Université Libre de Bruxelles

Transformation de requêtes algébriques dans un modèle Relationnel Objet

Directeur de mémoire:
Professeur E. Zimanyi

Mémoire présenté en vue de
l'obtention du grade de
licencié en informatique

Olivier Hubaut

Année académique 2002-2003

Table des matières

Chapitre 1 Introduction.... 6
Chapitre 2 Le projet MurMur.... 8
2.1 Le modèle MADS.... 9
2.2 Les outils MurMur.... 11
2.2.1 Editeur de schéma.... 11
2.2.1.1 Principe de fonctionnement.... 11
2.2.1.2 Exemples de transformation.... 13
2.2.2 Editeur de requêtes.... 14
2.2.3 Visualiseur de données.... 16
Chapitre 3 Objet du travail.... 17
3.1 Etat initial du projet.... 17
3.2 De MADS à MADSLog.... 18
3.2.1 L'algèbre MADS des requêtes.... 18
3.2.1.1 Origine.... 18
3.2.1.2 Quelques caractéristiques.... 18
3.2.1.3 Structure et Exemples.... 19
3.2.2 Règles de transformation.... 20
3.2.2.1 Les schéma en MADSLog.... 20
3.2.2.2 Les objets en MADSLog.... 23
3.2.2.3 Les fonction en MADSLog.... 23
3.3 De MADSLog au SGDB Oracle 9i.... 26
3.3.1 Structure d'une requête SQL.... 26
3.3.2 Recherche des Opérandes.... 27
3.3.3 Transcription des prédicats.... 28
Chapitre 4 Structures des données et Outils utilisés.... 29
4.1 XML.... 29
4.1.1 Description.... 29
4.1.2 Origine.... 30
4.1.3 Syntaxe.... 30
4.1.3.1 Données textuelles et balisage.... 31

4.1.3.2 Commentaires ... 32
4.1.3.3 Traitement du blanc et des fins de ligne ... 32
4.1.4 DTD ... 33
4.1.5 Représentation XML des données MADS et MADSLog ... 34
4.2 Java ... 34
4.2.1 Origine ... 34
4.2.2 Spécificités du langage ... 35
4.2.2.1 Orienté Objet ... 36
4.2.2.2 Absence de pointeurs ... 36
4.2.2.3 Garbage collector ... 37
4.2.2.4 Gestion des erreurs ... 37
4.2.2.5 Bytecode ... 38
4.3 JAXP ... 39
4.3.1 Le modèle SAX ... 39
4.3.2 Le modèle DOM ... 39
Chapitre 5 Méthodes utilisées ... 40
5.1 De MADS à MADSLog ... 40
5.1.1 Organisation des données ... 40
5.1.2 Structure du programme ... 41
5.1.3 Accès aux données et modifications ... 41
5.2 De MADSLog à Oracle 9i ... 43
5.2.1 Organisation des données ... 43
5.2.2 Principe de la NPI ... 44
5.2.3 Extensions de la NPI ... 45
5.2.3.1 Définitions ... 45
5.2.3.2 Opérateurs n-aires ... 45
5.2.3.3 Récupération des opérandes ... 46
5.2.4 Avantages et inconvénients ... 47
5.2.4.1 Avantages ... 48
5.2.4.2 Inconvénients ... 48
Chapitre 6 Conclusions ... 49

6.1 Etat des lieux 49
6.2 Pistes futurs 50
Chapitre 7 Remerciements 51
Chapitre 8 Bibliographie 52
Chapitre 9 Index 53

Chapitre 1 Introduction

Ce mémoire s'inscrit dans le cadre du projet MurMur. Dans ce travail nous détaillerons les recherches effectuées sur la mise en oeuvre d'un dispositif de traduction des données temporelles dans l'environnement des bases de données relationelles objet, et plus particulièrement Oracle 8i/9i. Cette étude sera effectuée tant sur le plan conceptuel que sur le plan de l'implémentation.

Le but de cette double approche est de mettre en évidence les mécanismes que de tels traitements imposent ainsi que de fournir un outil qui permette d'utiliser le fruit de ces recherches et qui s'intègre directement dans le cadre global du projet MurMur.

La première partie de ce mémoire est consacrée à une présentation globale du projet MurMur et de ses différentes composantes.

La deuxième partie présente l'analyse conceptuelle de la transfromation des données temporelles dans les deux étapes du processus, la traduction et la transcription.

La troisième partie décrit les différents outils utilisés pour l'implémentation des applications qui assureront la traduction de ces données.

La quatrième partie montre la mise en place de l'implémentation et les obstacles rencontrés lors de la mise en pratique des concepts. Pour cause de divers copyright et restrictions imposé par un partenaire du projet, les sources ne peuvent pas être mise à disposition, seul une approche algorithmique sera donc décrite.

Enfin, la cinquième partie reprend les conclusions quant à l'avancement du travail et des propositions de pistes futures visant à l'amélioration de celui-ci.

Chapitre 2 Le projet MurMur

Il existe actuellement quantité de SGDB[1] sur le marché, aux capacités et architectures multiples. Pour être efficace, un SGDB doit permettre à l'utilisateur de modéliser le plus fidèlement possible les objets à manipuler, tant dans leurs caractéristiques intrinsèques que dans leurs interactions avec leur environnement ou d'autres objets. Différentes architectures de SGDB ont été élaborées pour répondre à ces besoins, on peut citer le modèle « Relationnel Simple », le « Relationnel Objet » ou encore le modèle « Objet », mais aucun de ceux-ci n'est suffisament complet pour s'adapter à la diversité des objets que l'on peut être amené à traiter.

Un exemple pour montrer cette carence sont les données à caractère géographique. Celles-ci posent de nombreuses contraintes au niveau de l'implémentation :

- La précision des formes géométriques varie en fonction de l'unité choisie.
- Les formes géométriques sont susceptibles de varier dans le temps.
- Des objets peuvent changer de type (un terrain en friche devient un terrain à bâtir).
- Une même surface peut être représentée par différents objets (un pays peut être présenté dans sa globablité, mais également comme l'aggrégation de ses provinces / départements, ou comme un ensemble de terrains à batir, en friche ou forêts).

1 Système de Gestion de base de données

- Certains objets ont des interactions complexes (une inondation peut modifier des terrains et des habitations, une guerre également)
- ...

Ces contraintes peuvent pour la plupart être regroupée au sein d'une des trois catégories suivantes: multi-représentation, multi-résolution et gestion du temporel. Hors pour ces trois types de contraintes, les SGDB actuels n'offrent que peu de mécanismes de gestion à l'utilisateur, la plus grande part de l'implémentation étant laissée à ses soins.

Le projet MurMur veut pallier à ce manque en offrant à l'utilisateur une interface, avec les SGDB existants, lui permettant de reproduire au mieux la réalité avec laquelle il désire travailler. Pour ce faire, MurMur propose 3 modules: la conception de la base de données, un éditeur de requête et un visualiseur de donnée.

Le projet MurMur est un projet international financé par la communauté européenne. Il rassemble des partenaires académiques, publics et privés:

- CEMAGREF (Recherche pour l'Ingénierie de l'Agriculture et de l'Environnement)
- EPFL (Ecole Polytechnique Fédérale de Lausanne)
- IGN France (Institut Géographique National de France)
- Star Informatics
- ULB (Université Libre de Bruxelles)
- UNIL (Université de Lausanne)

Une description détaillée et l'ensemble des publications officiels sont accessibles à partir du site officiel du projet (http://www.mur-mur.org/).

2.1 Le modèle MADS

Afin de proposer à l'utilisateur un maximum de possibilités dans l'implémentation de sa réalité et de pouvoir la transposer ensuite dans un SGDB, il fallait définir un modèle de gestion des données complet et structuré. C'est dans

cette optique qu'a été réalisé le modèle MADS. Basé sur le modèle Entité-Association étendu, celui-ci tend à regrouper l'ensemble des possibilités offertes par les bases de données actuelles et même d'avantage.

Dans ce modèle, on trouve deux concepts de base: l'objet et la relation. La figure ci-dessous illustre leur agencement.

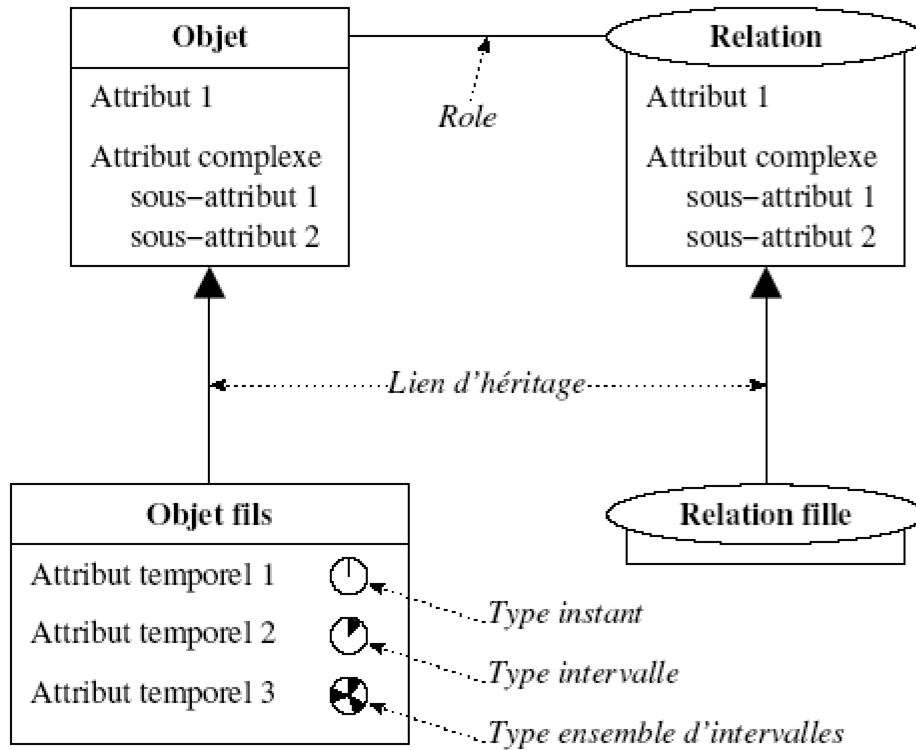

Illustration 1 : Concepts MADS d'objet et de relation

Le modèle repose sur les bases du modèle orienté-objet. Chaque objet et chaque relation peut ainsi disposer d'attributs, il peut exister des notions d'héritage entre les objets ou les relations, et les objets sont reliés aux relations par des rôles.

Le modèle MADS gère les attributs temporels et géométriques. Ces attributs peuvent être simples ou complexes, mono-valués ou multivalués. Le modèle gère également la multi-représentation. Un même objet peut donc avoir plusieurs représentations différentes suivant la vue choisie. Ces représentations peuvent avoir des rôles différents qui le relient aux relations.

A côté du modèle MADS original, il existe également une série de déclinaisons appauvries, appelées MADSLog, dont l'utilité est expliquée dans la section 2.2.

2.2 Les outils MurMur

Pour créer ces schémas MADS et les exploiter, le projet MurMur dispose de 3 applications distinctes, l'éditeur de schéma, l'éditeur de requête et le visaliseur de données.

2.2.1 Editeur de schéma

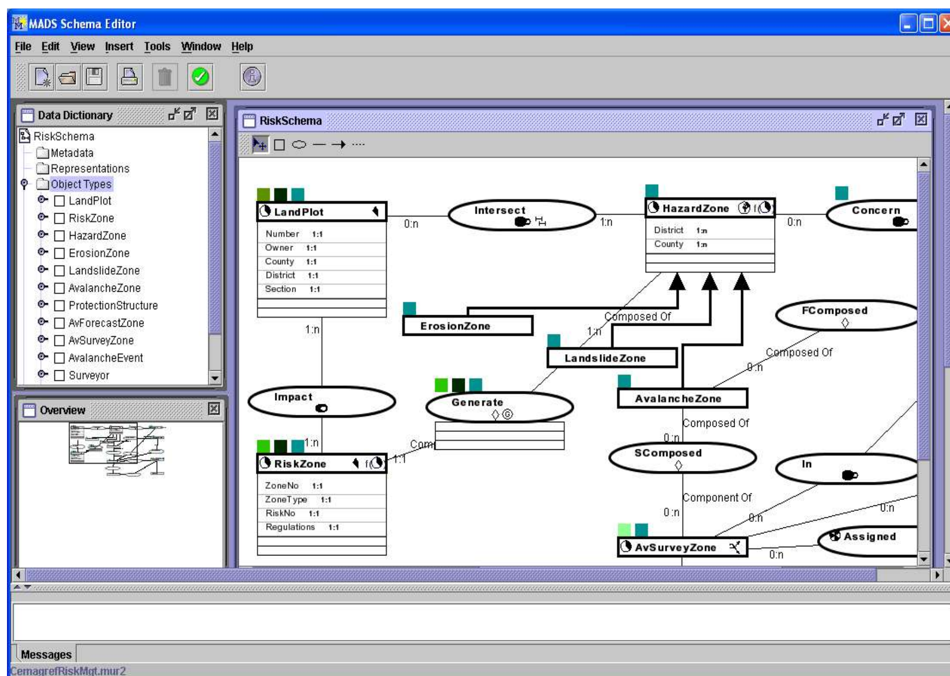

Illustration 2 : Interface graphique de l'éditeur de schéma

2.2.1.1 Principe de fonctionnement

Le premier module accessible à l'utilisateur est l'éditeur de schéma. C'est avec cet outil visuel que l'utilisateur va pourvoir définir le schéma de la base avec laquelle il voudra travailler. Ce module, comme les deux autres, fonctionne en trois étapes comme le montre l'illustration :

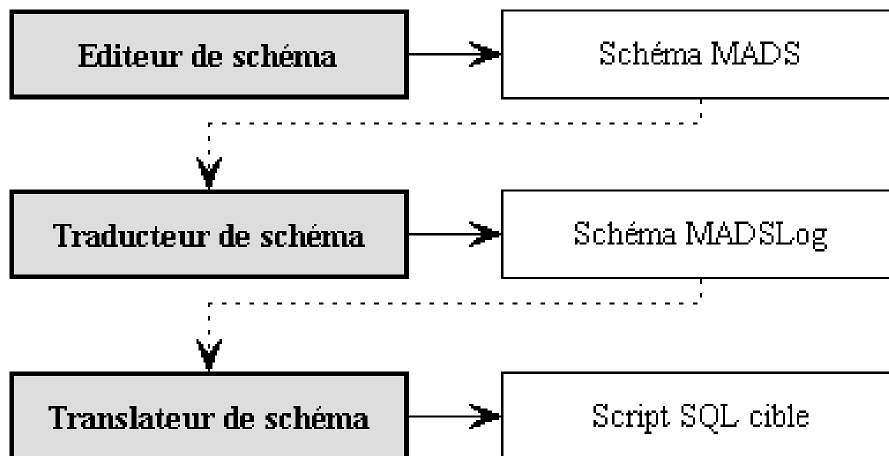

Illustration 3 : Etapes de transformation d'un schéma de base de données

- Tout d'abord, l'utilisateur créer un schéma visuel de la base qu'il désire obtenir. Ce schéma est sauvegardé selon le modèle MADS dans un fichier XML
- Dans un second temps, l'utilisateur va devoir se choisir un type de SGDB existant vers lequel il désire porter sa base. Dans les différents types de SGDB, on peut notamment citer, le relationnel, le relationnel objet et l'objet.
- Enfin l'utilisateur va se choisir un SGDB particulier (Oracle, SyBase, Access,...) qui contiendra effectivement sa base.

Lors de la deuxième étape, appellée « traduction », le choix de l'utilisateur va limiter les fonctionnalités disponibles en fonction du type de SGDB choisi. C'est ici que les modèles MADSLog interviennent. Chaque type de SGDB possède son modèle MADSLog qui définit l'ensemble des possibilités qu'un tel type de SGDB devrait pouvoir offrir. Lors de cette étape, le module va devoir transformer les objets et attributs du modèle MADS n'existant pas dans le modèle MADSLog du type de SGDB afin d'offrir au final les même possibilités de manière transparente pour l'utilisateur.

Ceci se réalise notamment en ajoutant certaines contraintes de vérification[2]. Le modèle MADSLog offre donc une étape générique globale à tous les SGDB d'un même type, évitant ainsi de réécrire une partie des transformations pour chaque SGDB. Ce modèle est également implémenté au sein d'un fichier XML à la DTD

2 Actuellement, de telles contraintes n'ont pas encore été implémentées dans la traduction.

aussi proche que possible de celle du modèle MADS.

La troisième étape, appelée « transcription », consiste en l'implémentation proprement dite au sein d'un SGDB, via requêtes SQL, du schéma de la base de données. Ici aussi, certaines transformations du schéma MADSLog peuvent apparaître si le SGDB choisi ne dispose pas (encore) de toutes les fonctionnalités prévues pour ce type de SGDB.

Au final, la base de données implémentée peut fortement différer dans sa structure du schéma initial, mais son utilisation devrait être identique à celui-ci grâce aux deux autres modules qui traduisent de façon la plus transparente possible pour l'utilisateur les actions sur son schéma en actions réalisables dans le SGDB sélectionné.

2.2.1.2 Exemples de transformation

Exemple 1 : Les attributs multivalués n'existent pas dans les SGDB relationnels et relationnels objets. Suivant le type de SGDB, deux options sont envisageables, la création d'une table supplémentaire (ainsi que la relation correspondante) contenant les différentes valeurs de l'attributs ou la création d'un domaine contenant ces mêmes attributs.

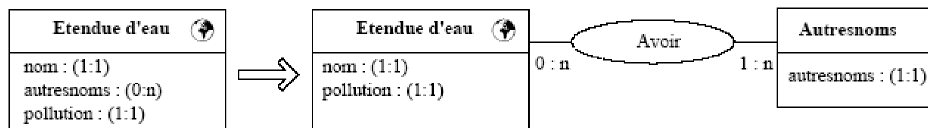

Illustration 4 : Transformation d'un attribut mutlivalué dans un SGDB relationnel

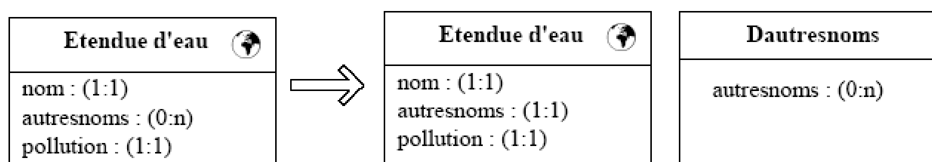

Illustration 5 : Transformation d'un attribut multivalué dans un SGDB relationnel objet

Exemple 2 : La notion de spatialité sur les objets n'est également pas supportée dans la plupart des SGDB, il faut donc, au minimum, transférer celle-ci dans un attribut.

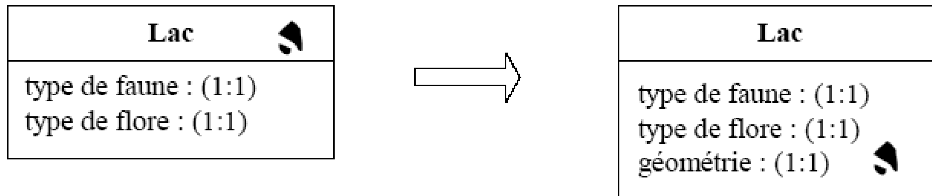

Illustration 6 : Transformation d'un objet spatial en objet avec un attribut spatial

2.2.2 Editeur de requêtes

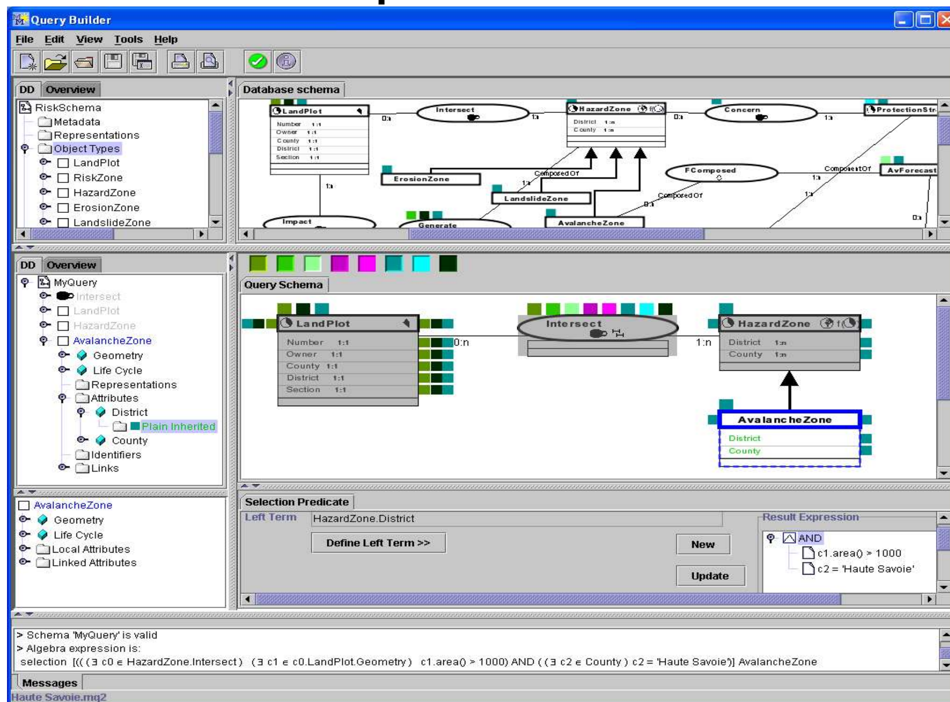

Illustration 7 : Interface graphique de l'éditeur de requête

L'éditeur de requêtes est le deuxième module du projet MurMur. Grâce à lui, l'utilisateur peut interroger la base de données physique comme s'il interrogeait son schéma MADS. Dans la version actuelle du traducteur, seul la partie consultation des données à été envisagée, l'insertion, la modification et la suppression étant laissées pour un futur développement du module. A l'instar du module de traduction de schéma, le traducteur de requête fonctionne également en trois étapes, illustrées

dans le schéma suivant:

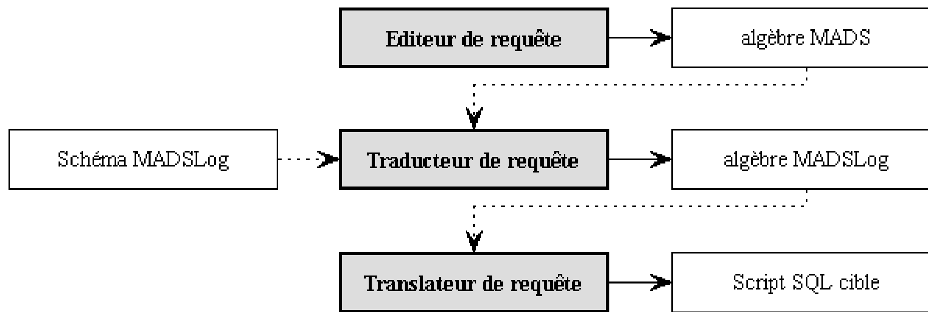

Illustration 8 : Etapes de transformation d'une requête

- L'utilisateur crée sa requête en se basant sur le schéma MADS de sa base.
- Une première traduction transforme la requête générique en une requête propre au type de SGDB choisi.
- Une deuxième traduction produit le code SQL d'interrogation de la base de données spécifique.

Lors de la première étape, l'utilisateur peut soit travailler avec un éditeur visuel, soit formuler directement sa requête. Dans les deux cas, les requêtes sont formulées sous forme d'expressions algébriques. Ces expressions sont ensuite sauvegardées et implémentées au format XML.

La deuxième et la troisième étapes sont les pendants des transformations appliquées au schéma de la base par le traducteur de schéma. On peut cependant observer que le comportement n'est pas identique, la traduction d'une requête n'ayant pas besoin d'un mécanisme aussi lourd que la traduction d'un schéma. En effet, les transformations de requêtes peuvent se faire dans un ordre séquentiel simple.

2.2.3 Visualiseur de données

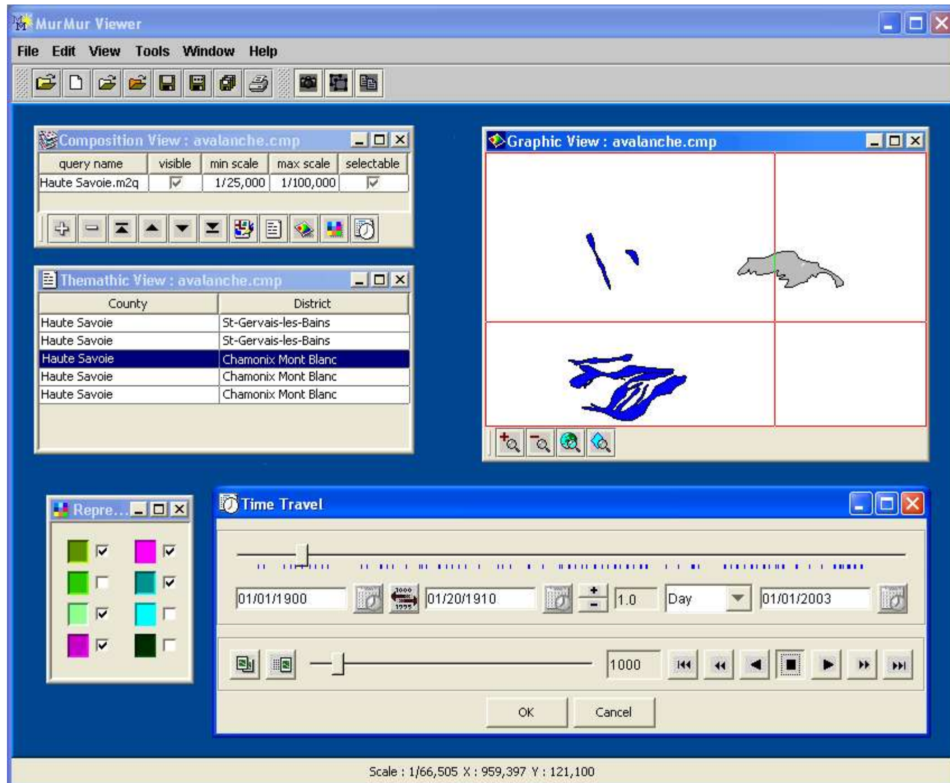

Illustration 9 : Visualiseur de données

Le visualiseur de données est le dernier module du projet MurMur. Celui-ci doit permettre à l'utilisateur de consulter les données qu'il aura demandés à la base via l'éditeur de requête. Ce module doit donc opérer de manière opposée aux deux autres.

Chapitre 3 Objet du travail

Le but de notre travail comporte deux volets : le développement et l'implémentation de la gestion des données temporelles au sein de l'éditeur de requêtes pour la consultation des données. Ceci comprend la transformation des requêtes du format MADS au format MADSLog pour les SGDB orientés objets, comme Oracle 8i et 9i, et la traduction du format MADSLog des requêtes en script SQL[3].

3.1 Etat initial du projet

Au début de ce travail, le projet disposait du module d'édition de schémas et du module d'édition de requêtes, tous deux développés pour un portage vers le SGDB Oracle 9i.

Le module d'édition de schémas est opérationnel dans les trois stades de conception/transformation d'un schéma, excepté dans l'implémentation des contraintes liés aux transformations. Il produit des bases de données Oracle 9i valides mais sans contraintes d'intégrité.

Le module d'édition de requêtes dispose d'un éditeur visuel dont certains disfonctionnement ont pu être observés lors de la réalisation de requêtes de test. Le traducteur MADS vers un modèle MADSLog relationnel-objet ne disposait pas de gestion des attributs temporels, géographiques et de mutli-représentations. Le

3 Structured Query Langage

transcripteur MADSLog/Oracle 9i ne disposait pas de gestion des attributs temporels ni de la multi-représentation.

3.2 De MADS à MADSLog

3.2.1 L'algèbre MADS des requêtes

3.2.1.1 Origine

Les bases de données relationnelles disposent d'une algèbre bien standardisée pour l'expression de leurs requêtes, même si certaines petites variations de présentation peuvent encore apparaître (comme le choix des opérateurs). Mais jusqu'il y a peu, il n'existait aucune algèbre standardisée pour les bases de données orientés objets. Le modèle MADS se rapprochant plus de ces dernières que du modèle relationnel simple, il fallait lui adjoindre une algèbre complete, simple, mais qui ne limiterait pas le portage des requêtes vers des SGDB moins fournis.

3.2.1.2 Quelques caractéristiques

L'algèbre MADS diffère de l'algèbre des bases de données relationnelles en plusieurs points:

- L'algèbre MADS accepte comme opérande aussi bien les objets que les relations. Ce choix se base sur le fait que le modèle MADS pose ces deux entités comme issues d'une même super-classe et que les relations, tout comme les objets supportent la notion d'héritage.
- Le résultat d'un expression algébrique MADS peut, au choix, être un objet, une relation ou les deux (via une super-classe). Ceci est une conséquence logique de la première caractéristique et de la propriété d'inclusion d'expressions algébriques au sein d'autres expressions algébriques. Cette caractéristique implique que tout opérateur manipule et retourne des objets, des relations, ou un type virtuel similaire.
- L'algèbre MADS supporte aussi bien les requêtes préservant les objets (chaque

entité résultante peut être directement reliée à une entité de la base de données) que les requêtes générant des objets (les entités résultantes n'ont plus une relation « 1-1 » avec les objets dont elles sont issues).

- L'algèbre MADS supporte bien sûr tous les différents types d'attributs, d'objets et de relations disponibles dans le modèle MADS, ainsi que diverses fonctions de manipulations sur ceux-ci.

3.2.1.3 Structure et Exemples

La forme générique d'une expression algébrique MADS, exprimée par une expression régulière, est la suivante:

Opérateur [Prédicat*] Opérande1, Opérande2?

Exemple 1 : Supposons une base de données dont le schéma MADS est illustré ci-dessous:

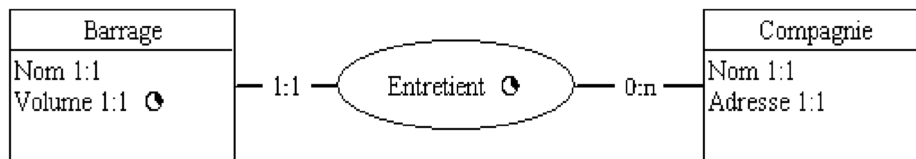

Illustration 10 : schéma MADS d'une base de données

Cet base de donnée contient deux éléments temporels: le volume d'eau contenu par un barrage et la relation qui lie un barrage à une compagnie d'entretien. Une requête dans cette base demandant de sortir toutes les Compagnies ayant travaillé sur un ou plusieurs barrages dont le volume en 2000 était supérieur à 80.000 litres aura la syntaxe suivante:

$$selection\,[\exists e \in Entretient \,|\, \exists b \in e.Barrage\; b.volume.attime(2000) > 80.000\,]\; Compagnie\; C$$

Ici, les composantes temporelles entrent en jeu, de manière implicite pour « Entretient », qui permet ou non la relation Compagnie – Barrage, et de manière explicite pour le volume grâce à la fonction « attime() » qui permet de sélectionner une valeur en fonction du temps.

Exemple 2 : Supposons le schéma MADS suivant:

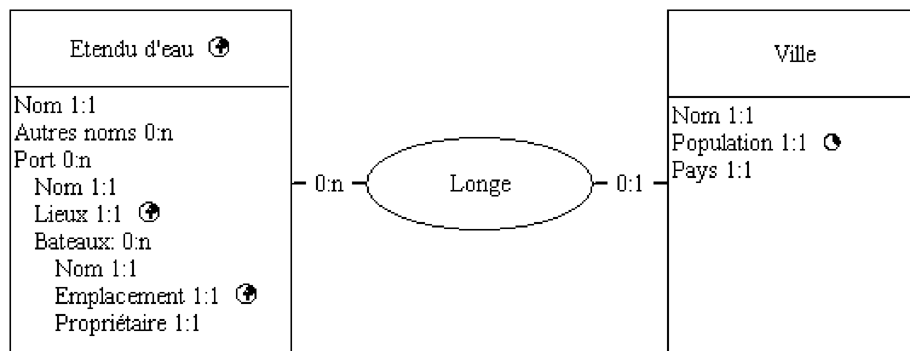

Illustration 11 : schéma MADS d'une base de données

Si l'on désire connaître les villes à côté desquelles un bateau appartenant à l'armateur 'xyz' est à quai, on produira une telle requête:

$$selection[\exists l \in Longe | \exists e \in (l.Etendue\ d'eau)\ \ e.Port.Bateaux.Propriétaire='xyz']\ Ville\ V$$

Dans cette requête, le mécanisme objet entre en jeu, le propriétaire est accessible grâce à la possibilité de descendre directement au sein des attributs multivalués et/ou complexes.

3.2.2 Règles de transformation

Les SGDB orientés objets, comme Oracle, ne disposent pas de toutes les fonctionnalités prévues dans L'algèbre MADS. La gestion du temporel y est inexistante et il faut donc passer par une transformation des requêtes algébriques dans un format appauvri mais dont le résultat sera identique à la requête originale.

3.2.2.1 Les schéma en MADSLog

Pour que la translcription de la requête algébrique en script SQL soit facilitée au maximum, il faut que la version MADSLog de l'expression algébrique possède une structure similaire au schéma MADSLog de la base de données. Pour obtenir ce résultat, il est impératif, lors de la traduction de l'expression algébrique, de disposer de ce schéma, comme le montre l'Illustration 8.

Pour comprendre les transformation appliquées au schéma pour la gestion des

données temporelles, il convient d'observer comment celles-ci sont définies dans le modèle MADS. L'illustration ci-dessous montre les différents éléments temporels possibles, de la ligne du temps, et leur hiérarchie:

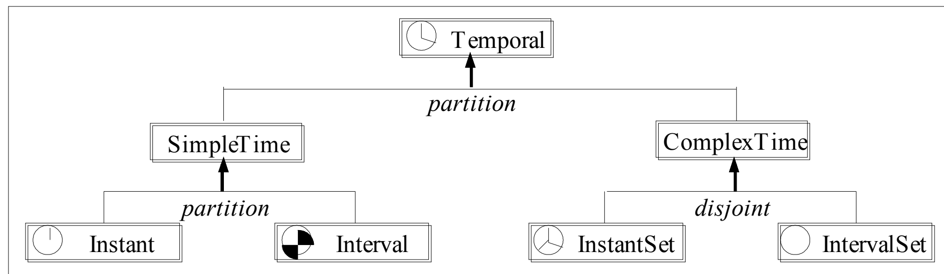

Illustration 12 : Hiérachie des éléments temporels

En sus, MADS définit également un type abstrait « TimeSpan » qui représente une durée mais qui ne se place pas sur la ligne du temps.

De ces différents types, nous pouvons en extraire deux que l'on peut qualifier « d'originels »: l'instant et l'intervalle. A cela, on pourrait ajouter la granulosité de la ligne du temps. Selon les besoins, celle-ci peut être discrète ou continue, mais le raisonnement pour la traduction n'en sera pas affecté.

Dans la traduction de schéma, la gestion des éléments temporels passe par plusieurs phases reprises ci-dessous:

1. S'il s'agit d'un objet ou d'une relation, l'élément temporel devient l'un des attributs de celui-ci. S'il s'agit d'un attribut, celui-ci se transforme en attribut complexe (si nécessaire) pour inclure l'élément temporel comme sous-attribut.

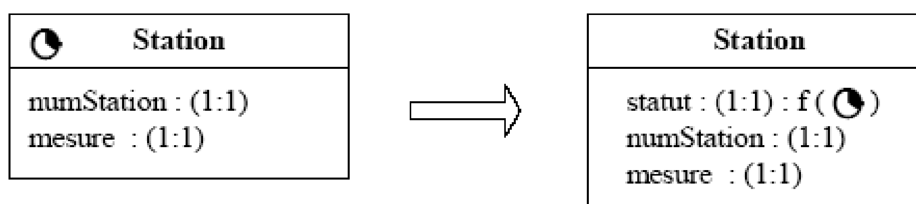

Illustration 13 : Passage de l'élément temporel comme attribut

2. L'attribut temporel est transformé en attribut complexe multivalué. Le premier sous-attribut détermine s'il s'agit du type originel « instant » ou du type « intervalle », le second sous attribut « valeur » détermine l'état de l'objet au moment du premier (prévu, actif, suspendu ou détruit). Dans la cas où le premier sous-attribut serait de type « instant », la transformation s'arrête et celui-ci prend la valeur de l'instant.

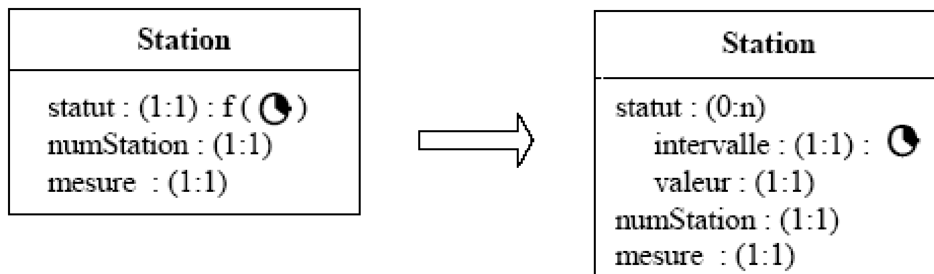

Illustration 14 : Décomposition de l'attribut temporel en attribut complexe multivalué

3. Dans le cas d'un intervalle, ce sous-attribut est à nouveau décomposé en un attribut complexe comprenant deux champs: le début et la fin de l'interval.

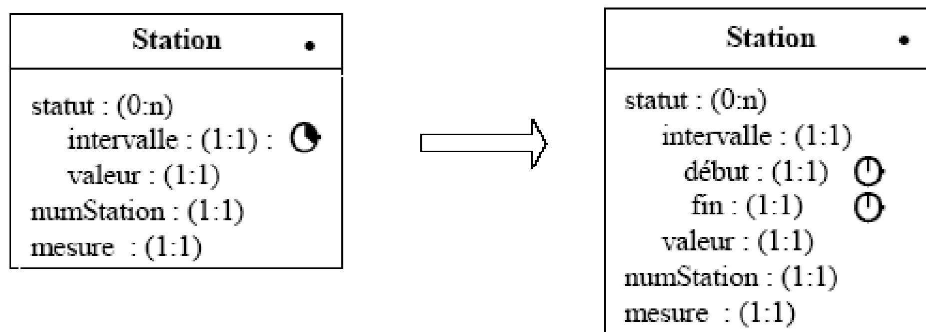

Illustration 15 : Décomposition de l'interval en attribut complexe

L'ensemble de ces transformations s'accompagne de contraintes d'intégrité sur les valeurs prises par les différents attributs.

Après ces transformations du point de vue temporel, viennent les transformations sur les attributs complexes et les attributs mutli-valués. Pour les

SGDB orientés objets, le schéma MADSLog traduit ces deux types de manière identique: il crée un nouveau domaine. Pour la transformation d'un élément temporel « instant », le résultat final sera la création de 2 domaines, contre 3 domaines pour les éléments « intervalle ». Dans le cas de set d'interval ou d'instant, il faudra rajouter un domaine supplémentaire, le set étant assimilable à un attribut complexe.

3.2.2.2 Les objets en MADSLog

Maintenant que la traduction des schémas pour le temporel est connue, la traduction des requêtes peut commencer. A chaque fois que le traducteur rencontre un attribut, un objet ou une relation dépendant du temps dans l'expression algébrique, il va chercher son équivalent dans le schéma MADSLog et faire le remplacement dans l'expression algébrique. Ce travail s'effectue de manière totalement séquentielle puisque le schéma final est connu et ne risque pas de subir de nouvelles traductions.

3.2.2.3 Les fonction en MADSLog

Pour les fonctions utilisant la notion temporelle, la démarche est un peu différente. Ici, impossible d'élaborer une traduction universelle pour l'ensemble des fonctions. Chaque type de fonction va avoir sa propre méthode de traduction. Les fonctions temporelles utilisée dans l'algèbre MADS sont inspirées des opérateurs de James Allen (1984) [biblio. 5] qui diffèrent dans certains cas des opérateurs que l'on peut trouver dans le langage TSQL2[4].

Le problème vient des interactions entre les différents attributs temporels et/ou entre un attribut temporel et une valeur temporelle. Examinons les différents cas utilisés par les fonctions :

1. **Comparaison entre deux instants :** dans ce cas de figure, le schéma MADSLog déployé est suffisant pour effectuer une comparaison directe, la fonction peut être remplacée par la comparaison sans autre modification.

4 Temporal Structured Query Language, version 2

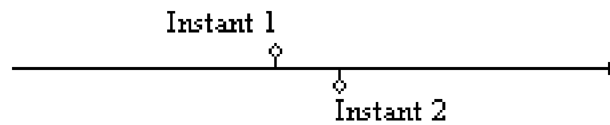

Illustration 16 : Deux instants sur la ligne du temps

2. **Comparaison entre un instant et un intervalle :** l'intervalle n'étant plus considéré comme une attribut simple, il faut décomposer la comparaison en deux étapes. La fonction devient alors une comparaison entre le début de l'intervalle et l'instant suivie d'une comparaison entre la fin de l'intervalle et l'instant. De plus, ces comparaisons doivent être liées par un « ET » ou un « OU » logique, suivant les cas.

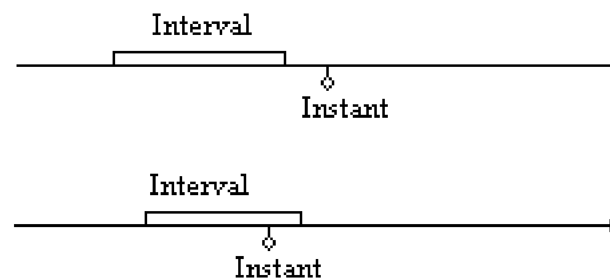

Illustration 17 : Différentes possibilités de comparaison entre un instant et un intervalle

3. **Comparaison entre deux intervalles :** la comparaison d'intervalles se basant sur leurs débuts et leurs fins, il faut traduire l'opérateur par quatre comparaisons successives. En effet, soit l'intervalle A et l'intervalle B, il faudra comparer (A.début, B.début), (A.début, B.fin), (A.fin, B.début), (A.fin, B.fin). L'ensemble sera toujours lié pare des « ET » / « OU » logiques.

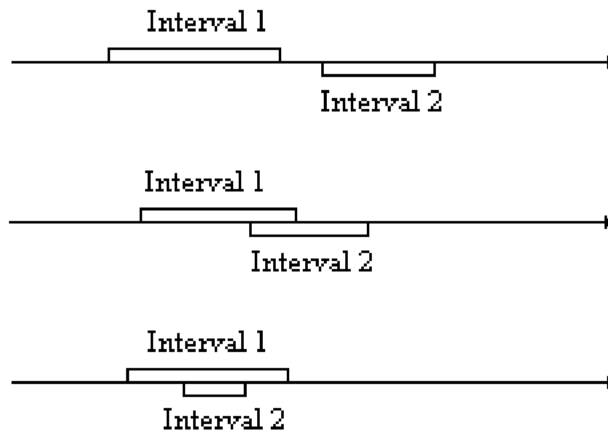

Illustration 18 : Différentes possibilités de d'agencement des intervalles

4. **Comparaison d'un intervalle avec une durée :** Pour ce cas, il faudra effectuer une différence entre la fin de l'intervalle et son début, le résultat obtenu nous donne une durée qui peut être comparée avec la première.

Dans l'ensemble des traductions envisagées, certaines peuvent en outre être optimisées. Par exemple, la traduction d'un fonction demandant de vérifier si un instant précède un intervalle ne nécessitera pas 2 comparaisons, le simple fait de savoir que l'instant précède ou non le début de l'intervalle suffit à valider ou à invalider la condition. Les déploiements indiqués plus haut représentent un traitement générique mais non optimal.

Exemple de traduction : Prenons le schéma MADS suivant:

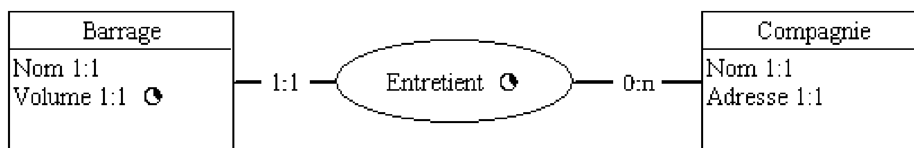

Illustration 19 : Exemple de schéma MADS

Après le passage dans le traducteur de schéma pour SGDB orienté objet, et en supposant que les éléments temporels soient de type instant, nous obtenons un

schéma MADSLog équivalent.

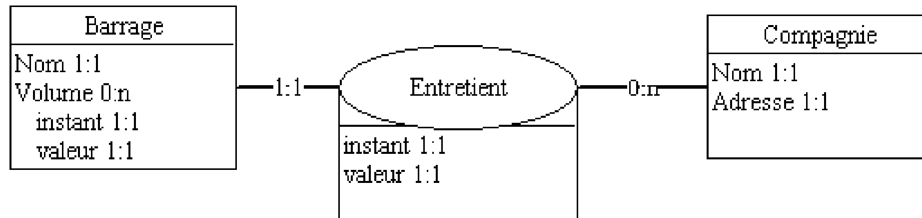

Illustration 20 : Traduction du schéma en MADSLog

Une requête demandant de sortir la liste des compagnies ayant travaillé sur les barages qui ont eu un volume supérieur à 80.000 litre pendant l'an 2000 passe de l'expression algébrique

$$selection\,[\exists e \in Entretient | \exists b \in e.Barrage\; b.volume.attime(2000) > 80.000\,]\,Compagnie\; C$$

à l'expression algébrique MADSLog

$$selection\,[\exists e \in Entretient | \exists b \in e.Barrage\; b.volume.instant > (31/12/1999) \wedge b.volume.instant \leq (31/12/2000) \wedge b.volume.valeur > 80.000\,]\,Compagnie\; C$$

3.3 De MADSLog au SGDB Oracle 9i

Une fois l'expression algébrique MADLog obtenue, il n'est plus nécessaire de connaître l'intégralité du schéma, l'ensemble des objets nécessaires lors de la transcription de la requête est indiqué dans celle-ci. Le passage au script SQL n'est cependant pas immédiat en raison des règles d'écriture du langage.

3.3.1 Structure d'une requête SQL

Une requête SQL peut se décomposer en 3 blocs distincts, ordonnés comme suit:

Opérateur , Opérande+ , Prédicats*

En comparaison de l'expression algébrique, on peut remarquer que si l'opérateur occupe le même emplacement syntaxique, ce n'est pas le cas pour les opérandes et les prédicats.

3.3.2 Recherche des Opérandes

Il existe en réalité trois types d'opérandes, plus ou moins aisément transformable dans la requête SQL:

- Certains opérandes sont facilement repérables, il s'agit des opérandes principaux sur lesquels porte la requête originale. Ceux-ci sont placés en fin d'expression algébrique et il est facile d'effectuer une permutation pour les placer directement derrière l'opérateur. Dans l'expression ci-dessous, « Compagnie » est un opérande principal.

$$selection\,[\exists e \in Entretient | \exists b \in e.Barrage\; b.volume.attime(2000) > 80.000]\, Compagnie\, C$$

- Une deuxième catégorie d'opérandes est aussi facile à identifier et replacer, ce sont les variables. On appelle « variable » tout ensemble d'occurrence d'un objet utilisé, cette ensemble est accessible grâce à un alias. Dans l'expression suivante, « e » est l'alias de la variable « Entretient » et « b » est l'alias de la variable « e.Barrage »

$$selection\,[\exists e \in Entretient | \exists b \in e.Barrage\; b.volume.attime(2000) > 80.000]\, Compagnie\, C$$

  Comme les variable apparaissent en début d'expression, il est facile de les placer comme opérandes dans le script SQL.

- La dernière catégorie d'opérandes est plus difficile à retranscrire. Ceux-ci sont disséminés dans le reste de l'expression, sans ordre spécifique et peuvent apparaître plusieurs fois. Ces opérandes sont des domaines, des objets et des attributs auxquelles il est possible d'accéder directement dans l'expression algébrique, mais qui nécessitent la création de tables temporaires, et donc d'opérandes, au sein d'un script SQL. Dans l'expression suivante, « volume » est un domaine, repris deux fois, qui servira d'opérandes dans la requête SQL.

$$selection\,[\exists e \in Entretient | \exists b \in e.Barrage\; b.volume.instant > (31/12/1999) \\ \wedge b.volume.instant \leq (31/12/2000) \wedge b.volume.valeur > 80.000]\, Compagnie\, C$$

  Pour replacer ces opérandes à leur place et éviter les doublons, il faut mettre en oeuvre un mécanisme de détection, de stockage, de création d'alias et de replacement en fin de traitement. Ce mécanisme sera expliqué plus en détail dans l'implémentation du transcripteur.

3.3.3 Transcription des prédicats

Ces différentes récupérations d'opérandes mises en place, la transcription des prédicats peut réellement commencer. A ce niveau de la transcription, la syntaxe de l'expression algébrique est assez proche de la syntaxe SQL et ne pose pas de difficulté supplémentaire du point de vue conceptuel. Le chapitre 5.2 sur la transcription montrera les difficultés rencontrées lors de l'implémentation

Chapitre 4 Structures des données et Outils utilisés

4.1 XML

4.1.1 Description

Le XML[5] décrit une classe d'objets de données appelés documents XML et décrit partiellement le comportement des programmes qui les traitent. XML est une forme restreinte de SGML[6], le langage normalisé de balisage généralisé créé au milieu des années '60 et standardisé en 1986 par l'organisme ISO[7]. Par construction, les documents XML sont des documents conformes à SGML.

Les documents XML se composent d'unités de stockage appelées entités, qui contiennent des données analysables ou non. Les données analysables se composent de caractères, certains formant les données textuelles, et le reste formant le balisage. Le balisage décrit les structures logique et de stockage du document. XML fournit un mécanisme pour imposer des contraintes à ces structures (cf. DTD [4.1.4]).

Un module logiciel appelé processeur XML est utilisé pour lire les documents XML et pour accéder à leur contenu et à leur structure. On suppose qu'un processeur XML effectue son travail pour le compte d'un autre module, appelé l'application. Cette spécification décrit le comportement requis d'un processeur XML,

5 Extensible Markup Language

6 Standard Generalized Markup Language

7 International Organization for Standardization

c'est à dire la manière dont il doit lire des données XML et les informations qu'il doit fournir à l'application.

4.1.2 Origine

XML a été développé par un groupe de travail (GT) XML constitué sous les auspices du Consortium du World Wide Web (W3C) en 1996.

Les objectifs poursuivis lors de la conception de XML furent les suivants :

- XML devrait pouvoir être utilisé sans difficulté sur Internet
- XML devrait soutenir une grande variété d'applications
- XML devrait être compatible avec SGML
- Il devrait être facile d'écrire des programmes traitant les documents XML
- Le nombre d'options dans XML devrait être réduit au minimum, idéalement à aucune
- Les documents XML devraient être lisibles par l'homme et raisonnablement clairs
- La conception de XML devrait être préparée rapidement
- La conception de XML serait formelle et concise
- Il devrait être facile de créer des documents XML
- La concision dans le balisage de XML serait de peu d'importance.

4.1.3 Syntaxe

Un objet de données est un document XML s'il est bien formé, tel que précisé dans les spécifications. De plus, un document XML bien formé peut être valide s'il obéit à certaines autres contraintes.

Chaque document XML a une structure logique et une structure physique. Physiquement, le document se compose d'unités appelées entités. Une entité peut appeler d'autres entités pour causer leur inclusion dans le document. Un document commence à la « racine » ou entité document. Logiquement, le document se compose de déclarations, d'éléments, de commentaires, d'appels de caractère et d'instructions de traitement, qui sont indiqués dans le document par du balisage

explicite.

Les éléments de syntaxe détaillés ci-dessous ne comprennent pas l'ensemble des possibilités du XML mais uniquement les éléments utilisés dans le projet. L'ensemble des spécifications est disponible à l'adresse suivante: http://www.w3.org/TR/REC-xml/

4.1.3.1 Données textuelles et balisage

Le texte se compose de données textuelles et de balisage. Le balisage prend la forme de balises ouvrantes, de balises fermantes, de balises d'éléments vides, d'appels d'entité, d'appels de caractère, de commentaires, de délimiteurs de section CDATA, de déclarations de type de document, et d'instructions de traitement.

Tout le texte qui n'est pas du balisage constitue les données textuelles du document. Il est à noter que du texte correspondant au sous-ensemble {#x20, #x9, #xD, #xA} (en notation hexadécimale) constitue du balisage et non pas des données textuelles (ce sont les sauts de ligne, retours charriot, ...).

Les caractères esperluète (&) et crochet gauche (<) peuvent apparaître sous leur forme littérale seulement quand ils sont utilisés comme délimiteurs de balisage ou dans un commentaire, une instruction de traitement (non abordée ici), ou une section CDATA (non abordée ici). S'ils sont nécessaires ailleurs, ils doivent être remplacés par des appels de caractères numériques ou en utilisant les chaînes de caractères « & » et « < » respectivement. Le crochet droit (>) peut être représenté en utilisant la chaîne de caractères « > » et doit, pour compatibilité, être remplacé par « > » ou un appel de caractère quand il apparaît dans la chaîne de caractères «]]> » dans du contenu, quand cette chaîne ne marque pas la fin d'une section CDATA.

Dans le contenu des éléments, toute chaîne de caractères ne contenant pas un délimiteur de début de balisage est considérée comme donnée textuelle. Dans une section CDATA, toute chaîne de caractères ne contenant pas le délimiteur de fin de section CDATA, «]]> », est également considérée comme donnée textuelle.

L'apostrophe (') peut être représentée par « ' », et le caractère guillemet

anglais (") par « " », afin de permettre à des valeurs d'attribut de contenir ces caractères.

4.1.3.2 Commentaires

Les commentaires peuvent apparaître n'importe où dans un document en dehors d'autre balisage ; de plus, ils peuvent apparaître dans la déclaration de type de document aux endroits permis par la grammaire. Ils ne font pas partie des données textuelles du document ; un processeur XML peut permettre à une application de récupérer le texte des commentaires. À des fins de compatibilité, la chaîne « -- » (double trait d'union) ne doit pas apparaître à l'intérieur de commentaires.

Règle de production d'un commentaire :

```
commentaire ::= '<!--' ((Car - '-') | ('-' (Car - '-')))* '-->'
```

où « car » correspond à l'ensemble des caractères disponibles

4.1.3.3 Traitement du blanc et des fins de ligne

En éditant des documents XML, il est souvent commode d'indenter (via des espaces, des tabulations et des interlignes) pour distinguer le balisage pour une plus grande lisibilité. Du tel « blanc » n'est pas typiquement destiné à être inclus dans la version livrée du document. D'autre part, le blanc « significatif » qui devrait être préservé dans la version livrée est courant, par exemple en poésie et en code source.

Un processeur XML doit toujours transmettre à l'application tous les caractères d'un document qui ne sont pas du balisage. Un processeur XML validateur doit également informer l'application des caractères qui constituent le blanc apparaissant dans du contenu élémentaire pur.

Un attribut spécial nommé « xml:space » peut être associé à un élément pour signaler l'intention que dans cet élément, le blanc soit préservé par les applications. Dans les documents valides, cet attribut, comme tout autre, doit être déclaré s'il est

utilisé. Si déclaré, il doit être donné comme type énuméré dont les seules valeurs possibles sont « default » et « preserve ». Par exemple :

```
<!ATTLIST poème   xml:space (default|preserve) 'preserve'>
```

La valeur « default » indique que les modes implicites de traitement du blanc sont acceptables pour cet élément ; la valeur « preserve » demande que les applications préservent tout le blanc. Cette intention déclarée s'applique à tous les éléments à l'intérieur du contenu de l'élément porteur de la déclaration (voir DTD), à moins qu'elle ne soit annulée par une autre apparition de l'attribut de « xml:space ».

L'élément racine de n'importe quel document est supposé n'avoir indiqué aucune intention en ce qui concerne le traitement du blanc, à moins qu'il ne fournisse une valeur pour cet attribut ou que l'attribut ne soit déclaré avec une valeur implicite.

Des entités XML analysables sont souvent enregistrées dans des fichiers qui, pour la commodité d'édition, sont organisés en lignes. Ces lignes sont typiquement séparées par une combinaison du caractère retour chariot (#xD) et retour à la ligne (#xA).

Pour simplifier la tâche des applications, quand une entité externe analysable ou une valeur littérale d'entité d'une entité analysable interne contient soit la séquence de deux caractères littéraux « #xD#xA » soit un littéral #xD, un processeur XML doit transmettre à l'application le seul caractère #xA. (Ce comportement peut simplement être produit en normalisant toutes les bris de ligne à #xA à l'entrée, avant l'analyse.)

4.1.4 DTD

La DTD[8] contient ou désigne des déclarations de balisage qui fournissent une grammaire pour une classe de documents. Cette grammaire est connue comme « déclaration de type de document ». La DTD peut désigner un sous-ensemble externe (un genre spécial d'entités externes) contenant des déclarations de

8 Document Type Declaration

balisage, elle peut contenir des déclarations de balisage directement dans un sous-ensemble interne ou peut faire les deux. La DTD d'un document se compose des deux sous-ensembles regroupés. Une DTD peut être déclarée au sein du fichier XML ou via une référence à un fichier extérieur.

La grammaire décrite par une DTD contient obligatoirement la liste des balises utilisable au sein d'un document, leurs attributs (avec indication sur le type: requis, implicite, ...), leurs possibilités d'enchassement (sous forme d'expression régulières). Elle peut également contenir des informations concernant le traitement des données.

4.1.5 Représentation XML des données MADS et MADSLog

La représentation des requêtes au format XML se rapproche fortement de leur écriture en expression algébrique. On retrouve donc l'opérateur suivi de l'opérande et, au sein de ceux-ci, les divers éléments de l'expression originale. La gestion du contenu, lui, n'est pas aussi bien structuré. Certains objets se voient entièrement définis par les attributs de certaines balises et se contentent donc d'une balise unique, d'autres utilisent des données entourées par une balise ouvrante et une balise fermante. Le choix de telles différences est discutable car il oblige à traiter des contenus similaires de façon différentes. Pour exemple, les opérateurs de comparaisons sont scindé en deux, l'opérateur « égal » qui utilise une balise, et les opérateurs « <, <=, >, => » qui utilisent une autre balise dotée d'un attribut pour préciser l'opérateur.

4.2 Java

4.2.1 Origine

Java est un langage objet mis au point en 1991 par la firme Sun Microsystems. Le but de Java, à l'époque, était de constituer un langage de programmation pouvant être intégré dans les appareils électroménagers, afin de pouvoir les

contrôler, de les rendre interactifs, et surtout de permettre une communication entre les appareils. Ce programme de développement se situait dans un projet appelé Green, visant à créer une télécommande universelle (Star 7) comprenant un système d'exploitation capable de gérer l'ensemble des appareils électroménagers de la maison. Etant donné que le langage C++ comportait trop de difficultés, James Gosling, un des acteurs du projet (considéré désormais comme le père de Java) décida de créer un langage orienté objet reprenant les caractéristiques principales du C++, en éliminant ses points difficiles, et en le rendant moins encombrant et plus portable (il devait pouvoir être intégré dans n'importe quel appareil...). Ainsi, ce langage fut baptisé dans un premier temps Oak (Oak signifiant chêne). Toutefois, comme ce nom était déjà utilisé, il fut rebaptisé Java en l'honneur de la boisson préférée des programmeurs, c'est-à-dire le café, dont une partie de la production provient de l'île Java.

A la même époque, le Web faisait son apparition, or Java possédait toutes les caractéristiques faisant de lui un langage approprié pour le Web:

- Le réseau des réseaux rassemblant sur une même structure des machines différentes, il fallait un langage capable de fonctionner sur chacune d'entre-elles: Java était conçu pour être portable.
- Le web était limité en bande passante: Java était conçu pour être petit.

Ainsi, en 1994, l'équipe décida de mettre au point un navigateur (baptisé HotJava) intégrant Java et capable de faire fonctionner des applets (des petites applications fonctionnant dans un navigateur). Fin 1995 Java eut un terrible essor avec le soutien de Netscape, qui ne tarda pas à inclure Java dans son navigateur...

Java a évolué et il est depuis lors reconnu comme l'un des langages majeurs. Sa dernière version standardisée utilisée pour notre projet est la version 1.4.2, la version 1.5 étant en cours d'élaboration.

4.2.2 Spécificités du langage

D'un point de vue syntaxique, Java est très proche du C++ car il en est assez

directement inspiré. Cependant, du point de vue des concepts, de nombreuses choses diffèrent.

4.2.2.1 Orienté Objet

Java est un langage qui se veut orienté objet. Une application Java est constituée d'une série de *packages*, regroupement d'objets. Chaque objet est représenté par une classe contenant les attributs et les méthodes de cet objet. La « philosophie » Java veut que l'on utilise au maximum l'encapsulation des données au sein des classes.

L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés. L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet. Java possède trois mode de déclaration: public, protégé et privé, dont le fonctionnement est identique aux déclarations C++.

L'héritage est le concept où Java diffère principalement de C++ dans sa vision de l'orienté objet. Java ne permet pas d'héritage multiple (source de nombreuses erreurs de conceptions en C++), et pour simuler un tel mécanisme il faut passer par le concept d' « interface » qui se borne à définir l'ensemble des déclarations de fonction. En Java, excepté les types atomiques (int, char, float, ...), tout est objet, et tous les objets descendent d'une même super-classe abstraite « Object ».

Java n'accepte pas de surcharge d'opérateur, ceux-ci sont réservés au types atomiques et à la classe String (seule classe qui surcharge l'opérateur). De même, aucun trans-typage ne peut être effectué de manière automatique, tout doit être spécifié afin d'éviter tout ambiguïté, quitte à forcer un typage vers la super-classe « Object ».

4.2.2.2 Absence de pointeurs

Java se voulant simple à programmer, portable et sûr, la notion de pointeur explicite est inexistante. Elle est présente de manière implicite lors de l'instanciation d'objets, mais jamais le programmeur ne peut manipuler un pointeur. En effet, d'une

part, la notion de pointeur est, dans un certain sens, opposée à la notion d'orienté objet, et, d'autre part, java étant destiné à diverses plates-formes sans aucune modification du code, il était impératif d'empêcher un programmeur d'accéder directement à une zone mémoire particulière via les pointeurs.

4.2.2.3 Garbage collector

L'un des concepts les plus controversés de Java est sans doute son *garbage collector*. Celui-ci à pour but de récupérer l'espace mémoire occupé par des objets qui ont cessé d'être accessible. En C++, le programmeur a la responsabilité de la gestion de la mémoire, c'est lui qui va définir les « destructeurs » qui vont supprimer les objets inutilisés, et il lui est également possible de définir quand ceux-ci doivent se déclencher. Pour Java, le fonctionnement est tout autre, il n'existe pas de destructeur et c'est la machine virtuelle qui s'occupe de la gestion de la mémoire. Aussi, d'une machine virtuelle à une autre, le fonctionnement peut être totalement différent, l'un pourrait libérer la mémoire allouée à un objet dès que celui-ci cesse d'être référencé, l'autre ne pourrait libérer de la mémoire que lorsqu'un nouvel objet ne trouve pas suffisamment de place pour être instancié. Les deux méthodes ont leurs avantages et leurs inconvénients qui ne seront pas détaillés ici, mais la manière dont elles sont implémentées dans la machine virtuelle joue également beaucoup sur les performances d'une application Java.

4.2.2.4 Gestion des erreurs

Java a été pensé pour des développements rapides mais sûrs, il intègre donc un mécanisme de gestion des erreurs. Toute méthode sensible ou routine de traitement pouvant inclure des fonctions susceptibles de retourner des erreurs doit être entourée d'un mécanisme de gestion de celle-ci. La gestion des erreurs est en partie laissée libre au programmeur, celui-ci peut décider de les traiter « à la source » lorsque celles-ci se produisent, ou bien les laisser remonter dans l'arborescence du programme pour un traitement plus global. Il peut également décider d'ignorer les erreurs et continuer l'application sans en tenir compte. Cette dernière méthode est à éviter autant que possible car elle ne permet pas de garantir le bon fonctionnement d'un programme.

4.2.2.5 Bytecode

L'un des principaux objectifs lors de l'élaboration du langage étant sa portabilité. Pour garantir celle-ci, trois possibilités étaient envisageables:

- La compilation complète des sources sur chacune des plates-formes existantes. Si ce procédé offre les meilleurs performances en terme d'exécution du programme, c'est également le plus complexe à mettre en oeuvre pour l'utilisateur. De plus, cette méthode obligerait les développeurs à divulguer leur code source en clair, ce que beaucoup se refusent à faire, ou à compiler eux-même leur code pour toutes les plates-formes existantes. Cette solution a donc été écartée.
- L'interprétation des sources à l'exécution. Cette méthode est simple à utiliser par l'utilisateur car il lui suffit de disposer d'un *parser* Java pour exécuter les divers programmes à sa disposition. Malheureusement, avec un tel procédé, il est quasiment impossible pour la machine de procéder à quelque optimisation du code et les performances peuvent en pâtir lourdement. En outre, le code est également visible par tous. Cette solution à également été écartée.
- La troisième solution consiste en un subtile mélange des deux autres. Le code source est pré-compilé pour fournir un langage intermédiaire, le *bytecode*. Lors de cette étape, il est déjà possible de fixer définitivement certaines optimisations. Ensuite, chaque plate-forme dispose d'une machine virtuelle Java (JVM) qui transforme le *bytecode* en instructions machines. Ce procédé permet d'obtenir des performances bien plus élevée qu'avec une interprétation classique, mais reste en deçà des performances atteintes par une compilation dédiée à la plate-forme.

Le *bytecode* offre également la possibilité au développeur de cacher (ou offusquer) le code source s'il le désire. Dans le cas contraire, un décompilateur permettra de retrouver l'intégralité des sources du programme, commentaires compris.

4.3 JAXP

JAXP[9] est un regroupement de 2 API[10] spécialisées dans le traitement des données XML. Initialement, ces packages n'était pas directement inclus dans la JDK, mais il font désormais partie de la distribution standard.

4.3.1 Le modèle SAX

La première API fournie pour traiter les fichier XML est SAX[11], elle a été définie par le groupe « XML.org ». Elle se base sur un modèle événementiel, à chaque élément syntaxique lut, une méthode est déclenchée. L'ensemble des méthodes déclenchées est définit via des interfaces prédéterminées.

Cette API se prête particulièrement bien à la création d'une structure parallèle grâce aux données récoltées dans le fichier XML au fur et à mesure de sa lecture. Par contre, la ré-écriture de fichiers XML est plus contraignante dans ce modèle.

4.3.2 Le modèle DOM

La seconde API se base sur le modèle DOM[12]. Elle fut proposée par le W3C[13], cette API est basée sur un modèle objets prédéterminé. Cette fois-ci, le document va d'abord être complètement traité avant de pouvoir le manipuler. Le but de cette API est de générer un arbre d'éléments représentant parfaitement le document XML originel. Via cette API, il est également possible de travailler directement au niveau de l'arborescence d'objets pour générer un nouveau document XML.

Cette API, codée à partir de l'API SAX est plus gourmande en terme de ressources utilisées mais elle permet, en contre-partie, de manipuler facilement la structure même des fichiers XML, ce qui s'avèrera utile lors de la traduction MADS vers MADSLog.

9 Java API for XML Parsing

10 Application and Programming Interface

11 Simple API for XML

12 Document Object Model

13 World WideWeb Consortium

Chapitre 5 Méthodes utilisées

5.1 De MADS à MADSLog

Cette partie de l'implémentation consite en l'ajout des fonctions de traduction des données temporelles au sein d'une application déjà existante. Celle-ci est programmée en Java et utilise la méthode DOM de JAXP pour accéder aux données, les manipuler et recréer un fichier de sortie XML conforme à l'algèbre MADSLog pour les SGDB relationnels objets.

5.1.1 Organisation des données

Les requêtes MADS et MADSLog sont stockées dans des fichiers XML. La structure de ces fichiers est quasiment identique à la structure de l'expression algébrique. Les schémas MADS et MADSLog sont également stockés au format XML. Le code qui suit représente, de manière très globale, l'enchassement des divers éléments de l'expression algébrique dans le fichier XML :

```
<prologue XML />
<expression algébrique>
   <opérateur>
      <variable> //facultatif
         description de la variable
      </variable>
      ...
      <formule>
         <predicat>
            description du prédicat
         </prédicat>
         ...
      </formule>
   </opérateur>
   <opérande>
      description de l'opérande
   </opérande>
   <opérande> // uniquement en cas d'opérateur binaire
      description de l'opérande
   </opérande>
</expression algébrique>
```

5.1.2 Structure du programme

La structure globale du traducteur est très simple : après que la bibliothèque JAXP ait créé un arbre via son API DOM, cet arbre est parcouru séquentiellement et chaque noeud est analysé et éventuellement transformé pour être conforme aux capacités des SGDB orientés objets.

5.1.3 Accès aux données et modifications

Lorsque le traducteur rencontre un prédicat qui contient des élements temporelles, il lance la routine de traîtement correspondant. Toutes ces routines peuvent s'inscrire dans un même canevas:

```
routine(predicat) {
   terme <- extraire_premier_terme()
   terme2 <- extraire_second_terme()
   varia <- trouver_type_de_varia_du_temps(terme)
   si varia = interval {
      MLtermeD <- créer_struct_MADSLog(terme, début)
      MLtermeF <- créer_struct_MADSLog(terme, fin)
      varia2 <- trouver_type_de_varia_du_temps(terme2)
      si varia2 = interval {
         MLtermeD2 <- créer_struct_MADSLog(terme2,début)
         MLtermeF2 <- créer_struct_MADSLog(terme2,fin)
         NewPredicat <- créer_predicat(MLtermeD,
MltermeF, MltermeD2, MLtermeF2)
      }
      sinon { // cas continu ou discret
         MLterme2 <- créer_struct_MADSLog(terme2)
         NewPredicat <- créer_predicat(MLtermeD,
MltermeF, Mlterme2)
      }
   }
   sinon { // cas continu ou discret
      MLterme <- créer_struct_MADSLog(terme)
      varia2 <- trouver_type_de_varia_du_temps(terme2)
      si varia2 = interval {
         MLtermeD2 <- créer_struct_MADSLog(terme2, début)
         MLtermeF2 <- créer_struct_MADSLog(terme2, fin)
         NewPredicat <- créer_predicat(MLterme,
MltermeD2, MLtermeF2)
      }
      sinon { // cas continu ou discret
         MLterme2 <- créer_struct_MADSLog(terme2)
         NewPredicat <- créer_predicat(MLterme, Mlterme2)
      }
   }
   predicat <- newPredicat
}
```

Dans ce squellette générique, seule la fonction « créer_prédicat » varie suivant

l'opérateur utilisé dans le prédicat. La fonction « créer_struct_MADSLog » utilise le schéma MADSLog de la base de données pour trouver comment le terme comprenant la temporalité a été traduit et créer ainsi une nouvelle structure qui permettra d'y accéder. Dans le cas des intervalle, la structure ayant dû être séparée en deux éléments (début et fin), ce sont deux structures qui doivent être créées.

5.2 De MADSLog à Oracle 9i

Un transcripteur pour Oracle existait déjà mais il ne prenait pas en compte les données temporelles tout comme d'autres aspects du projet MurMur. Il fut décidé de réécrire un transcripteur capable de gérer cet aspect temporel.

Plus que de construire un transcripteur spécifique pour Oracle, l'enjeu de la réécriture du code était également de trouver un squelette léger et réutilisable pour l'écriture d'autres transcripteurs à moindre coût. La solution proposée ici se base sur l'observation de la structure des fichiers XML de requêtes du modèle MADSLog pour les SGDB relationnel objet. Cependant, la structure de ces fichiers différant peu des fichiers issus du modèle MADS, on peut supposer que cette solution sera portable dans les autres modèles MADSLog.

Ce transcripteur a été écrit en Java avec l'utilisation de l'API SAX de JAXP.

5.2.1 Organisation des données

Les données reçues proviennent du traducteur de requête MADSLog, elles sont donc organisée dans un fichier XML. Ce fichier XML est validé par une DTD qui possède une caractéristique intéressante dans la définition du balisage utilisé pour l'expression régulière. En effet, aucune balise ne peut s'inclure directement dans elle-même. Ainsi, le code suivant ne peut jamais être observé:

```
<balise1>
   <balise1>
      ... // autres balises éventuelles
   </balise1>
   ... // autres balises éventuelles
</balise1>
```

Cette propriété est à retenir car elle va nous permettre de simplifier le traitement des données.

Une autre propriété inhérante au XML est son système d'enchassement. Ce système peut s'apparenter au parenthésage utilisé dans l'algèbre pour forcer la priorité de certains opérateur. Là où le XML diffère de l'algèbre classique, c'est dans le fait que ces « parenthèses » sont nommées, et on peut donc connaître, en regardant l'une des deux parenthèses quel opérateur est appliqué en son sein.
Si l'on ne considère que la dernière parenthèse, on peut alors faire le parallélisme avec un autre système de notation, la NPI.

Ces deux constatations permettent de généraliser les traitements des balises et de l'information utilisant un algorithme de NPI étendu comme nous le verrons en [5.2.3.2].

5.2.2 Principe de la NPI

La NPI, ou Notation Polonaise Inverse est une méthode de calcul à base de pile, chère aux amateurs de calculettes HP's. Le principe est simple, au lieu d'introduire classiquement un calcul comme « 7+3 », l'utilisateur va rentrer les données comme suit: « 7;3;+ ». Chaque donnée entrée qui ne fait pas partie de la liste des opérateurs est placée dans une pile, et lorsqu'un opérateur est rencontré, il prend les n premiers éléments de la pile nécessaires à son opération et place le résultat au sommet de la pile. Une telle méthode de calcul rend caduque l'utilisation de parenthèses. En effet, ce qui, par l'écriture traditionnelle, s'écrit « (3+7)*2 » pour obtenir 20 se notera « 3;7;+;2;* » en NPI.

5.2.3 Extensions de la NPI

Avant toute chose, il convient de poser la définition de certains termes utilisés dans cette section car ils diffèrent quelque peu de l'usage qui en est fait dans le reste de ce document.

5.2.3.1 Définitions

- **Opérateur :** toute balise ou paire de balises. Une balise unique <alpha /> équivaut à l'aggrégation de la balise ouvrante <alpha> et de la balise fermante </alpha>. Elle joue donc les deux rôles. L'élément déclencheur d'un opérateur est sa balise fermante.
- **Terme :** tout résultat d'un opérateur ou toute donnée non traitée.
- **Donnée :** tout attribut d'une balise ou tout contenu textuel compris entre une paire de balises.
- **Délimiteur :** tout marqueur indiquant la nature d'un opérateur déjà traité.

5.2.3.2 Opérateurs n-aires

Dans notre cas, le principe de la NPI simple ne suffit pas. La NPI simple souffre d'un défaut, il lui faut connaître le nombre exact de termes nécessaires à chaque opérateur. Dans le Modèle MADSLog relationnel objet, certains opérateurs portent sur un nombre indéfini de termes. C'est le cas, par exemple, de l'opérateur « selection » qui peut accepter un nombre indéfini de termes, les variables de l'expression algébrique.

Pour résoudre ce problème, nous allons faire intervenir la première propriété de la structuration des données. Grâce à cette impossiblité d'inclusion directe d'une même balise, il est possible de créer un mécanisme qui va permettre à un opérateur n-aire de détecter quand il a récolté l'ensemble de ses termes. La solution adoptée est l'utilisation de délimiteurs. Pour chaque opérateur réalisé, on pose dans une pile un délimiteur qui indique son type. Quand un opérateur utilise des termes résultant d'opérateurs précédents, il retire de la pile les délimiteurs correspondants et y pose le sien. Avec ce procédé, il est possible d'introduire des notions d'arrêt pour les opérateurs n-aires en fonction des délimiteurs rencontrés.

En quoi la propriété pré-citée de la DTD permet-elle celà? Imaginons le cas suivant:

```
<op n-aire>
   données
</op n-aire>
<op n-aire>
   <op-naire>
      données
   </op n-aire>
</op n-aire>
```

Si une telle structure était possible, les délimiteurs ne pourraient pas fonctionner correctement et on aurait une série d'erreurs en cascades. Démonstration :

1. lorsque le premier opérateur se déclenche, il recherche des termes jusqu'à la racine, les traite et place son délimiteur dans la pile
2. le deuxième opérateur à se déclencher est en réalité le troisième. Il recherche des termes et va remonter jusqu'au délimiteur placé dans la pile. Là, il ne sait pas s'il doit s'arrêter (ce qui serait juste) ou continuer. Supposons qu'il continue. Il avale dont une donnée qui ne lui appartient pas, la traite, enlève le délimiteur précédent et place le sien.
3. Le dernier opérateur se déclenche alors. Comme il dispose de la même routine de traitement, il va donc remonter jusqu'au sommet, traiter des données erronnées, enlever le délimiteur et placer le sien.

Si au contraire, l'étape 2 s'était passée correctement, l'étape 3 aurait posé le problème inverse. L'interdiction de ce genre d'inclusion permet bien d'éviter ce genre de cas limites.

5.2.3.3 Récupération des opérandes

Un autre obstacle à l'utilisation d'un algorithme de NPI classique provient des impératifs de la grammaire SQL. Pour rappel, une requête SQL est formée comme suit:

Opérateur , Opérande+ , Predicats*

Les opérandes dispersées dans l'ensemble de l'expression algébrique doivent donc être regroupée en un bloc qui suit l'opérateur.

Deux types d'opérandes ne posent aucun problème, l'opérande principal (ou les opérandes principaux en cas d'opérateur binaire) et les variables, toujours déclarées en début d'expression. Reste, comme indiqué dans le chapitre 3, les objets et autres domaines introduits dans l'expression lors du passage au format MADSLog.

Pour placer correctement ces opérandes dans la requête SQL, un double mécanisme est nécessaire. Il faut, bien évidemment, récolter et stocker ces opérandes pour les réinjecter dans la requête en fin de transcription, mais il faut également instaurer un système de gestion des alias qui vont permettre aux termes de connaître le chemin d'accès aux tables/domaines correspondants.

La collecte et l'attribution d'alias doivent être simultanés et coordonnés pour garantir l'exactitude de la retranscription. Supposons que, pour chaque domaine récolté, il ne soit fait aucune vérification si celui-ci existe déjà dans l'outil de collecte, dans ce cas, un nouvel alias lui sera attribué. Ce cas de figure peut notamment poser des problèmes pour les éléments temporels, le début et la fin d'un interval étant stockés physiquement dans une même table, créer deux alias reviendrait à effectuer des comparaisons sur deux copies de la table qui ne seraient pas liées.

Il faut cependant rester prudent quant à l'assigniation d'alias. En effet, l'opérande principal peut être le résultat d'une autre expression algébrique. Dans ce cas, il faut utiliser des alias distincts. La gestion des alias dispose donc d'un mécanisme de gestion des niveaux de requête.

5.2.4 Avantages et inconvénients

La classe de traitement des opérateurs est issue d'un squelette qui se veut

générique. Celui-ci se contente de déclarer les opérateurs existants sans y appliquer le moindre traitement. En effet, plusieurs opérateurs n'ont pas d'impact sur le résultat final et ne sont là que pour structurer l'information. Ceux-ci ne seront donc pas redéfinis dans le transcripteur spécifique. Cette architecture, et les outils utilisés, présentent quelques avantages et inconvénients qui sont détaillés ci-dessous.

5.2.4.1 Avantages

1. L'utilisation de l'API SAX en lieu et place de l'API DOM demande beaucoup moins de ressources à la machine et ne nécessite pas de création de structures aussi complexes.
2. L'appel aux opérateurs est simplifié. Ajouter un opérateur ne devrait demander aucune modification des opérateurs déjà présents ni de modifier la routine de traitement principale.
3. La transcription des différents opérateur est généralement très courte et facilement compréhensible pour un lecteur tiers.
4. La classe générique est réutilisable directement sans aucune modification. En cas de modifications du modèle MADS pour les requêtes, la classe générique pourra vraisemblablement être transformée sans toucher aux transcripteurs existants.

5.2.4.2 Inconvénients

1. L'utilisation et surtout la systématisation de certains traitements ne sont rendus possibles que par la grammaire particulière de la DTD. Pour un portage maximal, il est important de garder cette particularité lors de l'écriture de DTD pour différents types de SGDB.
2. Le programmeur est libre d'ajouter ou non des délimiteurs, ce qui peut être source d'erreur ou de perte d'optimisations.
3. Certaines optimisations sont encore possibles dans le stockage des données et leur consultation.

Chapitre 6 Conclusions

6.1 Etat des lieux

Le but de ce travail était de rechercher et d'appliquer les concepts nécessaires à la gestions du temps dans les bases de données relationnelles objets. Ce projet a abouti à l'élaboration d'une série de règles de transformations, tant au niveau de la traduction qu'aux niveau de la transcription des requêtes, permettant de générer des scripts SQL valides et respectant les contraintes initiales.

La description de l'implémentation des différents concepts analysés a montré certains obstacles rencontrés et les solutions qui y ont été apportées.

L'implémentation de ces recherches est, en grande partie, terminée. Certains opérateurs non temporelles du transcripteur n'ont pas encore été traduits. La grammaire MADSLog pour les SGDB relationnels objets n'est, à l'heure actuelle, pas complète, ce qui ne nous a pas permis de valider toutes les transcriptions sur un plan théorique. Une approche empirique fut utilisée pour ces traitements.

Signalons enfin qu'une faiblesse a été observée dans le développement du projet MurMur. En effet, celui-ci base ses traduction sur une série de grammaires qui permettent de développer les outils de traductions et transcription correspondants. Malheureusement, ces grammaires ne sont pas encore figées et plusieurs versions différentes d'une même grammaire circulent, rendant difficile les optimisations et l'analyse fine de leurs caractéristiques. Ce point est par ailleurs repris dans les pistes futurs.

6.2 Pistes futurs

- Compléter le transcripteur Oracle 9i avec les fonctions manquantes
- Uniformiser la (les) DTD(s) du modèle MADSLog en ce qui concerne les attributs, les données et certaines choix de balisage, comme les opérateurs de comparaison.
- Terminer la traduction de requêtes utilisant la multi-représentation afin de vérifier l'intégration complète de la gestion du temporel.

Chapitre 7 Remerciements

Je tiens à remercier les personnes suivantes qui ont participé de près ou de loin à la réalisation de ce travail :

- Mr E. Zimanyi pour m'avoir permis de réaliser mon mémoire dans son service, pour son aide et ses conseils.
- Mr M. Minnout pour ses conseils et les longs moment de réflexion passés ensemble sur l'évolution du travail.
- Mr J.-F. Raskin et Mr R. Wuyts qui ont acceptés de participer à mon jury malgré ma demande tardive.
- Tous ceux que j'ai honteusement oublié de citer mais qui m'ont aidé ou soutenu à quelque moment que ce soit du projet.

Chapitre 8 Bibliographie

1. Http://www.mur-mur.org/ (site officiel du projet MurMur)
   1.1. A Conceptual Data Model for MurMur (Délivrable 5)
   1.2. Query Language Specification (Délivrable 8)
2. http://cs.ulb.ac.be/publications/P-98-06.pdf (MADS, modèle conceptuel spatio-temporel)
3. http://cs.ulb.ac.be/publications/P-02-04.pdf (Implantation d'un modèle conceptuel avec multi-représentation)
4. « Traduction entre niveaux d'abstraction pour des applications de bases de données » - O. Samyn
5. « The TSQL2 Temporal Query Language » - R. T. Snodgrass – Kluwer Academic Publishers
6. http://java.sun.com/j2se/1.4.2/docs/api/ (documentation des librairies standards Java)
7. http://java.sun.com/xml/jaxp/dist/1.1/docs/api/ (documentation de la bibliothèque JAXP)
8. http://www.w3.org/TR/REC-xml/ (recommandations du format XML)

Chapitre 9 Index

Index des illustrations

Illustration 1 : Concepts MADS d'objet et de relation..10
Illustration 2 : Interface graphique de l'éditeur de schéma... 11
Illustration 3 : Etapes de transformation d'un schéma de base de données.......................12
Illustration 4 : Transformation d'un attribut mutlivalué dans un SGDB relationnel............... 13
Illustration 5 : Transformation d'un attribut multivalué dans un SGDB relationnel objet...... 13
Illustration 6 : Transformation d'un objet spatial en objet avec un attribut spatial................14
Illustration 7 : Interface graphique de l'éditeur de requête.. 14
Illustration 8 : Etapes de transformation d'une requête.. 15
Illustration 9 : Visualiseur de données.. 16
Illustration 10 : schéma MADS d'une base de données... 19
Illustration 11 : schéma MADS d'une base de données...20
Illustration 12 : Hiérachie des éléments temporels... 21
Illustration 13 : Passage de l'élément temporel comme attribut... 21
Illustration 14 : Décomposition de l'attribut temporel en attribut complexe multivalué..........22
Illustration 15 : Décomposition de l'interval en attribut complexe....................................... 22
Illustration 16 : Deux instants sur la ligne du temps..24
Illustration 17 : Différentes possibilités de comparaison entre un instant et un intervalle.....24
Illustration 18 : Différentes possibilités de d'agencement des intervalles............................ 25
Illustration 19 : Exemple de schéma MADS...25
Illustration 20 : Traduction du schéma en MADSLog..26