$) réussit avec une substitution q ' alors :
$q = q \cup q' \cup \{\$x_{2i}/\$x_{1j}\}$ et supprimer C_{1j} de UC sinon échec.
}
Si $UC \neq f$ alors échec.
// vérifier l'unification des prédicats
Pour chaque p_{2i} (i = 1,..,k)
{
s'il existe $p_{1j} \in UP$ tell que **Unification**(p_{1j}, $p_{2i} q$) réussit avec une substitution q ' alors :
$q = q \cup q'$ ' et supprimer p_{1j} de Up sinon échec.
}
Si $UP \neq f$ alors échec.
// vérifier l'unification des résultats des deux clauses return (c'est la projection)
Si **Unification**($proj_1$, $proj_2$ q) réussit alors retourner q sinon échec.
}

Q2= for \$y in collection("SchemaGlobal")/livre
where \$y/éditeur =\$a and \$y/année = \$b
return [Titre = \$y/titre]

En appliquant l'algorithme unification(Q1, Q2), on vérifie si la requête $Q1 = Q2 q$.

L'étape1 : vérifier l'unification des collections dans la clause For : l'algorithme réussit d'unifier les deux collections et $q = \{\$y/\$x\}$.

L'étape2 : vérifier l'unification des prédicats dans la clause Where: l'algorithme réussit d'unifier les prédicats et $q = \{\$y/\$x, \$a/$"Edition EL FIKR",$\$b/$"2004"$\}$.

L'étape3 : vérifier l'unification des projections dans la clause Return : l'algorithme réussit d'unifier les projections et par conséquent il réussit d'unifier Q1 et Q2 et enfin retourne la substitution $q = \{\$y/\$x, \$a/\text{"Edition EL FIKR"}, \$b/\text{"2004"}\}$ où : $Q1 = Q2q$.

Si deux requêtes Q_1 et Q_2 ne sont pas unifiables, il est possible de vérifier s'il existe une requête Q_3 logiquement équivalente à Q_2 et contenant Q_1q comme sous requête. On dit que Q_3 est écrit en terme de Q_1q qui unifie avec Q_1 par la substitution q . Et Q_3 est la reformulation de Q_2 en utilisant Q_1. Nous adaptons l'algorithme défini dans[FLOR 95] qui permet de vérifier à partir de deux requêtes objet OQL Q_1 et Q_2, s'il est possible de reformuler Q_2 en une requête Q_3 contenant Q_1q comme sous requête. Notre algorithme(figure algorithme PossibleReformulation) est divisé en trois étapes principales: premièrement vérifier l'unification des collections ensuite les prédicats et finalement construit la nouvelle requête Q_3 et la substitution q .

Exemple :

Par exemple nous avons les deux requêtes *Q1*, *Q2*, et on veut savoir s'il est possible de trouver une requête *Q3* logiquement équivalente à *Q2* et contenant $Q1q$ comme sous requête.

```
Q1 =  for $y in collection("SchemaGlobal")/livre
      where  $y/éditeur = $a
      return   $y

Q2 =  for $x in collection("SchemaGlobal")/livre
      where  $x/éditeur = " Edition EL FIKR" and $x/année = "2004"
      return  [ Titre = $x/titre]
```

En appliquant l'algorithme PossibleReformulation(*Q1*, *Q2*), on vérifie s'il est possible de trouver une requête *Q3* logiquement équivalente à *Q2* et contenant $Q1q$ comme sous requête :

L'étape1 : vérifier l'unification des collections dans la clause For : l'algorithme réussit d'unifier les deux collections et $q = \{\$y/\$x\}$.

Algorithme **PossibleReformulation**(Q_1, Q_2)

Entrée : Q_1, Q_2 en forme simple

Sortie : une substitution q et une requête Q_3 est logiquement équivalente à Q_2 et contenant Q_1q comme sous requête.

Q_1 = For $\$x_{11}$ in C_{11},...,$\$x_{1n}$ in C_{1n} Where p_{11} and p_{12} and ... and p_{1q} Return $proj_1$	Q_2 = For $\$x_{21}$ in C_{21},...,$\$x_{2m}$ in C_{2m} Where p_{21} and p_{22} and ... and p_{2k} Return $proj_2$

{

soit $q = f$ (initialement vide), soit UC=$\{C_{21},...,C_{2m}\}$, soit MC=f ,

Pour chaque C_{1i} (i = 1,..,n) // vérifier l'unification des collections

{

s'il existe $C_{2j} \in UC$ tel que **Unification**($C_{1i}q$, C_{2j}) réussit avec une substitution q ' alors :

$q = q \cup q' \cup \{\$x_{1i}/\$x_{2j}\}$, supprimer C_{2j} de UC et ajouter C_{2j} à MC. sinon échec.

}

soit UP=$\{p_{21},..., p_{2k}\}$,

Pour chaque p_{1i} (i = 1,..,q) // vérifier l'unification des prédicats

{

s'il existe $p_{2j} \in UP$ tel que **Unification**($p_{1i}q$, p_{2j}) réussit avec une substitution q ' alors :

$q = q \cup q'$ ' et supprimer p_{2j} de UP sinon échec.

}

// construire la nouvelle requête Q_3

Soit $\$x_0$ une nouvelle variable différente de toute les variables existent,

Si $proj_1$ est de la forme [$champ_1$=expr$_1$,…, $champ_p$=expr$_p$] alors remplacer chaque expr$_i q$ dans $C_{21},...,C_{2m}$, $proj_2$ et les prédicats restés dans UP par $\$x_0/champ_i$ Sinon

Alors remplacer chaque $proj_1 q$ dans $C_{21},...,C_{2m}$, $proj_2$ et les prédicats restés dans UP par $\$x_0$

Soit *X* le sous ensemble de l'ensemble MC dont lesquels il existe des variables encore restent apparus dans UP ou dans $proj_2$.

Soit *Y* le sous ensemble minimum de MC vérifier les deux conditions :

- $X \subset Y$,
- Pour chaque $C_{2i} \in Y$, s'il existe $C_{2j} \in X$ avec $\$x_{2j}$ apparaît dans C_{2i} alors $C_{2j} \in Y$

Soit *lien_predicat*= f

Pour chaque $C_{2j} \in Y$

{

Si $proj_1$ contient un champ de la forme $champ_l$=$\$x_{2j}/AttrClé$ où *AttrClé* est l'attribut clé primaire dans C_{2j} et x_{2j} est une variable associée à C_{2j} alors ajouter la conjonction $\$x_{2j}/AttrClé = \$x_0/champ_l$ à l'ensemble *lien_predicat* sinon échec.

}

Retourner la substitution q et la requête Q_3 qui est logiquement équivalente à Q_2

Q_3 = For $\$x_0$ in Q_1q , $\$x_{2i}$ in C_{2i},...,$\$x_{2t}$ in C_{2t},$\$x_{2j}$ in C_{2j},...,$\$x_{2r}$ in C_{2r}

Where p_{2j} and ... and p_{2k} and p_{2a} and ... and p_{2f}

Return $proj_2$

Où: UC=$\{C_{2i},...,C_{2t}\}$, *Y*=$\{C_{2j},...,C_{2r}\}$, UP=$\{p_{2j},...,p_{2k}\}$ et *lien_predicat* =$\{p_{2a},...,p_{2f}\}$

}

L'étape2 : vérifier l'unification des prédicats dans la clause Where: l'algorithme réussit d'unifier le premier prédicat et $q = \{\$y/\$x, \$a/\text{"Edition EL FIKR"}\}$.

L'étape3 : la construction de la nouvelle requête *Q3* qui est logiquement équivalente à *Q2*:

Q3 = for \$x0 in *Q1q*
where \$x0/année = "2004"
return [Titre = \$x0/titre]

où : $q = \{\$y/\$x, \$a/\text{"Edition EL FIKR"}\}$ et

Q1= for \$y in collection("SchemaGlobal")/livre
where \$y/éditeur = \$a
return \$y

a- Décomposition des requêtes

Nous décrivons dans notre solution le processus de décomposition d'une requête *Q* écrite sur le schéma global en une requête de recomposition et des sous requêtes. Chaque sous requête q_i est écrite sur un schéma source S_{si}.

Notre processus de décomposition se fait sur quatre étapes : transformer la requête *Q* en une forme plus simple à traitée, reformulation, identification des agents participent à l'exécution de la requête, et la génération des sous requêtes.

1. La transformation de la requête consiste à l'écrire sous la forme canonique ou approchée à la forme canonique.

2. La reformulation de la requête *Q* (figure algorithme, reformulation) : notre algorithme consiste à reformuler une requête *Q* (en utilisant les règles de mapping *M*) en une requête logiquement équivalente à *Q* et écrite seulement en terme de(s) $q_{g_i}q$. Cette étape prépare à l'étape d'identification des sources de données participant à l'exécution de la requête *Q*. On distingue trois cas :

1- le cas où il existe une règle $r_i : q_{g_i} \rightarrow q_{s_i}$ tel que : $Q = q_{g_i}q$,

2- le cas où *Q* est en terme des $q_{g_i}q$, et

3- Le cas où il n'existe pas une reformulation de la requête à cause du manque de règles de mapping. Dans ce cas l'algorithme donne un échec.

Nous proposons que l'application des règles de mapping suive un ordre de priorité pour assurer une bonne reformulation et aussi permettre de prendre en compte les contraintes sur les sources qui sont définies dans les règles de mapping. Donc l'algorithme doit éviter les reformulations courtes.

Procédure **Reformulation**
Entrée : Q, M // où $M=\{r_1,r_2,...,r_i,...,r_n\}$ et $r_i : q_{g_i} \rightarrow q_{s_i}$.
Sortie : S
{
$S=\{Q\}$; $U=\{Q\}$; /* Q est en forme simple */
Tant que ($U \neq f$) faire
{
$w=f$;
Pour chaque $q \in U$ et $r_i \in M$
{
Vérifier **Unification**(q , q_{g_i}) si réussit avec la substitution q alors remplacer q par $q_{g_i}q$ (le remplacement se fait par l'entête de $q_{g_i}q$) sinon : Vérifier **PossibleReformulation**(q_{g_i} , q) si réussit avec la substitution q et la requête q ' alors remplacer q par q ' (tel que q ' contenant $q_{g_i}q$ comme sous requête).
Ajouter le résultat à w
$U=w-S$; $S=S \cup U$
}
S'il existe dans S une requête de la forme $q_{g_i}q$ ou de la forme :
For x_j in $q_{g_i}q_i$,...,x_t in $q_{g_{tt}}q_t$
Where p_h and p_j and ... and p_y
Return $proj$
Alors garder cette requête et éliminer les autres et retourner S
}
Retourner S
}

3. L'identification des agents participent à l'exécution de la requête (algorithme ci dessous). L'identification des agents (sources de données) participant à l'exécution de la requête est réalisée en utilisant la correspondance définie dans les règles de mapping entre une q_{g_i} et q_{s_i} .

Procédure **IdentificationParticipentAgent**
Entrée : Q reformulé , M // où $M=\{r_1,r_2,...,r_i,...,r_n\}$ et $r_i : q_{g_i} \rightarrow q_{s_i}$.
Sortie : Q en terme des schémas sources .
{
Pour chaque $r_i \in M$
{
Si q_{g_i} apparaît dans Q alors remplacer q_{g_i} par q_{s_i} dans Q
}
retourner Q
}

4. La génération des sous requêtes de la requête *Q*. Dans la génération des sous requêtes on distingue deux cas :

a- si la requête *Q* a la forme $q_{s_i}q$ (cas particulier) : *Q* est l'union des sous requêtes chacune portant sur des éléments d'un schéma source. La requête *Q* est considérée comme une requête de recomposition d'unions de sous requêtes.

b- Si la requête *Q* est écrite en terme des $q_{s_i}q$: dans ce cas la requête *Q* est considérée comme une requête de recomposition et les $q_{s_i}q$ sont considérées comme des sous requêtes chacune peut contenir des unions de sous requêtes chacune portant sur des éléments d'un schéma source.

Les sous requêtes destinées à chaque agent participant à l'exécution de *Q* sont regroupées pour être envoyées à ces agents.

Exemple

L'exemple simple suivant montre comment une requête est décomposée en sous requêtes. Soit le schéma global et les schémas associés aux agents A1, A2 et A3.

Schéma global :

Dept(DeptClé, Dnom, Budget) ;

Emp(EmpClé, Enom, DeptCléEtr, Salarie) ;

Le schéma global est écrit en XML schéma et interrogé par XQuery.

Schéma local de l'agent A1 :

Departement(DepartementClé, Dname, Bdg) ;

Schéma local de l'agent A2 :

Depart(DepartClé, DN, Budg) ;

Schéma local de l'agent A3 :

Employ(EmployClé, Ename, DCléEtr, wages);

La figure 5.3 montre les liens entre les différents schémas.

Soit la requête *Q* d'un utilisateur : supposons que la requête est décomposée par le planificateur de l'agent A1.

```
Q =     for $x in collection("SchemaGlobal")/Dept,  $y in collection("SchemaGlobal")/Emp
        where  $x/DeptClé= $y/DeptCléEtr and  $x/Dnom=  "département 1" and
               $y/Salaire= "20000,00 DA"
        return  [ Nom = $y/Enom]
```

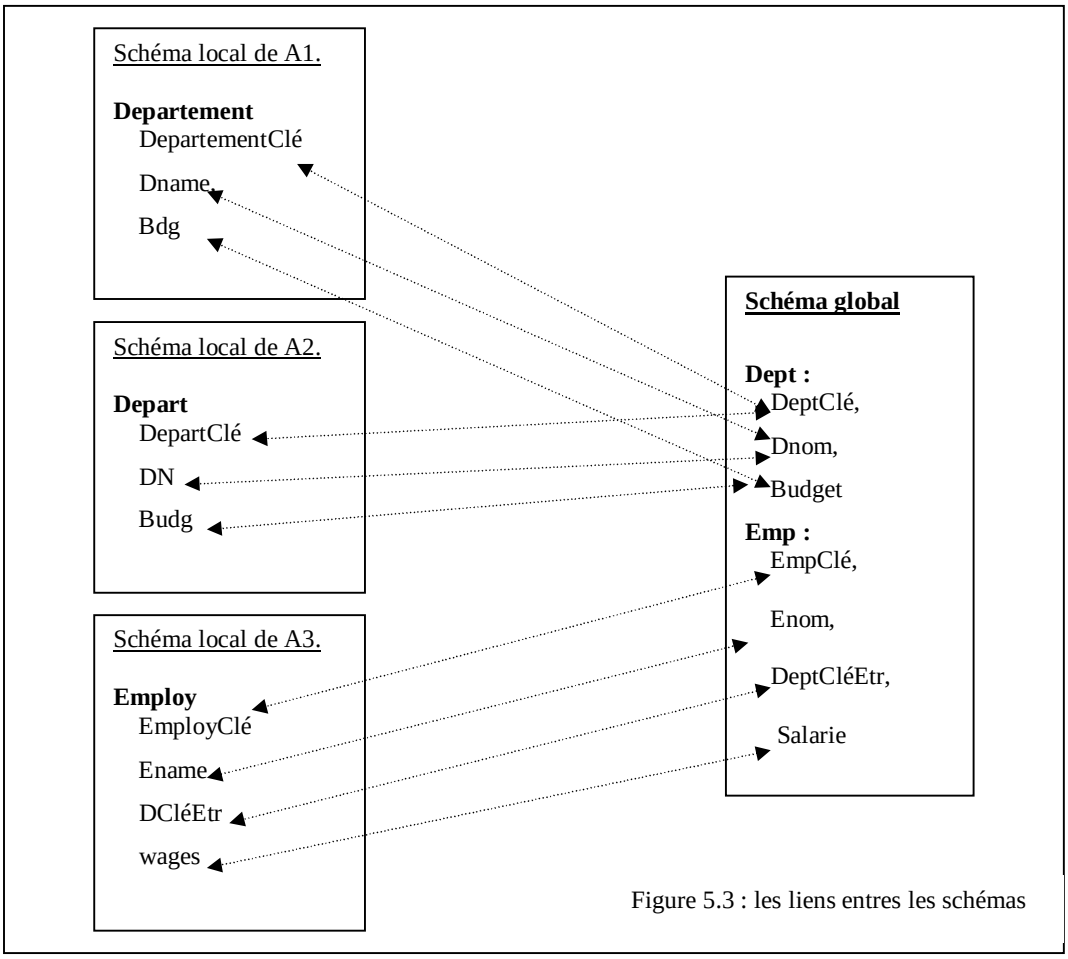


Figure 5.3 : les liens entres les schémas

et soit le sous ensemble des règles de mapping GLAV constitué de deux règles :

r_1	q_{g_1}	for $z in collection("SchemaGlobal")/ Dept where $z/Dnom= $a return $z
	q_{s_1}	for $z in (for $t in collection("SchemaLocal-AgentA1")/ Department where return [DeptClé=$t/ DepartmentClé, Dnom=$t/Dname , Budget=$t/bdg]) where $z/Dnom= $a return $z **union** for $z in (for $t in collection("SchemaLocal-AgentA2")/ Depart where return [DeptClé=$t/ DeparClé, Dnom=$t/DN , Budget=$t/budg]) where $z/Dnom= $a return $z
r_2	q_{g_2}	for $z in collection("SchemaGlobal")/ Emp where $z/Salaire= $a return [A1= $z/Enom, A2= $z/DeptCléEtr]
	q_{s_2}	for $z in collection("SchemaLocal-AgentA3")/ Employ where $z/wages= $a return [A1= $z/Ename, A2= $z/DCléEtr]

Pour décomposer Q en sous requêtes, on applique les étapes :

1- la simplification de la requête : Q est sous forme simple.

2- La reformulation : en applique l'algorithme de reformulation. La requête est reformuler en utilisant q_{g_1} :

Q = for $x0 in $q_{g_1}q$, $y in collection("SchemaGlobal")/Emp

where $x0/DeptClé= $y/DeptCléEtr and $y/Salaire= "20000,00 DA"

return [Nom = $y/Enom]

où : $q = \{\$z/\$x, \$a/"département1"\}$

Ensuite la requête est reformulée en utilisant q_{g_2}

Q = for $x0 in $q_{g_1}q$, $x01 in $q_{g_2}q'$'

where $x0/DeptClé= $x01/A2

return [Nom = $x01/A1]

où : q '= $\{\$z/\$x, \$a/"20000,00DA"\}$

De ce fait la requête est reformulée. Q est écrite en terme des $q_{g_i}q_i$

3- L'identification des agents participant à l'exécution de la requête Q :

- q_{g_1} correspond q_{s_1} (donc l'agent A1 et A2 participent à l'exécution de Q) et
- q_{g_2} correspond q_{s_2} (donc l'agent A3 participe à l'exécution de Q).

4- L'agent A1 possède la requête de recomposition *Q* et sa sous requête. L'agent A1 traduit sa sous requête dans le langage local associée à sa source locale (nous décrivons la technique de traduction en section 4.3.9). La sous requête de A1 est (après application de la substitution $q=\{\$z/\$x,\$a/"département1"\}$) :

```
for $x in ( for $t in collection("SchemaLocal-AgentA1")/ Department
        where return [DeptClé=$t/ DepartmentClé, Dnom=$t/Dname , Budget=$t/bdg ] )
where  $x/Dnom= "département1"
return  $x.
```

La sous requête destinée à l'agent A2 est:

```
for $x in ( for $t in collection("SchemaLocal-AgentA2")/ Depart
        where return [DeptClé=$t/ DeparClé, Dnom=$t/DN , Budget=$t/budg ] )
where  $x/Dnom=  "département1"
return  $x .
```

La sous requête destinée à l'agent A3 est:

```
for $x in collection("SchemaLocal-AgentA3")/ Employ
where  $x/wages=  "20000,00DA"
return  [ A1= $x/Ename, A2= $x/DCléEtr ]
```

5.3.2.4 Le Replanificateur

Le replanificateur est utilisé lorsque l'agent décide d'employer la stratégie 2. dans la stratégie 2 le Planificateur crée un Exécuteur pour attendre des informations sur les résultats de ces sous requêtes. Le Replanificateur analyse ces informations et régénère de nouveaux messages destinés aux agents participant à l'exécution de la requête. Ces messages contiennent des ordres de redirections des résultats vers d'autres agents ou des sous requêtes de recompositions des résultats. Nous nous inspirons de la stratégie d'optimisation concernant l'ordonnancement des jointures inter-site[EVRE 95]. L'idée est de traiter une jointure ou union de requêtes inter-site le plus tôt possible, dès que deux sous-requêtes ont produit leur résultat. Cette approche a deux avantages: elle remplace la stratégie d'optimisation statique qui détermine un plan d'exécution à partir d'un modèle de coût, et elle prend en considération le temps de réponse réel des sources.

Les informations sur le résultat d'une sous requête exécutée par un agent A_j sont : la taille TR_{q_j} du résultat, et les coûts de communication entre A_j et les autres participants à l'exécution de la requête *Q*. Notons qu'un agent peut exécuter un ensemble de sous requêtes

pour une requête donnée. Nous donnons les étapes principales du replanificateur (L'algorithme ci dessous).

Algorithme Re-planificateur

Entrée : *Q*, des que des informations sur $q_j,...q_i,...q_m$ des deux agents soit disponibles

Sortie : *Q*', des ordres de redirections et des sous requêtes de recomposition

Q a la forme For x_j in $q_{s_i}q_i$,...,x_t in $q_{s_{it}}q_t$

Where p_h and p_j and ... and p_y

Return *proj*

ou *Q* a la forme $q_{s_i}q_i$

où chaque $q_{s_i}q_i = \bigcup_{j=1}^{p} q_j \; q_i$ et $p=1,...n$

n : est le nombre d'agents du système SMA

{

- Identifier les sous requêtes appartient à une même collection $q_{s_i}q_i$ s'il existe alors :
  1- Estimer le coût de toutes les possibilités d'exécution des sous requêtes en utilisant les tailles TR_{q_j} et les coûts de communication entre les agents;
  2- Choisir la possibilité au coût minimum ;
  3- Mettre à jour de la requête de recomposition *Q*;
  4- Préparer les ordres de redirection des résultats et les messages des requêtes de recompositions.

- Sinon (ce cas est possible seulement si *Q* est à la forme complexe) vérifier si ces sous requêtes sont égales aux collections (une $q_{s_i}q_i$ est composée d'une seule sous requête, p=1) alors :
  1- Estimer le coût de toutes les possibilités d'exécution des sous requêtes de recompositions (jointure inter agents), en utilisant les formules classiques de la sélectivités (statistiques sur données) et les coûts de communication entre les agents;
  2- Choisir la possibilité au coût minimum ;
  3- S'il y a redirection des résultats alors rendre les deux $q_{s_i}q_i$ et $q_{s_j}q_j$ comme si elle appartiennent à un même agent et décomposer *Q* ;
  4- Mettre à jour de la requête de recomposition *Q* et Préparer les ordres de redirection des résultats et les messages des requêtes de recompositions.

- Sinon attendre d'autres informations sur les résultats

}

5.3.2.5 L'Exécuteur

Le rôle d'un exécuteur est multiple selon la mission pour laquelle est crée. Parmi les missions que fait un exécuteur :

1- Un Exécuteur est crée par le Planificateur pour exécuter une requête de recomposition Q suivant la stratégie 1 ou 2,
2- Un Exécuteur est crée par le Planificateur pour exécuter une sous requête q_i de son agent suivant la stratégie 1 ou 2,
3- Un Exécuteur est crée par l'Interpréteur pour exécuter une sous requête q_i d'un autre agent suivant la stratégie 1 ou 2,
4- Un Exécuteur est crée par l'Interpréteur pour décider d'accepter ou non l'exécution d'une requête Q d'un autre agent,
5- Un Exécuteur est crée par le Planificateur pour évaluer les capacités des agents qui acceptent l'exécution d'une requête Q . Dans ce cas, l'agent choisi est celui qui possède la plus grande capacité $Cap_{A_i}^{Q}$.

5.3.2.6 Le Wrapper

Le wrapper joue le rôle d'interface entre la source de données et le système multi agents d'intégration de données. Le wrapper fait principalement trois rôles :

1- La traduction des sous requêtes écrites en langage commun vers le langage local à la source locale.
2- La conversion des résultats obtenus de la source locale vers le format commun (XML). Si le résultat est des données relationnelles ou objets on construit l'arbre XML correspond au résultat, un document XML est créé.
3- La traduction du schéma de la source locale vers un schéma écrit en utilisant le modèle de données commun (XML Schema).

a- La traduction d'une sous requête XQuery vers le langage local de la source

Chaque agent possède ses règles de mapping qui lient son schéma local au schéma global. Pour chaque agent A ses règles de mapping qui lient son schéma local au schéma global sont notées par : $M_A = \{r_1, \ldots, r_n\}$ où $r_i : q_g \rightarrow q_s$

- q_g : est une requête XQuery portant sur des éléments du schéma global et
- q_s : est une requête XQuery portant sur des éléments du schéma local de l'agent A

.

Pour traduire une sous requête XQuery destinée à l'agent *A*, il suffit de remarquer que cette requête est unifiable avec une q_s de l'agent *A*. De ce fait pour permettre la traduction d'une sous requête, nous définissons une table de traduction (cette idée est inspirée de celle présentée dans[PENS 02]) qui correspond à chaque q_s écrite en XQuery une requête écrite en langage local de la source (SQL ou OQL). Cette méthode permet de supporter aussi les sources de données anciennes (legacy system) qui sont basées sur les modèles de données réseaux ou Codasyl, il suffit de définir une table de correspondance des vues (requêtes) entre les vues XQuery et les vues écrites en utilisant le langage propriétaire de la source.

Définition d'une table de traduction

Soit l'ensemble des requêtes $Q_s^A=\{q_{s1},...,q_{si}\}$ extrait de l'ensemble M_A (partie droite des $r_i : q_g \rightarrow q_s$) défini sur le schéma local de *A*, écrit sur le modèle XML Schéma, où chaque q_{si} est écrite en XQuery. Soit l'ensemble des requêtes $Q'^A_s=\{q'_{s1},...,q'_{si}\}$ défini sur un schéma local écrit en utilisant le modèle local (modèle relationnel ou objet), où chaque q'_{si} est l'équivalent de la vue q_{si} mais écrite en utilisant le langage local (SQL ou OQL selon la source). La table de traduction TT défini par l'ensemble : TT={$< q_{s1}, q'_{s1} >$,…,$< q_{si}, q'_{si} >$ }.

Pour exécuter une sous requête XQuery nous proposons les étapes :

1- Eliminer, en utilisant l'ontologie locale de l'agent et sa bibliothèque de fonctions, les conflits sémantiques de valeurs dans la sous requête XQuery lorsqu'ils existent. Cela se fait par une vérification du type des attributs en utilisant l'ontologie locale de l'agent. Vérifier qu'ils corresponds au niveau global ou non. Sinon utiliser une fonction de transformation de type pour modifier le type d'une valeur d'un attribut.
2- Vérifier l'unification de la sous requête XQuery par l'une des requêtes de l'ensemble $Q_s^A=\{q_{s1},...,q_{si}\}$ (figure algorithme TrouveUnification).

3- En utilisant la table de traduction, la correspondance de q_{sj} est q'_{sj}, donc la traduction de la sous requête XQuery q est la requête $q'_{sj}\,q$. Il reste d'envoyer la requête $q'_{sj}\,q$ à la source locale pour l'exécution.

Procédure **TrouveUnification**
Entrée : *q (sous requête XQuery)* , $Q_s^A=\{q_{s1},...,q_{si}\}$
Sortie : q_{sj} ,q
{
  Pour chaque $q_{sj}\in Q_s^A$
  {
    Vérifier **Unification**(q , q_{sj})
    Si réussit avec la substitution q alors Retourner : q_{sj} et q
  }
}

4- Convertir le résultat de $q'_{sj}\,q$ en format commun XML et éliminer les conflits sémantiques de valeurs s'ils existent (processus inverse). C'est à dire préparer les résultats à l'étape d'intégration au niveau global, à ce niveau on obtient le résultat final de la sous requête XQuery q.

Exemple

Considérant l'exemple précédant :

Supposons que l'agent A1 est associé à une source de données XML, l'agent A2 est associé à une source de données objets et l'agent A3 est associé à une source de données relationnelles.

Chaque agent possède sa table de traduction qui lie ses $q_{s1},...,q_{si}$ (qui sont écrits en utilisant XQuery) par $q'_{s1},...,q'_{si}$ (qui sont écrits en utilisant le langage local).

Puisque le langage local de A1 est XQuery alors A1 ne traduit pas sa sous requête :

```
for $x in ( for $t in collection("SchemaLocal-AgentA1")/ Department
        where return [DeptClé=$t/ DepartmentClé, Dnom=$t/Dname , Budget=$t/bdg ] )
where  $x/Dnom= "département1"
return  $x.
```

Donc il l'envoie directement à sa source locale.

L'agent A2 traduit sa sous requête XQuery vers OQL en utilisant sa table de traduction: la sous requête de A2 (qui est envoyée de l'agent A1) est :

```
for $x in ( for $t in collection("SchemaLocal-AgentA2")/ Depart
            where return [DeptClé=$t/ DeparClé, Dnom=$t/DN , Budget=$t/budg ] )
where  $x/Dnom=  "département1"
return  $x .
```

L'agent A2 vérifie l'unification de sa sous requête q avec l'une des q_{s_i} de sa table de traduction. L'unification réussit par q_{s_i} et la substitution $q=\{\$z/\$x,\$a/"département1"\}$. Donc la traduction de q est $q'_{si}q$ (qui est écrite en utilisant OQL).

q_{s_i}	For $z in (for $t in collection("SchemaLocal-AgentA2")/ Depart where return [DeptClé=$t/ DeparClé, Dnom=$t/DN , Budget=$t/budg]) where $z/Dnom= $a return $z
q'_{si}	Select $z From $z in (Select DeptClé:=$t.DeparClé, Dnom:=$t.DN , Budget:=$t.budg From $t in Depart) Where $z.Dnom= $a

$q'_{si}q$ =
```
Select $x
From  $x in ( Select DeptClé:=$t.DeparClé, Dnom:=$t.DN , Budget:=$t.budg
              From $t in Depart )
Where  $x.Dnom= "département1"
```

L'agent A2 envoie $q'_{si}q$ à sa source locale pour l'exécution.

Aussi l'agent A3 traduit sa sous requête XQuery q vers le langage SQL en utilisant sa table de traduction : la sous requête de A3 (qui est envoyée de l'agent A1) est :

```
for $x in collection("SchemaLocal-AgentA3")/ Employ
where  $x/wages=  "20000,00DA"
return  [ A1= $x/Ename, A2= $x/DCléEtr ]
```

Supposons qu'ils existent des conflits sémantiques de valeurs, la source associée à A3 code l'attribut *wages* en dollar (conflits sémantiques de valeurs : le niveau global en dinar et le niveau local en dollar). De ce fait l'agent A3 détecte ce type de conflits en utilisant sa ontologie locale et il fait appel à une fonction (de sa bibliothèque de fonctions) pour transformer la valeur de l'attribut *wages* ="20000,00DA" du dinar vers le dollar (20000,00DA=250,00DO).

L'agent A3 vérifie l'unification de sa sous requête q avec l'une des q_{s_i} de sa table de traduction. L'unification réussit par q_{s_j} et la substitution$q$ '= {$z/$x,$a/"250,00DO"}. La traduction de q est $q'_{sj}q''$ (qui est écrite en utilisant SQL).

q_{s_j}	For $z in collection("SchemaLocal-AgentA3")/ Employ where $z/wages= $a return [A1= $z/Ename, A2= $z/DcléEtr]
q'_{sj}	Select A1 as $z.Ename, A2 as $z.DCléEtr From $z as Employ Where $z.wages= $a

$q'_{sj}q''$ = Select A1 as $x.Ename, A2 as $x.DCléEtr
From $x as Employ
Where $x.wages= "250,00DO"

L'agent A3 envoie $q'_{sj}q''$ à sa source locale pour l'exécution.

A cette étape tous les agents A1, A2 et A3 ont envoyé leurs sous requêtes. Quand les résultats arrivent, chacun convertit son résultat vers le format commun XML (en prend en compte l'élimination des conflits sémantiques de valeurs s'ils existent) et l'envoie à l'agent A1. Supposons que le résultat de la sous requête de A1, A2 et A3 sont :

Résultat de la sous requête du A1	Résultat de la sous requête du A2	Résultat de la sous requête du A3
<resultat A1 > <tuple> <DeptClé>125</DeptClé> <Dnom> département1</Dnom> <Budget> "1500000,00DA" </Budget> </tuple> </resultat A1>	<resultat A2 > <tuple> <DeptClé>235</DeptClé> <Dnom> département1</Dnom> <Budget> "2400000,00DA" </Budget> </tuple> </resultat A2>	<resultat A3 > <tuple> <A1>Benabdelkader</A1> <A2>125 </A2> </tuple> <tuple> <A1>Brahim</A1> <A2>130 </A2> </tuple> </resultat A3>

Donc l'agent A1 possède tous les résultats nécessaires pour exécuter la requête de recomposition définie par :

Q = for \$x0 in $q_{g_1}q$, \$x01 in $q_{g_2}q'$'

where \$x0/DeptClé= \$x01/A2

return [Nom = \$x01/A1]

où: $q_{g_1}q$ =for \$x in (for \$t in collection("SchemaLocal-AgentA1")/ Department

where return [DeptClé=\$t/ DepartmentClé, Dnom=\$t/Dname , Budget=\$t/bdg])
where \$x/Dnom= "département1"
return \$x
union
for \$x in (for \$t in collection("SchemaLocal-AgentA2")/ Depart
where return [DeptClé=\$t/ DeparClé, Dnom=\$t/DN , Budget=\$t/budg])
where \$x/Dnom= "département1"
return \$x

et $q_{g_2}q'$'= for \$x in collection("SchemaLocal-AgentA3")/ Employ

where \$x/wages= "20000,00DA"
return [A1= \$x/Ename, A2= \$x/DcléEtr]

Le résultat final de la requête *Q* est :

<resultat *Q* >

<Nom> Benabdelkader</Nom>

</resultat *Q* >

5.3.3 Communication entre agents

Le processus de communication dans un système multi agent se base sur trois éléments: des actes de langage, une ontologie commune et des règles (ou protocoles) de conversation. Ces protocoles sont nécessaires pour q'un agent puisse réussir ses communications. C'est grâce à ces protocoles que l'agent arrive à interpréter les réponses reçues à ses messages. En respectant ces protocoles l'agent saura comment répondre aux messages des autres agents. Ainsi ils maximisent l'uniformité des interprétations et favorisent donc la stabilité du système. Les protocoles ou les règles de conversations utilisés dans notre solution sont identiques aux stratégies de coopération entre les agents qu'on a défini. De ce fait chaque agent de notre solution possède un gestionnaire de conversations.

Nous utilisons ici le terme *conversation* pour indiquer une instance particulière d'un protocole. Pour pouvoir permettre aux agents de participer à plusieurs tâches collectives simultanément (le cas de notre solution), la conversation elle-même doit être identifiée explicitement dans les messages, de façon à éliminer les ambiguïtés, en rattachant les messages aux contextes qui les concernent.

En pratique, dans un système donné, une partie des méta-informations de la communication sera implicite, une autre partie sera explicite, encodée dans les messages échangés. Cette partie explicite est plus importante dans les systèmes multi agents que dans d'autres systèmes distribués, ce qui offre de multiples possibilités[DENI 03] : Enrichir et expliciter la sémantique de chaque message, d'une façon générale. Permettre aux agents d'implémenter des protocoles plus compliqués ou optimisés grâce à l'élimination d'ambiguïtés. Enfin permettre à un agent d'effectuer plusieurs conversations simultanément avec d'autres agents ou avec le même agent mais dans des objectifs différents.

Nous avons adopté le langage FIPA-ACL[FIPA 00] dans notre solution (chapitre 4). FIPA-ACL supporte le concept de conversation, il définit cinq paramètres liés à la gestion de la conversation : protocol, conversation-id, reply-with, in-reply-to, et reply-by. En pratique, reply-with et in-reply-to sont peu utilisés, au profit de *conversation-id* qui identifie globalement la conversation.

Donc le protocole appliqué est référencé dans les messages, ainsi qu'un identifiant de la conversation. Par contre, les rôles que font les agents ne sont pas identifiés, ils sont implicites, déduits du performatif du premier message reçu par les participants, durant l'initialisation de la conversation.

5.3.3.1 Structure des messages entre agents

La structure d'un message dans notre solution est :

Elément	**Description**
Performative	Type de l'acte de langage du message
Sender	Emetteur du message
Receiver	Destinataire du message
Reply-to	Non utilisé
Content	Contenu du message
Language	Langage dans lequel le contenu est représenté
Encoding	Non utilisé
Ontology	Le nom de l'ontologie utilisé pour donner un sens aux termes utilisé dans le *content*
Protocol	Stratégie de coopération appliquée (stratégie 1, stratégie 1*, stratégie 2, stratégie 3)
Conversation-id	Identificateur de la conversation (instance d'un dialogue)
Reply-with	Non utilisé
In-reply-to	Non utilisé
Reply-by	Maintenant

Le gestionnaire du conversation gère la communication entre les agents au niveau des conversations.

Si l'agent est l'initiateur d'une conversation, il doit :

1- créer un identifiant de conversation unique, au moins dans le contexte de son utilisation;
2- associer cet identifiant à la structure représentant la conversation nouvellement crée, de manière à pouvoir reconnaître les messages lui appartenant par la suite;
3- générer le ou les premiers messages, et affecter les agents participants aux différents rôles;

Si l'agent n'est pas l'initiateur :

1- S'il reçoit un message initiant une conversation, il doit créer une structure mémorisant l'état de la conversation, et y associer l'identifiant spécifié en paramètre;
2- Si le message appartient à une conversation déjà existante, il doit le passer à la structure gérant le suivi de cette conversation;

A chaque message reçu, l'agent vérifie que le message est cohérent avec le protocole, et aussi vérifier la cohérence avec la sémantique réelle de la conversation. En cas d'incohérence, un message d'exception de type not-understood doit être généré et renvoyé à l'agent concerné[DENI 03].

Nous utilisons l'élément *protocol* pour deux objectifs :

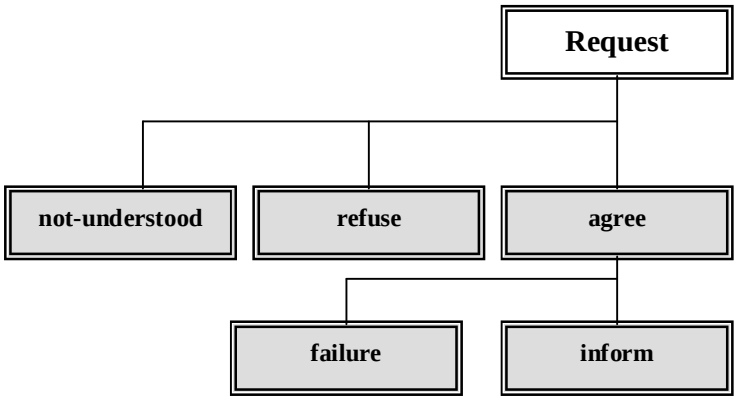
1- Identifier d'une manière explicite la stratégie de coopération appliquée;
2- Indiquer d'une manière implicite à l'agent récepteur l'état du chargement de l'agent émetteur (ces informations sont : stratégie 1, stratégie 1*, stratégie 2 ou stratégie 3). Cette information permet à l'agent récepteur de mettre à jour ses connaissances sur les autres agents et aide l'agent à décider d'une manière plus efficace afin d'améliorer le rendement global du système multi agent et éviter les interactions nuisibles.

Nous utilisons, pour des raisons de commodité, la notation employée dans[FIPA 97], pour décrire les protocoles (les stratégies de coopération) :

1- Les boîtes avec doubles bords représentent des actes de langage;
2- Les boîtes blanches représentent des actions effectuées par l'initiateur;
3- Des boîtes grises sont exécutées par l'autres participant(s) dans le protocole.

Nous nous inspirons du protocole *demande d'action* et *Contract net* de la FIPA[FIPA 97] pour décrire la stratégie 1 et la stratégie 3.

Stratégie 1



Lorsque l'agent initiateur décide d'appliquer la stratégie 1, l'agent envoie à chaque agent participant à l'exécution de la requête *Q* un message de la forme :

(**request**
: sender A_i
: receiver A_j
: content q_j
: language XQuery
: ontology XQueryOntology
: protocol Strategie1
: conversation-id conv-ij-1
)

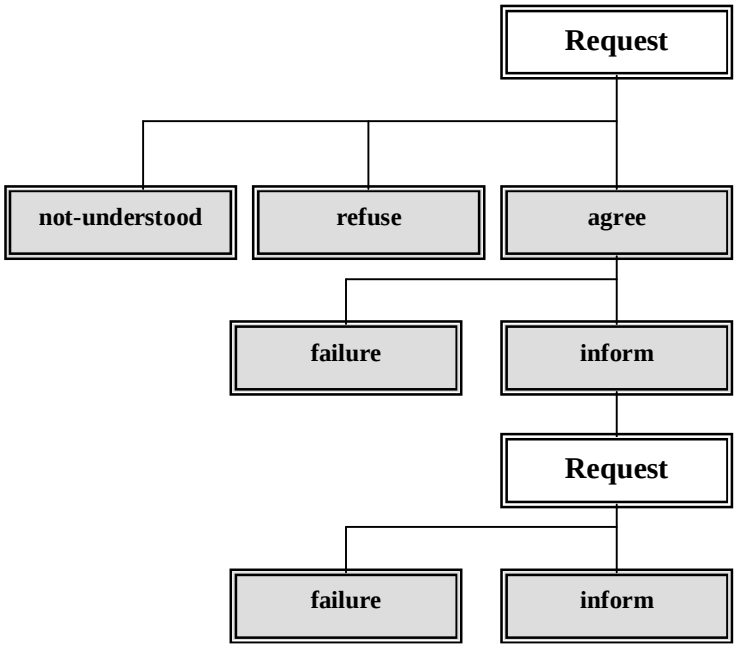
Par exemple l'agent A_i est l'initiateur de la conversation. Le message identifie d'une manière unique : l'émetteur, le récepteur, le protocole, et l'instance du protocole (conversation-id). L'agent A_i demande de l'agent A_j d'exécuter la sous requête q_j suivant la stratégie 1. l'agent A_j peut répondre selon trois manières :

1- il accepte l'exécution de la sous requête q_j et envoie à A_i l'accord en utilisant la performative **agree.** Durant l'exécution de q_j il peut survenir des échecs (par exemple requête mal reformulée), l'agent A_j envoie à A_i un échec (performative **failure**). Si tout se passe bien A_j donne les résultats à A_i (performative **inform**);

2- l'agent A_j peut refuser l'exécution de q_j (par exemple problème de sécurité) il envoie à A_i un message contenant la performative **refuse,** en expliquant pourquoi;

3- l'agent A_j peut ne pas comprendre le message de A_i ,il envoie alors un message contenant la performative **not-understood.**

Stratégie 2



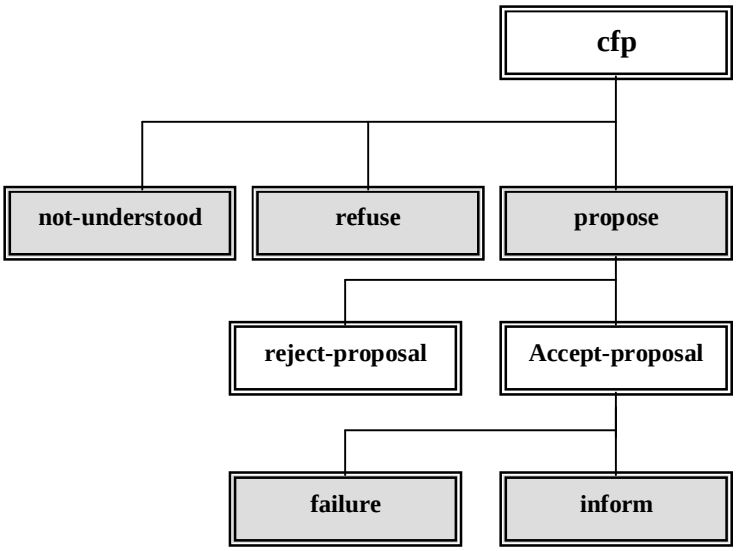
Lorsque l'agent initiateur décide d'appliquer la stratégie 2, l'agent envoie à chaque agent participant à l'exécution de la requête *Q* un message de la forme:

(**request**
: sender A_i
: receiver A_j
: content q_j
: language XQuery
: ontology XQueryOntology
: protocol Strategie2
: conversation-id conv-ij-1
)

Par exemple l'agent A_i est l'initiateur de la conversation. Comme dans la strategie1, l'agent A_j peut répondre selon trois manières :

1- il accepte l'exécution de la sous requête q_j et envoie à A_i l'accord en utilisant la performative **agree.** Durant l'exécution de q_j un échec peut survenir (par exemple requête mal reformulée), l'agent A_j envoie à A_i un échec (performative **failure**). Si tout se passe bien A_j informe A_i de la fin d'exécution et lui donne les informations sur la taille TR_{q_j} du résultat et les coûts de communication entre A_j et les autres participants à l'exécution de la requête *Q* (performative **inform**). L'agent A_i envoie à A_j un autre message **request** demandant d'exécuter une tâche selon le contenu du message : il peut demander à l'agent A_j d'envoyer le résultat à un autre agent ; ou d'exécuter une sous requête de recomposition ; ou d'envoyer le résultat à A_i. Durant l'exécution de la tâche, des échecs peuvent survenir, alors l'agent A_j envoie à A_i un échec (performative **failure**). Si tout se passe bien A_j informe A_i par un (performative **inform**);
2- Similaire à la strategie1;
3- Similaire à la strategie1.

Stratégie 3



Lorsque l'agent initiateur décide d'appliquer la stratégie 3, il envoie un appel d'offres (**cfp,** Call For Proposal) décrivant la requête aux agents qu'il croit ne sont pas chargés.

(**request**

: sender A_i

: receiver A_j

: content Q

: language XQuery

: ontology XQueryOntology

: protocol Strategie3

: conversation-id conv-ij-1

)

Cette stratégie ressemble au protocole Contract net. Chaque agent A_j recevait l'appel et fournit ses compétences (T_{p_j}, T_{ft_j}) pour exécuter la requête *Q*. Une fois que l'agent émetteur a reçu des réponses de tous les agents, il évalue ces capacités et fait son choix sur un agent pouvant exécuter efficacement la requête (l'agent a la plus grande capacité $Cap_{A_j}^{Q}$). L'agent choisi sera notifié par un message d'acceptation (**accept-proposal**), et les autres par un message **reject-proposal**. Le résultat de la requête sera envoyé à l'agent

émetteur une fois que l'agent choisi l'a terminé (**inform**). Comme pour les deux autres stratégies, il est possible que des exceptions surviennent, elles sont alors traiter comme précédemment.

5.3.4 Scénario

Dans ce scénario nous considérons un système multi agent constitué de quatre agents: A, B, C et D. Chaque agent fait une abstraction d'une source locale de données. Ces sources locales sont autonomes, hétérogènes et distribuées.

A partir des besoins globaux nous construisons un schéma global. Ce dernier est enrichie par une ontologie globale qui explicite les concepts du domaine. Chaque agent voulant intégrer le SMA doit informer l'agent administrateur. Ce dernier envoie à tous les agents un message contient les règles de mapping du nouveau agent, et chacun essaye de trouver des similitudes entre les nouveaux règles et sa base des règles GLAV (cela se fait en utilisant le mécanisme d'unification).

Pour faciliter l'intégration d'un agent et détecter les règles de mapping qui lient le schéma global par son schéma local, chaque agent possède une ontologie locale qui explicite les concepts de son domaine. Les conflits syntaxiques sont éliminés en utilisant le modèle de données commun XML Schema et le langage d'interrogation commun XQuery. Les conflits schématiques et sémantiques sont éliminés en utilisant des règles de mapping, des bibliothèques de fonctions et des ontologies locales des agents.

Supposons que la source locale associée à l'agent A contient des données relationnelles, la source locale associée à l'agent B contient des données objets, la source locale associée à l'agent C contient des données XML et la source locale associée à l'agent D contient des données relationnelles.

Les utilisateurs peuvent accéder parallèlement au SMA d'intégration de sources de données autonomes, hétérogènes et distribuées. Supposons dans ce scénario, qu'aucune erreur n'est produite lors de l'exécution d'une requête d'un utilisateur. Initialement les croyances des agents sur le taux du chargement des autres agents sont nuls.

Un utilisateur pose sa requête (écrite en XQuery) portant sur le schéma global destinée à l'agent A. Ce dernier reçoit la requête, il le met sur ses buts à accomplir. Il l'analyse et vérifie sa validité. Ensuite l'agent décide la stratégie de coopération appliquée. Cette décision est en fonction des trois paramètres qui sont la nature de la requête, le taux de chargement de l'agent (son état interne) et l'état de l'environnement (la croyance de l'agent sur les taux des chargements des autres agents). En fonction de ces paramètres l'agent A choisit la stratégie 1 ou 1* ou 2 ou 3 (l'algorithme décideur section 5.3.2.2).

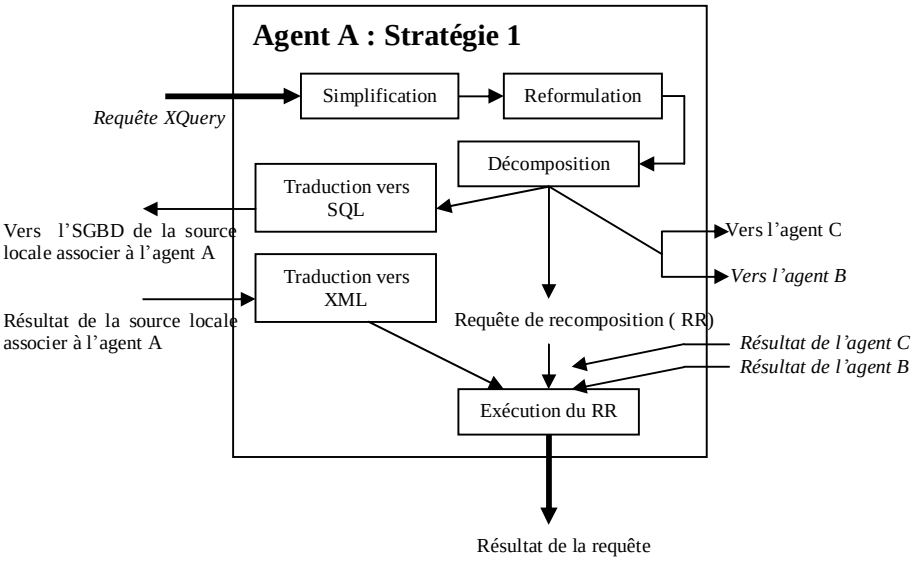


Figure 5.4 : Exemple d'exécution d'une requête selon la stratégie 1

Supposons que A choisit la stratégie 1 (figure 5.4), cela signifie que A est chargé et le taux de présence des données est grand ou A croit qu'ils existent des agents chargés. L'agent A joue le rôle d'un médiateur. Il réécrit la requête en une requête plus simple, puis il reformule cette requête en une requête portant sur des éléments des schémas locaux des sources participants à l'exécution de cette requête. Ensuite, il décompose la requête reformulée en une requête de recomposition et des sous requêtes portant chacune sur un schéma local, à ce niveau les conflits schématiques et sémantiques réapparaissent. L'agent A joue deux rôles parallèlement :

1. Le rôle d'un wrapper : s'il existe une sous requête (ou des sous requêtes) destinée à la source locale associée à l'agent A, alors il traduit sa sous requête du langage de requête commun XQuery vers SQL (cette traduction prend en compte les conflits

sémantiques) et l'envoie à sa source locale. La source locale exécute la sous requête SQL et envoie la réponse à l'agent A, qui traduit la réponse du format relationnelle vers le format commun XML en prenant en compte l'élimination des conflits sémantiques;

2. Identifie la règle de conversation (le protocole appliqué est la stratégie 1). L'agent A envoie à chaque agent participant à l'exécution de cette requête un message FIPA-ACL contenant une sous requête XQuery. Supposons que les deux agents B et C participent à l'exécution de cette requête. Donc chaque agent B et C reçoit un message de l'agent A. Le message envoyé à l'agent B ou C demande de lui d'exécuter une sous requête XQuery suivant la stratégie 1 et au même temps informe B et C que l'agent A est chargé, de ce fait B et C mettent à jour ses connaissances et ils croissent maintenant que A est chargé. L'agent B et C jouent le rôle 1 que joué A1, seulement l'agent B traduit sa sous-requête XQuery vers OQL et l'agent C ne fait pas de traduction parce que le format commun est XML. Les résultats de la sous-requête de l'agent B et C sont disponibles en format commun XML. Chaque agent B et C envoie la réponse à l'agent A sous forme d'un message FIPA-ACL contenant la réponse en format XML, à ce niveau l'agent A possède toutes les réponses des sous requêtes en format XML.

L'agent A exécute la requête de recomposition qui est portée sur ces résultats. Une étape de post-traitement consiste à filtrer le résultat de redondances. Finalement le résultat de la requête de recomposition est un document XML. Ce document est le résultat de la requête de l'utilisateur. L'agent A envoie le résultat final à l'utilisateur.

Dans le cas où l'agent A applique la stratégie 1* cela signifie que A n'est pas chargé et il croit que les agents participant à l'exécution de *Q* sont chargés. Tous se passe comme la stratégie 1, mais seulement les deux agents B et C croissent que l'agent A n'est pas chargé.

Si l'agent A applique la stratégie 2, cela signifie que A n'est pas chargé et il croit que les agents participant à l'exécution de *Q* ne sont pas chargés (B et C). L'agent A joue deux rôles parallèlement :

1. Le rôle d'un wrapper : similaire à la stratégie 1;

2. Identifie la règle de conversation (le protocole appliqué est stratégie 2). L'agent A envoie à chaque agent participant à l'exécution de cette requête un message FIPA-ACL contenant une sous requête XQuery. Supposons que les deux agents B et C participent à l'exécution de cette requête. Donc chaque agent B et C reçoit un message de l'agent A. Le message envoyé à l'agent B ou C demande de lui d'exécuter une sous requête XQuery suivant la stratégie 2 et au même temps informe B et C que l'agent A n'est pas chargé, de ce fait B et C mettent à jour ses connaissances et ils croissent maintenant que A n'est pas chargé. L'agent B et C jouent le rôle 1 que joué A1, seulement l'agent B traduit sa sous-requête XQuery vers OQL et l'agent C ne fait pas de traduction parce que le format commun est XML. Les résultats de la sous-requête de l'agent B et C sont disponibles en format commun XML. En fonction de la taille des résultats, des coûts de communication et de la nature de la requête, l'agent A génère des nouveaux messages destinés aux B et C pour améliorer l'exécution de la requête (algorithme replanificateur).

Lorsque l'agent A applique la stratégie 3 cela signifie que A est chargé et le taux présence des données est petit et l'agent croit qu'il existe au moins un agent qui n'est pas chargé. De ce fait l'agent A préfère qu'un autre agent exécute la requête. Les autres agents fournissent leurs capacités pour exécuter la requête et croissent maintenant que A est chargé. L'agent A évalue les capacités des autres agents et choisi la meilleure capacité (la méthode d'évaluation est expliquée en section 5.3.2.1 définitions).

Durant l'exécution des requêtes d'utilisateurs par les agents A, B, C, D, ces agents communiquent et échangent des informations suivant des règles bien déterminées, cela favorise la stabilité du système. Les agents éliminent tous les conflits des données et la distribution des données, ils donnent l'illusion aux utilisateurs qu'ils interrogent un système homogène et centralisé. L'architecture multi agents assure l'extensibilité et la flexibilité aux changement des sources de données et ainsi assure la scalabilité du système face à l'évolution du nombre de participants. En communiquant des informations sur l'environnement, les agents améliorent le traitement des requêtes dans le futur et évitent les interactions nuisibles.

5.4 Conclusion

Dans ce chapitre nous avons présenté une architecture multi agents pour l'intégration et l'interrogation d'une large classe de données hétérogènes et distribuées (données relationnelles, objets et XML) en prenant en compte les conflits sémantiques. Les agents font une abstraction totale de l'hétérogénéité et la distribution de données, ainsi ils donnent l'illusion aux utilisateurs qu'ils interrogent un système homogène et centralisé. Un agent dans notre solution est pourvu d'une large aptitude. Il coopère avec les autres agents, et selon des buts communs peut travailler comme un médiateur ou wrapper et a le choix de décider d'accepter ou refuser l'exécution de quelques types de requêtes afin d'améliorer le rendement global du système.

.

Conclusion générale

Notre travail consiste principalement à proposer une solution au problème d'intégration de sources de données hétérogènes et distribuées et de supporter les objectifs suivants: la transparence d'accès, la gestion des conflits sémantiques, l'autonomie des systèmes, l'extensibilité de l'architecture pour maîtriser l'ajout et le retrait de sources de données et la scalabilité du système face à l'évolution du nombre de participants. Cette solution combine l'approche de type médiation et l'utilisation du paradigme agent et utilise de nouvelles technologies afin de supporter cette large hétérogénéité et complexité des environnements. Notre solution est principalement caractérisée par :

1- Elle offre une transparence d'accès à la localisation, aux schémas sources et aux langages de requêtes des données sources. L'architecture est multi agents décentralisée et évite les points faibles des architectures centralisées;

2- La définition des stratégies de coopération entre agents pour l'efficacité du traitement des requêtes et l'amélioration du rendement global du système et assure la scalabilité du système. Chaque agent décide son comportement pour satisfaire les besoins définis au niveau global. Et la définition des algorithmes pour qu'un agent réalise ses buts;

3- Offre une intégration sémantique des données en utilisant de nouvelles technologies pour la représentation de la sémantique des informations et l'utilisation des ontologies pour différents objectifs;

4- Les données hétérogènes supportées sont des données relationnelles, objets et XML;

5- Utilise le langage d'interrogation expressive commun XQuery et le modèle XML Schéma comme modèle de données commun;

6- Approche d'intégration descendante et adaptation des règles de mapping GLAV offrent une flexibilité aux changements. Nous définissons les règles de mapping comme des correspondances entre des requêtes XQuery écrites en terme du schéma global et des requêtes XQuery écrites en terme des schémas sources;

7- Traitement des sous requêtes destinées aux sources de données permettant de supporter des systèmes anciens (legacy system). La traduction des sous requêtes destinées aux sources de données se fait sémantiquement, en prend en compte les conflits sémantiques de valeurs;

8- Les agents communiquent en terme de transfert de connaissances plutôt qu'en terme de transfert de données. Au cours de la communication les agents échangent implicitement des informations sur leurs états de chargement.

Sans doute, ce travail est susceptible d'amélioration. Nous en proposons les suivants :

1- Réaliser des algorithmes pour l'exécution d'une requête de recomposition,
2- Détailler l'algorithme de replanification,
3- Définir des algorithmes pour parcourir des ontologies,
4- Définir d'un langage pour décrire les ordres de redirections des résultats,
5- Exploiter l'historique des requêtes qui ont été exécuté pour améliorer la reformulation des requêtes,
6- L'utilisation de toutes les possibilités fournies du langage XQuery.
7- Une implémentation permettant de valider la solution proposée.

Bibliographie

[AMSA 96] L. Amsaleg, M. J. Franklin, A. Tomasic, and T Urban. Scrambling Query Plans to Cope With Unexpected Delays. In International Conference on Parallel and Distributed Information systems, Florida, 1996.

[ANGL ??] Stéphane Anglerot, Guillaume Bonnet, Guy Regnault, LES AGENTS INTELLIGENTS SUR INTERNET, IRESTE, DUTIL 10

[BAYA 97] R. J. Bayardo Jr.et al, "InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments", Microelectronics and Computer Technology Corporation,1997

[BENA 02] Ammar Benabdelkader, "Information Integration among Heterogeneous and Autonomous Applications" , Universiteit van Amsterdam, 2002

[BENH 05] Saber Benharzallah, Zaïdi Sahnoun, Coopération multi agents pour le traitement des requêtes sur des sources de données hétérogènes et distribuées, est soumis à ISPS 2005.

[BENS 99] Djamel Benslimane et al, « Interoperabilité de SIG : un état de l'art », université de Bourgogne, Lyon1,1999

[BERG 02] Bergamaschi S., Ilario Benetti, Benventano D., Guerra F., and Vincini M., "An information intégration framework for E-Commerce". IEEE Intelligente systems. 2002.

[BONN 99] Philippe BONNET, Prise en compte des sources de données indisponibles dans les systèmes de médiation, Thèse préparée au sein de l'INRIA Rhône Alpes (action Médiation du GIE Dyade), 25 janvier 1999

[BOUR 01] Ronald Bourret, Mapping W3C Schemas to Object Schemas to Relational Schemas, March, 2001,http://www.rpbourret.com/

[BOUZ 03] Bouzahzah Mounira, Mémoire Magister « Gestion de la sémantique dans l'intégration de données hétérogènes. Approche basée sur les logiques de descriptions » Université Mentouri de Constantine, 2003.

[BRUI 03] Jos de Bruijn, "Semantic Information Integration Inside and Across Organizational Boundaries", Multimedia Databases, Data and Knowledge Systems group, Faculty of Electrical Engineering, Mathematics and Computer

Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands, October 2003

[BUSS 99] Susanne Busse, Ralf-Detlef Kutsche, Ulf Leser, Herbert Weber, "Federated Information Systems: Concepts, Terminology and Architectures", Technische Université Berlin,1999

[CHAI 99] Brahim chaib-draa, Agents et systèmes multiagents (IFT 64881A), Note de cours, département d'informatique faculté des sciences et de génie, université LAVAL QUEBEC, Novembre 1999.

[CHAM 02] D. Chamberlin, XQuery: An XML query Language, IBM SYSTEMS JOURNAL, VOL 41, NO 4, 2002

[CHAZ 01] Grégory CHAZALON Joséphine LEMOINE, les schémas XML, Publié le 21 janvier 2001 sur http://site.voila.fr/xmlschema/

[CODY 95] W.F. Cody et al, Query Multimedia Data from Multiple Repositories by Content: The Garlic Approach. Proceedings of the Fifth International Workshop on Research Issues in Data Engineering- Distributed Object Management (RIDE-DOM'95), p. 124-131. Taipei, Taiwan, IEEE-CS, ISBN 0-8186-7056-8, march 1995.

[COST 02] Cécilia COSTES, Ronan HERVE, LES TECHNOLOGIES EMERGENTES DU WEB SEMANTIQUE, , école polytechnique de l'Université de Nantes ,PROJET SILR 3, Le 24 / 01 / 2002

[CULL 03] Nadine Cullot - Fabrice Jouanot- Kokou Yétongnon, Une méthode de réconciliation sémantique pour l'extraction de connaissances. Laboratoire Electronique Informatique Image université de Bourgogne – France, Laboratoire de Bases de Données (LBD) école Polytechnique Fédérale de Lausanne – Suisse, RSTI – 17/2003. EGC2003,

[DANG 03] Tayet Tram DANG NGOC, « Fédération de données semi-structruées avec XML », Université de Versailles Saint-Quentin-en-Yvelines, 2003

[DENI 03] *Denis Jouvin, Thèse de* Doctorat *: «* Délégation de Rôle et Architectures Dynamiques de Systèmes Multi-Agents Conversationnels*» , Université Lyon I – Claude Bernard, Décembre 2003.*

[DOGA 96] Dogac A. et al, Metu Interoperable Database System, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Canada, ACM Press, June 1996

[EVRE 95] Cem Evrendilek Asuman Dogac Query Decomposition, Optimization and Processing in Multidatabase Systems Software Research and Development Center Scientific and Technical Research Council of Turkiye Middle East Technical University (METU) Turkiye 1995

[FERB 95] J. Ferber. *Les systèmes multi-agents, vers une intelligence collective.* InterEditions, 1995.

[FINI 92] Tim Finin, Don McKay, Rich Fritzson, An Overview of KQML: A Knowledge Query and Manipulation Language, March 1992

[FIPA 00] FIPA Communicative Act Library Specification, 2000

[FIPA 97] FIPA 97 Specification, Part 2 , Agent Communication Language, *Geneva, Switzerland*

[FLOR 95] Daniela Florescu, Louiqa Raschid, Patrick Valduriez, Query Reformulation in Multidatabase Systems using Semantic Knowledge, INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE, mail 1995.

[FLOR 96] Daniela Florescu, Espace de recherche pour l'optimisation de requêtes objets, Thèse de doctorat de l'université de Paris 6, France, 1996.

[FURS 02] FREDERIC FURST, L'ingénierie ontologique, Institut de Recherche en Informatique de Nantes, RAPPORT DE RECHERCHE No 02-07 Octobre 2002

[GARC 97] M.GARCIA-MOLINA H., Y. Papakonstantinou, J. Ullman, D. Quass, S. Shawathe, J. Widom et K. Ireland. The TSIMMIS project: Integration of Heterogenous Information Sources, *Journal of Intelligent Information Systems*,(2): 117-132, 1997,

[GEIB 98] Jean-Marc Geib - Christophe Gransart - Philippe Merle, CORBA : des concepts à la pratique, Laboratoire d'Informatique Fondamentale de Lille Université des Sciences et Technologies de Lille, 1998

[GENO 02] Genoveva Vargas Solar_, Anne Doucet , « Médiation de données : solutions et problèmes ouverts », PARIS, 2002,

[GIRA 01] R. GIRARDI, "An analysis of the contributions of the agent paradigm for the development of complex systems", In (SCI 2001) and (ISAS 2001), Orlando, Florida. 2001.

[GIRAR 01] Didier Girard, Tanguy Crusson, XML pour l'entreprise, Version provisoire 0.91,2001, http://www.application-servers.com/livresblancs/xml/

[GRAB 00] M. Grabner, F. Gruber, L. Klug, W. Stockner ,EvalAgents, D2.1 - Agent technology: State of the Art, 2 / 29.08.2000

[HAKI 03] Farshad Hakimpour, Using Ontologies to Resolve Semantic Heterogeneity for Integrating Spatial Database Schemata, Mathematisch naturwissenschaftlichen Fakultat der Universitat Zürich, Zürich 2003

[HALE 01] A.Y. Halevy, Answering queries using views: A survey, The VLDB Journal 10: 270–294 (2001)

[JARR 02] Imed Jarras et Brahim Chaib-draa, Aperçu sur les systèmes Multiagents, Série Scientifique, Montréal Juillet 2002

[JASP 99] Robert Jasper and Mike Uschold, A Framework for Understanding and Classifying Ontology Applications, Boeing Math and Computing Technology, P.O. Box 3707, Seattle, USA, 1999

[JENH 03] Olfa Jenhani, Ontologies pour le WEB: relations, construction d'ontologies et méthodes de raisonnement pour la génération de langue naturelle, INRIA-ARC GeNI, Mai 2003

[JOUA 01] Fabrice Jouanot, « DILEMMA : vers une coopération de systèmes d'informations basée sur la médiation sémantique et la fusion d'objets », université de bourgougne, novembre 2001.

[KLUS 01] Matthias Klusch, Information Agent Technology for the Internet, *German* Research Center for Artificial Intelligence, Deduction and Multi-Agent Systems Lab, ,Germany,2001

[LARR 99] Larry M. Stephens and Michael N. Huhns, Database Connectivity Using an Agent-Based Mediator System, Center for Information Technology Department of Electrical and Computer Engineering University of South Carolina Columbia, 1999

[LENZ 01] Maurizio Lenzerini, Data integration is harder than you thought, Dipartimento di Informatica e Sistemistica Universit`a di Roma "La Sapienza", *CoopIS 2001* — Trento, Italy, September 5, 2001

[LENZ 03] Maurizio Lenzerini, Zoran Majkic, "General framework for query reformulation", SEWASIE project, 10/02/2003

[LESE 00] Ulf Leser, Query Planning in Mediator Based Information Systems", Informatik der Technischen Université Berlin, 17 September 2000

[LEVY 00] Alon Y. Levy, Logic-based techniques in data integration, Department of Computer Science and Engineering University of Washington,2000

[MANI 02] M. Mani, D. Lee, F. Chiu et W.W. Chu. Net and Cot : Translating Relational Schemas to XML Schema using semantic constraints. CIKM02, M Lean, Virginia, USA, 4-9 Novembre 2002, ACM, 2002

[NAAK 99] Huber Naake, « Modèle de coût pour médiateur de base de données hétérogènes », université de versailles saint- Quentin-en-yvelines, 28 septembre 1999

[NWAN 96] Hyacinth S. Nwana, Software Agents: An Overview, *Knowledge Engineering Review,* Vol. 11, No 3, pp. 205-244, October/November 1996.

[PARE 96] C. Parent , S. Spaccapietra : Intégration de bases de données: Panorama des problèmes et des approches , *école Polytechnique Fédérale de Lausanne, CH 1015 Lausanne,* Ingénierie des systèmes d'information Vol.4, N°3, 1996

[PASQ 01] Philippe Pasquier, Communication entre agents, doctorant DAMAS, université Laval, Canada. 13 septembre 2001.

[PENS 02] Loris Penserini, "integration and Coordination in both Mediator-Based and Peer-to-Peer Systems", universita degli studi di ancona,2002

[PITO 95] Evaggelia Pitoura, Omran Bukhres, and Ahmed Elmagarmid. Object Orientation in Multidatabase Systems. ACM Computing Surveys, 27(2) :141-195, june 1995.

[PREE 01] Preece, A. D., Hui, K.-Y., Gray, W. A., Marti, P., Bench- Capon, T. J. M., Cui, Z., and Jones, D. . Kraft: An agent architecture for knowledge fusion. International Journal of Cooperative Information Systems, 10(1-2):171–195, 2001.

[PUCH 99] P. PUCHERAL,Cohérence et performance des traitements multibases, Université de Versailles / St-Quentin,1999

[PURV 00] Purvis M, Cranefield S., Bush G., Carter D., McKinlay B., Nowostawski M. Ward R., The NZDIS Project: an Agent-Based Distributed Information Systems Architecture, Proceedings of the Hawai'i International Conference On System Sciences, January 4- 7, 2000, Maui, Hawaii.

[ROUS 02] Marie-Christine Rousset et al, « Construction de médiateurs pour intégrer des sources d'information multiples et hétérogènes » le projet Picsel , 2002

[SAID 03] Said Brahimi, Thèse magister :« Une approche de planification distribuée dans un environnement multi agent », Université Mentouri de Constantine, 2003.

[SERA 98] Luciano Serafini, Chiara Ghidini, "Using wrapper agents to answer queries in distributed information system",1998

[SHET 98] Amit P. Sheth, "Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics", 1998

[TAME 99] M. Tamer Ozsu, Patrick Valduriez, principal of distributed database systems, second edition, Prentice-Hall, Upper Saddle River, New Jersy 07458, 1999

[THOM 01] Jean-Jacques Thomasson ,XML Schema tome 0 : Introduction Recommandation du W3C du 2 Mai 2001, http://xmlfr.org/w3c /TR/ xmlschema-0/

[TIXI 01] Bruno TIXIER, La problématique de la gestion des connaissances, Le cas d'une entreprise de développement informatique bancaire, RAPPORT DE RECHERCHE No 01.9, IRIN, Université de Nantes, Septembre 2001

[TURK 97] Can Türker, Gunter Saake, stefan Conrad, Modeling Database Federations in Terms of evolving Agents, Otto-von-Guericke-Universitat Magdeburg, Institut für Technische Informationssyteme, Postfach 4120, D-39016 Magdeburg, Germany, 1997.

[VIKR 03] Vikram Goenka, Peterson's algorithm in a multi agent database system, Research Paper for 433-481, Knowledge representation and reasoning, July 2, 2003.

[WOHR 04] Alexander Wohrer and Peter Brezany, Mediators in the Architecture of Grid Information Systems, GridMiner TR2004-01, University of Vienna, Austria , February 2004.

[W3C 01] XQuery 1.0 : An XML Query Language, 07 June 2001. W3C Working Draft, http://www.w3.org/TR/2001/WD-xquery-20010607

[YANN 03] Yann Secq, Thèse Doctorat : «RIO: Rôles, Interactions et Organisations une méthodologie pour les systèmes multi-agents ouverts », l'Université des Sciences et Technologies de Lille, décembre 2003

[YU 04] Cong Yu, Lucian Popa, Constraint based XML Query Rewriting for data integration, SICMOD 2004, june 2004, Paris, France.

[ZARO 04] Nacereddine Zarour, Thèse Doctorat : " Contribution à la modélisation des SIs distribués et hétérogènes : le système DARAACHE « Application au entreprise de production » ", université de Mentouri Constantine, 2004.