UNIVERSITE LIBRE DE BRUXELLES
Faculté des Sciences Appliquées

Année académique 2001-2002

Traduction entre niveaux d'abstraction pour des applications de bases de données

DIRECTEUR DE MEMOIRE :
Prof. E. Zimányi

DISSERTATION PRESENTEE
PAR OLIVIER SAMYN
EN VUE DE L'OBTENTION DU DIPLOME
D'ETUDES APPROFONDIES
EN SCIENCES APPLIQUEES

Chapitre 1

Introduction

Quelque soit le domaine auquel il s'intéresse, l'ingénieur est régulièrement amené à décrire la réalité à l'aide d'un modèle. Ce travail est d'autant plus important pour l'ingénieur informaticien qu'il existe, en informatique, de nombreuses manières de décrire le même concept. L'ingénieur informaticien est donc régulièrement amené à apprendre à utiliser de nouveaux modèles, voire à en créer de nouveaux. L'abondance de modèles divers amène, afin de pouvoir faire évoluer les applications déjà écrites, un besoin de conversion entre ces modèles.

Le domaine des bases de données est un de ces domaines riches en modèles différents. Le travail qui suit montre quelques uns de ces modèles, ainsi que différents cas dans lesquels on fait appel à des traductions entre ces modèles. Quelques outils pouvant aider à la conception de tels traducteurs seront aussi passés en revues. Pour terminer je donne quelques pistes à explorer afin de faciliter la tâches de ceux qui doivent créer des traducteurs entre modèles de bases de données.

1.1 Concepts généraux

Je vais exposer ici quelques concepts de base qui seront utilisés dans le reste de ce travail. Ces concepts sont largement utilisés, mais il est bon de les restituer dans leur contexte.

1.1.1 L'utilisateur

La notion d'utilisateur est une notion qui peut paraître simple, mais qui nécessite parfois quelques précisions. Ainsi dans le cas le plus général de l'utilisation d'un modèle, un utilisateur est celui qui essaie décrire la réalité avec ledit modèle.

Dans le cas d'un méta-outil (par exemple, un outil méta-CASE), l'utilisateur est celui qui utilise cet outil, il s'agit donc d'un personne cherchant à implémenter une nouvelle manière de modéliser la réalité. On parlera aussi dans ce cas de l'utilisateur final, qui est celui qui utilisera l'outil conçu à partir du méta-outil.

1.1.2 Niveaux d'abstraction

Lorsque l'on parle de bases de données, on distingue classiquement trois niveaux d'abstraction :

Niveau conceptuel C'est le niveau le plus abstrait. Il utilise des modèles riches en concepts. Il doit permettre à l'utilisateur de décrire le monde réel de la manière la plus fidèle possible. Quelques exemples de modèles : Entité/Relations, UML, MADS.

Niveau logique C'est un niveau intermédiaire qui réexprime le niveau conceptuel dans un modèle compréhensible par le SGDB. Par exemple, dans le cas d'une base de données relationnelle, on parlera en termes de tables et de contraintes.

Niveau physique Le niveau physique est le niveau tel qu'il est stocké par le gestionnaire de base de données. Ce niveau diffère du niveau logique par l'ajout d'optimisations. Ces optimisations peuvent tenter : d'améliorer les temps d'accès à la base de données, de minimiser l'espace de stockage utilisé, de supprimer la redondance d'information, ... Bien sûr, ces différentes optimisations sont incompatibles entre elles, le niveau physique sert donc à adapter la base de données aux besoins techniques.

En général, le niveau physique est exprimé dans le même modèle que le niveau logique, tandis que le niveau conceptuel est exprimé dans un modèle tout différent. C'est pourquoi la plus part des traductions qui seront exposées par la suite seront des traductions du niveau conceptuel vers le niveau logique ; les différentes optimisations possibles sortant du cadre de cet exposé.

Chapitre 2

Traduction de modèles dans le cadre du projet MurMur

Le projet MurMur est un projet financé par la communauté européenne. Il rassemble plusieurs partenaires industriels, académiques et publics :

- Star Informatics
- L'Institut Géographique National Français
- CEMAGREF
- L'École Polytechnique Fédérale de Lausane
- UNIL
- L'Université Libre de Bruxelles

On pourra trouver une introduction au projet dans l'article [17] et des informations supplémentaires sur le site web du projet : `http://www.mur-mur.org`.

La base de ce projet est la modélisation de bases de données multi-représentées pouvant contenir des types de données géographiques et temporelles ; le modèle adopté est le modèle MADS (voir à ce propos [20]). Le but final du projet étant l'implémentation d'un "gestionnaire" permettant la manipulation directe d'une base de données décrite en MADS.

Pour ce faire, plusieurs outils ont été créés :

- Un éditeur de schémas
- Un éditeur de requêtes
- Un visualisateur de données

Il n'existe pas de gestionnaire de bases de données MADS, l'implémentation choisie se base sur des gestionnaires de bases de données existant tel qu'Oracle. L'absence d'outil d'introduction des données est donc compensée par un accès direct au gestionnaire de base de données sous-jacent.

Ce choix d'architecture nécessite de traduire les opérations au niveau MADS en opérations compréhensibles par le SGDB sous-jacent. Deux traducteurs ont été développés par le service Informatique de l'Université Libre de Bruxelles :

- Le traducteur de schémas
- Le traducteur de requêtes

La traduction des données du SGDB sous-jacent vers le visualisateur a été faite par l'entreprise Star Informatics en utilisant des données de traduction fournies pas le traducteur de requêtes. Une certaine expérience dans ce domaine a donc été acquise par l'Université Libre de Bruxelles.

Certaines traductions n'ont pas été abordées, telles que la traduction des contraintes et l'insertion des données. Ces deux domaines pourraient être abordés dans un projet ultérieur.

La suite de ce chapitre décrira brièvement quelques concepts du modèle MADS, le travail fourni sur les deux traducteurs développés, une comparaison entre ces deux traducteurs, et enfin donnera quelques pistes pour la traduction des parties non abordées dans le cadre du projet.

2.1 Le modèle MADS

Le modèle MADS est un modèle tentant d'unifier plusieurs concepts intéressants du monde des bases de données modernes. Ce modèle est basé en partie sur le formalisme d'objets et d'attributs provenant d'UML. La figure 2.1 donne un aperçu de la représentation des différents concepts décrits ci-dessous.

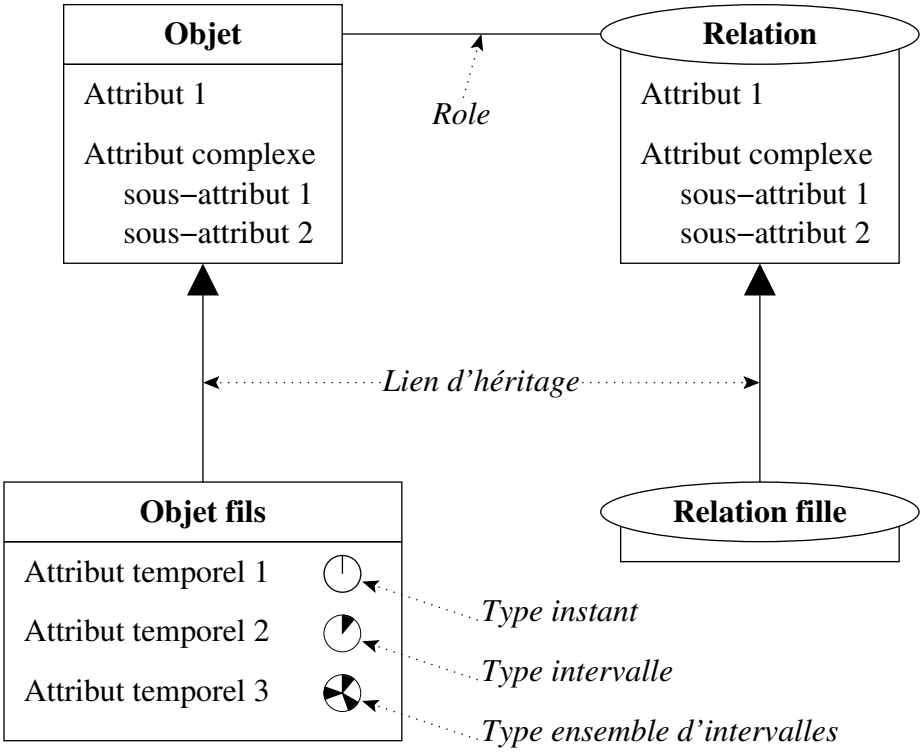

FIG. 2.1 – Concepts MADS

On retrouve deux concepts de base dans le modèle MADS : les objets et les relations. Les objets et les relations peuvent contenir un ensemble d'attributs mono- ou multi-valués. Les objets et relations peuvent être reliés par des rôles de plusieurs types.

Le modèle gère les notions de spatialité et de temporalité. Il existe des attributs spécifiques prédéfinis associés à ces notions : la géométrie de l'objet et son cycle de vie. De même, il existe des types de données étendus permettant d'exprimer des géométries et des temporalités.

Le modèle est orienté objet, il peut donc exister des relations d'héritage entre les objets ou entre les relations.

La notion de multi-représentation a été ajoutée à tout cela. Chaque objet, relation, attribut ou rôle peut être présent dans une série de représentations et invisible dans les autres. Les valeurs des attributs peuvent varier selon la représentation ; ils sont alors dits R-stamped.

Je parlerai donc dans le reste de ce chapitre d'objets, de relations et d'attributs au sens MADS.

2.2 Traducteur de schémas

La traduction d'un schéma MADS vers un schéma compréhensible par un gestionnaire de bases de données courant est une étape importante du projet, puisque c'est sur cette base que se sont faits tous les développements ultérieurs.

Certaines simplifications ont été apportées lors du développement du projet. Ainsi, deux traductions consécutives du même schéma seront identiques, mais la traduction d'un schéma modifié n'est plus compatibles au niveau des noms avec la traduction initiale.

Pour des raisons de temps, les contraintes d'intégrité n'ont pas été formalisées. Le schéma est donc traduit sans ajouter de contraintes d'intégrité ni en traduisant d'éventuelles contraintes déjà définies sur ce schéma. Ce point sera abordé ultérieurement à la section 2.5.2.

2.2.1 Exemples de traduction

Voici un exemple de traduction qui permet d'illustrer le fonctionnement générique du traducteur de schémas. Il s'agit de la traduction du type de données : *Ensemble d'intervalles* vers un SGDB SQL quelconque. Afin de pouvoir aborder la décomposition complète de l'ensemble d'intervalles, voici tout d'abord quelques décompositions de bases.

Pour des questions de modularité, la pluspart des traductions reste exprimé dans le modèle MADS. La dernière étape de traduction consistant alors à traduire les concepts ainsi simplifiés vers un modèle(ou langage) compréhensible par le SGDB cible. La description de ce processus sera abordée à la section 2.2.2

Traduction d'un intervalle simple

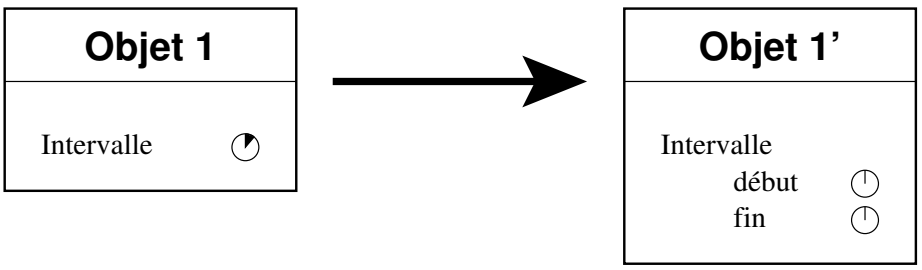

FIG. 2.2 – Décomposition d'un intervalle vers un attribut complexe

Le modèle MADS acceptant la notion d'attribut complexe, la première étape consiste à traduire un intervalle en un complexe contenant deux instants : un début et une fin. Il faut ajouter à cela une contrainte d'intégrité :

L'instant début doit précéder l'instant fin.

Traduction des attributs complexes

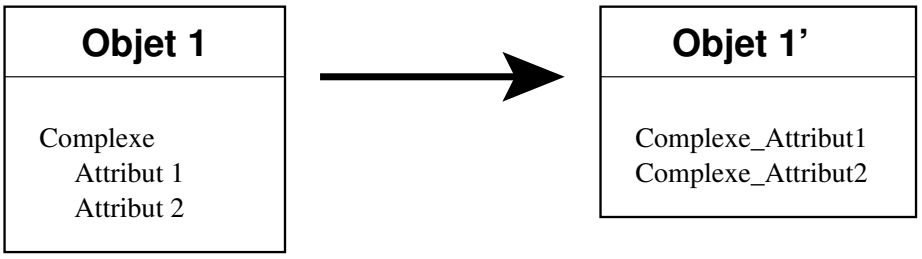

FIG. 2.3 – Décomposition d'un attribut complexe

Voici un exemple de traduction d'un complexe vers deux attributs simples. La traduction proposée ici n'est valable que dans le cas d'attributs mono-valués ; mais elle est suffisante pour l'objectif à atteindre dans ce cas-ci. Dans des cas plus complexes, il faudra transformer les attributs complexes en de nouveaux attributs et créer une relation liant ce nouvel objet à l'objet original.

Traduction des attributs multi-valués

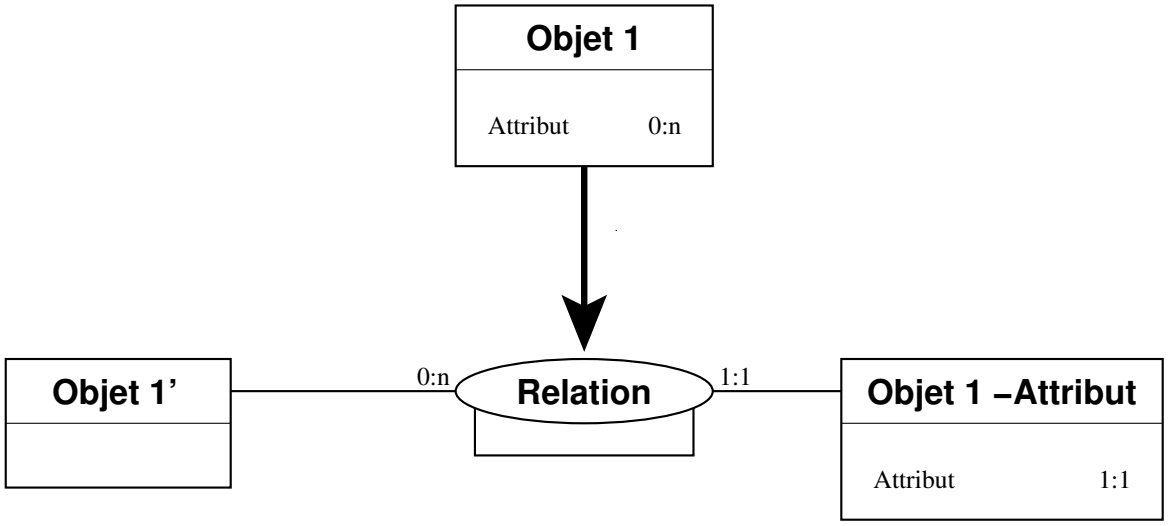

FIG. 2.4 – Décomposition d'un attribut multivalué en un objet et une relation

La traduction d'un multi-valué implique la création d'un nouvel objet et d'une nouvelle relation.

Le passage par une relation pose un problème lorsque deux instances de l'objet contiennent une même valeur pour l'attribut. Il existe deux possibilités dans ce cas : soit stocker une seule fois la valeur (le lien relation-attribut devra alors être de cardinalité 1 :n), soit stocker plusieurs fois

la valeur, avec un éventuel numéro d'identification. Puisque le modèle MADS ajoute un numéro d'identification pour chaque instance d'un objet, c'est la seconde solution qui sera utilisée. Cette solution allège le traitement des modifications dans la base de données. Ces modifications pouvant se faire directement sur l'attribut, sans devoir recopier l'attribut dans la table s'il est utilisé par une autre instance de l'objet.

Traduction de l'ensemble d'intervalles

Pour traduire un attribut de type ensemble d'intervalles, on va tout d'abord le traduire vers un attribut multi-valué de type intervalle. Ensuite, on appliquera les différentes traductions précédemment exposées :

- Attribut multi-valué vers objet et relation (voir 2.2.1)
- Intervalle vers attribut complexe (voir 2.2.1)
- Attribut complexe vers attributs simples (voir 2.2.1)

2.2.2 Stratégie adoptée

En partant des exemples, on peut constater qu'il existe un ensemble de règles de traduction indépendantes les unes des autres qui peuvent se combiner afin de traduire des concepts plus complexes. C'est sur cette base qu'a été conçu le traducteur de schéma MurMur.

Un ensemble de règles plus ou moins simples ont été développées. Selon le SGDB cible, un sous-ensemble de ces règles est appliqué au schéma.

L'application de ces règles se fait de manière itérative. Le schéma de la base de données est stocké sous forme d'arbre. Le traducteur part des feuilles de ce schéma et remonte vers la racine en essayent d'appliquer à chaque élément un ensemble de règles. Si une de ces règles s'applique, on recommence toute l'opération. Le cycle continue jusqu'à ce plus aucune règle ne s'applique.

Le traducteur ainsi créé est un traducteur *générique* travaillant par *itérations*.

Ce style de traducteur est totalement générique. Si les conditions d'entrées pour chaque règles sont bien écrites, il ne demande pas d'ordonner les règles.

Quelques modifications ont été apportées à cette base lors de l'implémentation, principalement pour des raisons de simplifications des conditions d'entrées aux règles :

- Les règles ont été divisées en catégories : génériques, multi-représentation, spatiale, temporelle et cible.
- Les règles de cible modifient le schéma exprimé en MADS vers un schéma compréhensible par le SGDB cible.

Ce style de traducteur a quelques désavantages qui seront abordés dans la section 2.4.

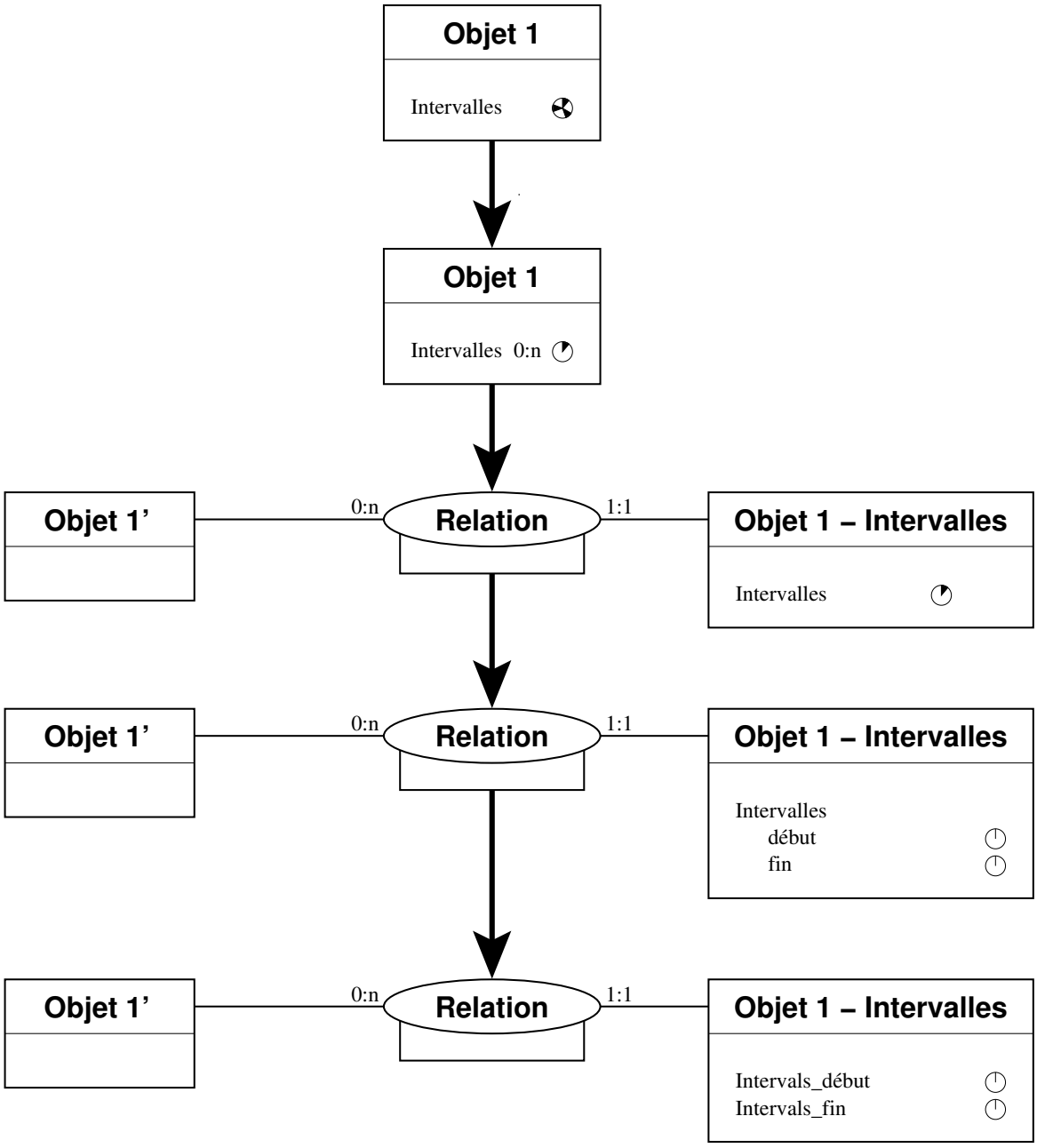

FIG. 2.5 – Traduction d'un attribut de type ensemble d'intervalles

2.3 Traducteur de requêtes

La traduction de requêtes est une deuxième clef du projet. S'il est possible de créer une base de données, l'intérêt est aussi de pouvoir y stocker des données et de les utiliser. L'insertion des données a été laissée de côté (voir le point 2.5.3), l'utilisation des données, par des requêtes, a par contre été implémentée.

Le but de l'opération étant de reconstituer l'information au sens MADS à partir des informations stockées dans le SGDB cible. La structure de l'information stockée étant déterminée par la traduction du schéma. Les deux outils sont fortement liés. Ainsi, quelques informations de traductions sont passées du traducteur de schéma au traducteur de requêtes afin de faciliter le travail de celui-ci.

2.3.1 Exemple de traduction d'une requête

Voici un exemple de traduction de requête. Cette requête reprend comme base la traduction du schéma qui a été faite précédemment. Cette requête est exprimée dans le langage de requête du modèle MADS ; pour plus d'informations, on pourra consulter [15].

Schéma et requête d'exemple

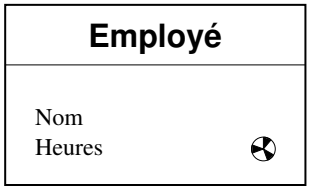

FIG. 2.6 – Schéma utilisé pour l'exemple de requête

Partant d'un schéma assez simple (voir figure 2.6) ne contenant qu'un objet, je vais poser une requête assez simple : il s'agit de retrouver tous les employés travaillant aujourd'hui. Autrement dit, on recherche les objets *employés* dont l'attribut *heures* contient l'instant *today*. La requête exprimée en algèbre MADS donne :

***SELECT**(heures.**contains(today)**) Employé*

La traduction de ce schéma découle de ce qui a été exposé précédemment (voir section 2.2). La traduction de la requête sera divisée en deux étapes : traduction de la fonction *contains* et traduction du type de donnée *ensemble d'intervalles*

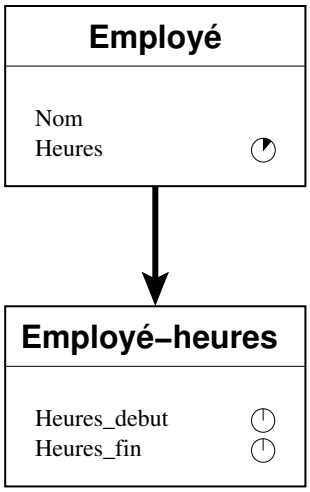

FIG. 2.7 – Schéma contenant un intervalle et sa traduction

Traduction de la fonction *contains*

Dans le cas où le SGDB cible est un SGDB relationnel classique, il n'existe presque pas de gestion du temps, si ce n'est un type de donnée *date* correspondant au type *instant* en MADS. Il faut donc palier aux manques de fonctions en les implémentant directement dans les requêtes.

Soit l'expression *a.contains(b)* où *a* est un attribut de type *intervalle* et *b* est un instant. Les fonctions *start()* et *end()* renvoient respectivement le début et la fin d'un intervalle. On pourra traduire la notion de contenance temporelle par :

$$(a.\textbf{\textit{start}}() \leq b) \textbf{\textit{and}} \ (a.\textbf{\textit{end}}() \geq b)$$

Dans le cas qui nous préoccupe, les intervalles ont été traduits par des attributs de type *instant* (voir figure 2.7). Il faudra donc adapter la traduction de la fonction en conséquence. Dans le cas *a.contains(b)*, cela donnerai :

$$(a\_debut \leq b) \textbf{\textit{and}} \ (a\_fin \geq b)$$

Traduction du type de donnée *ensemble d'intervalles*

Les SGDB classiques n'acceptant pas les attributs multi-valués, on y pallie grâce à une relation et un nouvel objet. Lors de la traduction d'une requête sur ce schéma (voire figure 2.8), il faudra tenir compte de l'apparition de cette relation dans la description des chemins. Il faudra aussi tenir compte du fait qu'un des rôles reliant la relation à un objet est de cardinalité multiple ; il faudra donc ajouter un quantificateur existentiel.

Soit l'expression *a.contains(b)* où a est un attribut de type *ensemble d'intervalles* et b est un *instant*. La traduction de *a* en une relation et un objet contenant un attribut de type *intervalle* donnera comme expression :

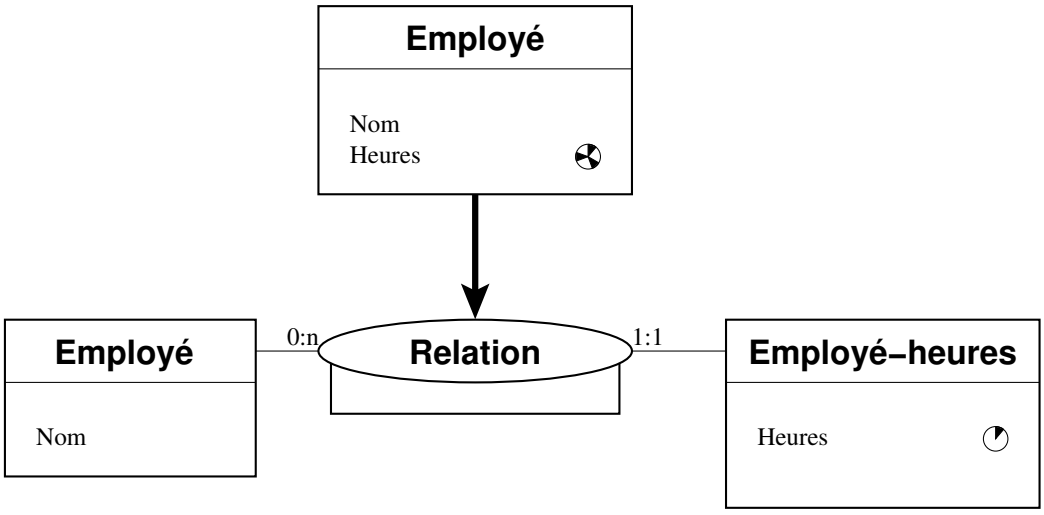

FIG. 2.8 – Schéma et la traduction de la multi-valutation apparaissant dans le cas d'un attribut de type *ensemble d'intervalles*

$$\exists\ r \in \mathit{Relation},\ r.\mathit{objet}.\boldsymbol{contains}(b)$$

Traduction complète

En reprenant le schéma figure 2.6 et en le traduisant selon les règles définies au point 2.2, on obtient un schéma(voir figure 2.9) ne contenant ni intervalle, ni multi-valutation.

On pourra, en appliquant les deux parties vues précédemment, déduire la traduction finale de la requête :

$$\boldsymbol{SELECT}(\exists\ r \in \mathit{Relation},\ r.\mathit{Employé\text{-}Heures.Heures\_debut} \leq \boldsymbol{today}\ \boldsymbol{and}\ r.\mathit{Employé\text{-}Heures.Heures\_fin} \geq \boldsymbol{today})\mathit{Employé}$$

2.3.2 Stratégie adoptée

En observant les deux règles de traduction qui précèdent, on peut déduire que le fonctionnement du traducteur de requêtes est différent de celui du traducteur de schéma. Les itérations successives ont été abandonnées au profit d'une application systématique des règles. Chaque règle de traduction recherche, dans l'arbre de requête, les noeuds qui l'intéressent et applique diverses transformations sur ces noeuds.

Chaque règle n'est appliquée qu'une et une seule fois, ce qui entraîne un besoin d'ordonnancement strict des règles.

Je parlerai dans ce cas-ci d'un traducteur *générique* travaillant par application *ordonnée* des règles.

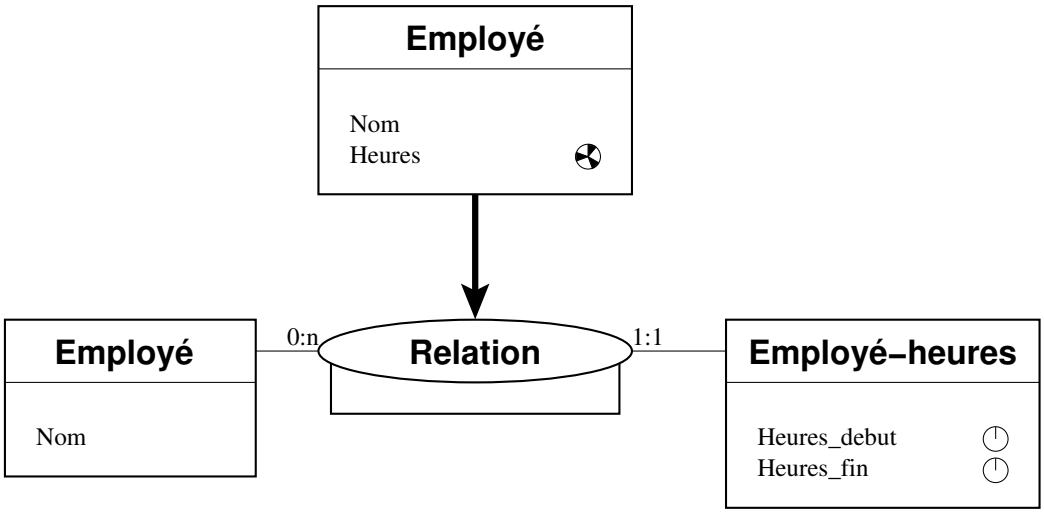

FIG. 2.9 – Schéma de requêtes et sa traduction

La généricité de ce traducteur est un peu moindre que celle du traducteur de schéma, principalement parce que les règles sont écrites en tenant compte de la traduction finale du schéma. Les règles sont donc dépendantes de la cible.

Le traducteur n'en reste pas moins générique, car la configuration des règles à appliquer reste très simple et parce qu'il existe un ensembles de règles réutilisables pour plusieurs cibles différentes.

2.4 Comparaison entre les deux traducteurs

Il n'est pas aisé de comparer deux programmes dont le but est différent. On peut néanmoins comparer les modes de fonctionnement des deux programmes ainsi que les performances respectives, donnant une idée de l'efficacité de l'algorithme choisi.

Dans les deux cas, les arbres décrivant le schéma ou la requête sont stockés dans un format XML. Les deux programmes sont écrit dans le langage Java. Les technologies utilisées étant les même, on peut comparer les temps d'exécution des différents algorithmes en supprimant les temps de chargement des données.

Le traducteur de schéma travaille sur un seul fichier d'entrée dont la longueur varie en fonction de la taille du schéma d'entrée.

Le traducteur de requêtes travaille avec trois fichiers d'entrée :
- Le schéma d'origine
- Le schéma traduit
- La requête originale

La taille du fichier principal à traiter est celle du fichier requête. Cette taille varie donc avec la complexité de la requête. Elle est indépendante du schéma.

Le temps d'exécution de la traduction d'une requête simple sur un schéma complexe sera pénalisée par les temps de recherche des éléments de traduction dans le schéma d'origine et le schéma traduit.

Si l'on compare qualitativement les temps d'exécution des deux traducteurs, on remarquera que le traducteur de requêtes est plus rapide que le traducteur de schémas, même lors de la traduction de requêtes complexes sur des schémas complexes.

La grande différence entre les deux traducteurs se situe au niveau de l'algorithme. On peut comparer qualitativement différentes caractéristiques des deux algorithmes employés.

Critère	**Traducteur**	
	de schémas	**de requêtes**
Vitesse d'exécution	-	+
Réutilisabilité des règles	+	-
Simplicité d'ajout ou de suppression d'une règle	=	=
Simplicité des conditions d'entrées	-	+
Simplicité du traitement des règles	+	-
Facilité de corrections d'erreurs	-	+

TAB. 2.1 – Comparaison qualitative entre les deux traducteurs

Quelques notes à propos du tableau 2.1 :
- La simplicité d'ajout et de suppression des règles n'est autre que la manière dont la liste des règles à appliquer est écrite. Le système est le même pour les deux traducteurs, si ce n'est qu'une information d'ordonnancement, sous forme d'un numéro d'ordre, a été ajoutée dans le traducteur de requêtes. Les deux systèmes sont les mêmes.

- Au niveau des conditions d'entrées, le traducteur de requêtes est plus simple, car chaque règle ne s'intéresse qu'à un ensemble restreint de noeuds, de plus, de par l'ordonnancement des règles, des hypothèses peuvent être faites quand à la structure de la requête au moment de l'application de la règle. Chaque hypothèse permet de supprimer une ou plusieurs conditions d'entrée.
- Au niveau du traitement, les règles du traducteur de schéma sont plus simples, car elles se contentent de modifier l'arbre. Les règles du traducteur de requêtes sont plus complexes car il est nécessaire d'aller rechercher des informations de traduction dans les schémas. La différence, à ce point de vue, entre les deux traducteur provient essentiellement des buts poursuivit et non du choix de l'algorithme.
- Au niveau de la correction des erreurs, l'ensemble de conditions d'entrée et le caractère itératif des règles du traducteur de schémas entraîne une dépendance forte entre les différentes règles. Il n'est donc pas aisé d'isoler une erreur, et une condition d'entrée posant problème, entraîne souvent des erreurs dans d'autres parties de la traduction n'ayant rien à voir avec la règle fautive. Dans le cas du traducteur de requêtes, les règles de traduction sont beaucoup moins dépendantes les unes des autres. La correction d'erreurs en est d'autant facilitée.

D'un point de vue théorique, le traducteur de schémas est plus puissant car il reste fortement générique. D'un point de vue pratique, le traducteur de requêtes est plus puissant car il allège le temps de calcul nécessaire.

Des améliorations futures devront être apportées aux différents traducteurs afin de trouver un équilibre entre les deux méthodes de traduction. Quelques pistes à explorer sont exposées dans les chapitres suivants.

2.5 Parties non abordées durant le projet

Un ensemble de traductions connexes aux traductions de schéma et de requêtes n'ont pas été prises en compte lors de la conception du projet.

Les différents logiciels écrits n'ont pas non plus été conçus pour l'ajout futur de telles traductions. Un travail non négligeable devrait donc être fait afin d'améliorer les outils actuels.

Différentes pistes visant à intégrer les différents modules de traductions seront abordées dans les chapitres suivants et devront faire l'objet de recherches plus avancées.

2.5.1 Modifications du schéma

Une base de données reflète une certaine réalité et est utilisée par un ensemble d'acteurs différents. La réalité modélisée dans la base de données peut varier, ainsi que les besoins des différents acteurs utilisant la base de données. Le schéma de la base de données va donc varier en fonction de ces besoins, entraînant le besoin d'outils de modification du schéma.

Aucune trace des différentes règles appliquées, lors de la traduction du schéma, n'étant stockée, il n'est pas possible de réexécuter la même séquence de traduction et d'y ajouter des séquences de modifications du schéma. La traduction d'un schéma modifié risque donc de donner une base de données très différente de la base de donnée originale. De plus, les parties non modifiées du schéma risquent de se trouver modifiées au niveau de la base de données.

L'implémentation d'un outil de modification n'est donc pas possible avec les outils de traduction tels qu'ils sont implémentés actuellement.

2.5.2 Traduction des contraintes

Les contraintes d'intégrité n'ayant pas été modélisées dans le cadre du projet, il n'y a pas eu de module de traduction spécifique écrit pour ces contraintes.

La traduction des contraintes d'intégrité doit se faire en deux étapes :

1. Traduction des chemins utilisés
2. Traduction de la contrainte

La traduction des chemins est identique à la traduction des chemins dans le cadre du traducteur de requêtes. La traduction de la contrainte fait appel à des notions de programmation au niveau de la base de donnée cible. Il est donc nécessaire d'ajouter un module de traduction de ces contrainte.

2.5.3 Manipulation des données (insertion, suppression, modification)

Un gros point noir du projet MurMur concerne la manipulation des données elles-mêmes : aucun logiciel de manipulation des données n'a été prévu. L'insertion et la modification des données est laissée à l'utilisateur, obligeant celui-ci à savoir utiliser la base de données cible.

La réalisation d'un tel outil serait possible sur base des outils actuels. Les besoins d'un tel outil étant limités à la traduction de chemins simples, une réutilisation du code du traducteur de requêtes pourrait être envisagée.

2.6 Conclusion

Le projet MurMur a permis au service informatique de la faculté des sciences appliquées de l'Université Libre de Bruxelles d'acquérir une certaine expérience dans le domaine de la traduction entre modèles de bases de données.

Les logiciels écrits sont fonctionnels, extensibles et fiables. Il comportent tout de même des lacunes qui sont dues principalement à des choix stratégiques fait lors de la conception même du projet.

Des développements futurs dans le domaine de la traduction de concepts doivent être envisagés afin d'envisager le problème de la traduction de concepts de manière globale.

Chapitre 3

Différents contextes d'utilisations de traducteurs

Le chapitre précédent montrait le travail fournit pour le projet MurMur ; ce projet proposant un modèle conceptuel travaille principalement dans le contexte de création du schéma et des requêtes d'une base de donnés, tout en traduisant le résultat vers les autres niveaux d'abstractions.

Une fois la base de données conçue et mise en production, il est impératif de la garder à jour et de la faire évoluer afin de rester en concordance avec la réalité qu'elle est censée modéliser. Le cycle de vie d'une base de données rencontrera donc probablement des phases d'évolution et éventuellement des phases d'intégration à d'autres bases de données.

Ces différentes phases travaillant généralement au niveau conceptuel, il sera nécessaire de répercuter les changements apportés sur les autres niveaux d'abstraction. Il sera donc nécessaire d'utiliser des traducteurs, afin de pouvoir réexprimer les modifications apportées dans les autres niveaux d'abstraction.

3.1 Conception d'une base de données

Le contexte de conception d'une base de données a déjà été abordé dans la description du travail réalisé pour le projet MurMur (chapitre 2). Il existe néanmoins d'autres manières concevoir et modéliser des bases de données. Cette section aura donc pour but de décrire un ensemble non exhaustif de traductions classiques. L'ensemble des traductions présentées ici sont exprimées dans le modèle relationnel, compatible avec à la plus part des SGDB actuels (excepté les SGDB Objets).

3.1.1 Modèle Entité-Relation

Le modèle entité-relation est l'un des modèles le plus utilisé pour la conception de bases de données. Ce modèle est bien établi et a déjà fait l'objet de nombreuses études. On pourra trouver dans [5] une introduction au modèle entité-relation, ainsi qu'un ensemble de règles visant à traduire les schémas exprimés dans ce modèle vers le modèle relationnel.

Je vais présenter ici quelques règles de traduction portant sur les relations. Il n'existe pas, dans le modèle relationnel, de relations à proprement parler, il faut donc pallier à ce manque par un ensemble d'astuces dépendant du type de relation concerné.

Traduction des relations 1 :1

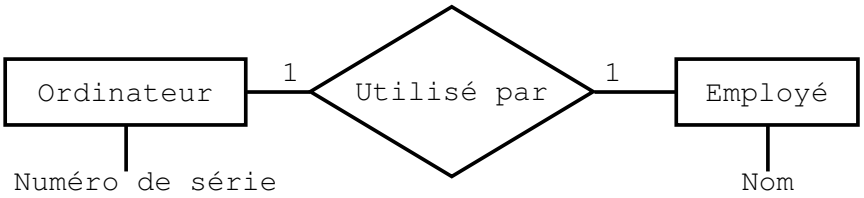

FIG. 3.1 – Schéma avec une relation 1 :1

Dans la figure ci-dessus, un ordinateur est utilisé par un et un seul employé, et un employé n'utilise qu'un est un seul ordinateur. Nous avons donc une relation 1 :1. Si on traduit dans le modèle relationnel chacune des entités, on obtient :

Ordinateur (Numéro_série)
Employé (Nom)

Pour y reproduire la relation, on ajoute le nom de l'employé utilisant l'ordinateur dans la table **Ordinateur**. Ce qui nous donne comme schéma final (l'attribut nom est renommé vers "nom_employé" pour éviter toute confusion) :

Ordinateur (Numéro_série , Nom_employé)
Employé (Nom)

Traduction des relations m :n

Dans la figure ci-dessous, un employé travaille pour un ou plusieurs départements, et chaque département emploie un ou plusieurs employés. Nous avons donc une relation m :n.

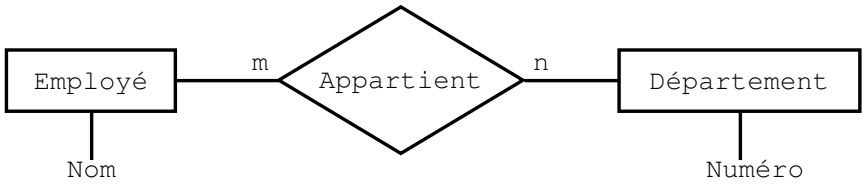

FIG. 3.2 – Schéma avec une relation m :n

Si on traduit dans le modèle relationnel chacune des entités, on obtient :

Employé (Nom)
Département (Numéro)

On ne peut, comme dans la transformation précédente, ajouter le nom de l'employé dans l'entité département. Il faudrait recopier plusieurs fois les informations d'un département pour y stocker un employé. Il y aurait donc redondance dans la base de données ce qui d'un point de vue conceptuel est à éviter.

Pour reproduire la relation dans le schéma relationnel, on crée une entité qui contient une copie des clefs primaires des objets que la relation lie. Ce qui donne dans ce cas-ci :

Employé (Nom)
Département (Numéro)
Appartient (Numéro_Département , Nom_Employé)

Conclusion

Ces deux exemples nous montrent qu'il est tout à fait possible et sans grandes difficultés de traduire des schémas exprimés dans le modèle entité-relation vers le modèle relationnel. Ces règles de traduction sont bien connues et sont largement disponibles.

Du point de vue des requêtes, contraintes d'intégrité, ... on se repose en général sur une écriture directe au niveau relationnel.

3.1.2 Traduction de concepts temporels

De nombreuses extensions ont été proposée pour enrichir le modèle relationnel et son langage de prédilection : SQL. Parmi ces différentes extensions, l'une des plus abouties est l'extension TSQL[3] qui propose l'ajout de la gestion du temps dans les bases de données relationnelles.

TSQL2 est un langage de gestion de bases de données, tout comme SQL, mais il ajoute un ensemble de fonctionnalités propres à la gestion du temps :

- Ajout des types de données temporels, tels que : instant et intervalle
- Ajout de fonctions de comparaison des types de données temporels
- Adaptations de l'algèbre de requêtes aux manipulations temporelles
- Possibilité de déclarer des tables "timestamped". Dans ce type de tables, chaque tuple est associé à une période de validité. Lorsque l'une des valeurs de ce tuple change, un nouveau tuple est ajouté avec une nouvelle période de validité. L'évolution des données est ainsi conservée.
- Ajout de la définition de moment de transaction. Dans les tables utilisant cette possibilité, le moment d'ajout et de suppression d'un tuple est stocké.

Parallèlement à l'ajout de ces fonctionnalités, un ensemble d'études a été réalisé autour de l'implémentation de ce langage. Il est possible de trouver des articles ou livres donnant la manière exacte de traduire des schémas, des requêtes,... décrits avec les concepts de TSQL vers les mêmes schémas, requêtes,... dans le modèle relationnel classique.

Bien que le langage ait été standardisé, il n'en existe pas encore d'implémentation commerciale ; les entreprises de développement attendant l'intégration du langage TSQL dans le langage SQL standard.

Les concepts mis en évidences dans TSQL sont néanmoins utilisés dans plusieurs projets. C'est ainsi que la traduction de la partie temporelle du modèle MADS vers des modèles plus simples est fortement inspirée des traductions données pour la traduction des concepts TSQL vers les bases de données relationnelles. Les exemples de traduction donnés au chapitre2 faisant intervenir les types de données temporels sont de bons exemples de traduction possible de TSQL vers le modèle relationnel classique (Exceptés l'introduction des relations, mais ce problème est résolu ci-dessus).

3.2 Évolution d'une base de données

L'évolution d'une base de données est un élément important du cycle de vie d'une base de données. En effet, lors de sa conception, la base de données reflète une certaine réalité, prise à un moment précis. Cette réalité peut évoluer à cause d'un ensemble d'éléments extérieurs, par exemple : nouveaux besoins des utilisateurs, modifications de contraintes légales, changement du matériel ou du SGDB utilisé, ... Dès lors, il faudra adapter la base de données à la nouvelle réalité qu'elle doit représenter.

3.2.1 Exemple d'évolution

Voici pour commencer un exemple classique d'évolution d'une base de données. Il s'agit de la décomposition d'un attribut, dans ce cas-ci, on décomposera l'adresse d'une personne en un nom de rue, un code postal et une ville. La base de données sous-jacente devra donc être modifiée en conséquence.

L'exemple contient chaque fois : le schéma conceptuel dans le modèle entité-relation, le schéma logique dans le modèle relationnel, un exemple de données et un exemple de requête exprimée dans le langage SQL et visant à retrouver l'adresse de la personne ayant pour nom : "petit".

Schéma Entité-Relation

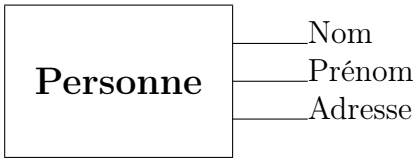

Schéma Relationnel

Personne (Nom, Prénom, Adresse)

Exemples de données

Personne		
Nom	***Prénom***	**Adresse**
Samyn	Olivier	Chaussée de Watermael, 84 1160 Bruxelles.
Petit	Jules	Rue de l'étuve 1000 Bruxelles.

Exemples de requête

```
SELECT adresse FROM Personne WHERE nom="Petit"
```

FIG. 3.3 – Schéma initial de la base de données

Le nouvel attribut adresse restant unaire, la modification revient à supprimer l'attribut adresse du schéma initial (figure 3.3) et à ajouter au schéma modifié(figure 3.4) trois nouveaux attributs : rue, code postal et ville.

Schéma Entité-Relation

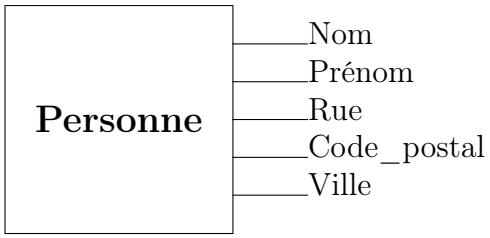

Schéma Relationnel

Personne (Nom, Prénom, Rue, Code_Postal, Ville)

Exemples de données

Personne				
Nom	***Prénom***	**Rue**	**Code_postal**	**Ville**
Samyn	Olivier	Chaussée de Watermael, 84	1160	Bruxelles
Petit	Jules	Rue de l'étuve	1000	Bruxelles

Exemples de requête

SELECT rue, code_postal, ville FROM Personne WHERE nom="Petit"

FIG. 3.4 – Schéma modifié de la base de donnée

Du simple exemple donné ci-dessus, on peut noter que :
- Les modifications faites à un niveau d'abstraction doivent être répercutées sur les autres niveaux
- Les données sont modifiées.
- Les requêtes sont modifiées.

L'évolution d'une base de données est donc un processus complexe nécessitant des modifications à tous les niveaux d'abstractions.

3.2.2 Fonctionnement type d'un outil d'évolution

Afin de faciliter le travail du gestionnaire de bases de données, un ensemble d'outils d'évolution de bases de données ont vu le jour. Certains outils permettent de modifier le schéma à n'importe quel niveau d'abstraction et propagent ces modifications vers les autres niveaux. Je me limiterai

afin d'illustrer le fonctionnement de ces outils aux modifications du niveau conceptuel propagées vers les niveaux logiques et physiques.

On trouvera ci-dessous quelques concepts importants pour le fonctionnement de ce type d'outils. Les différents concepts sont décrits en général pour un passage du schéma conceptuel vers le schéma logique, les mécanismes de traduction entre les autres niveaux étant similaires.

Transformations simples

Plutôt que d'utiliser des transformations très complexes, et peu utilisées, on essaiera de créer un ensemble de transformations pouvant se combiner pour donner naissance à des transformations plus complexes.

Chaque transformation devra être exprimée pour chaque niveau d'abstraction. Il faudra aussi exprimer l'impact de cette transformation sur les requêtes, les contraintes, ... Chaque transformation de base doit faire l'objet d'une étude approfondie et doit être totalement spécifiée.

Lorsque les modifications de bases sont totalement définies, on peut ajouter au modèle un ensemble de modifications étendues. Ces modifications seront de préférence composées de modifications simples, bien que dans certains cas, il faudra directement spécifier certaines parties dans les niveaux d'abstractions inférieurs.

On peut voir figure 3.5 l'application d'une modification et la traductions du schéma conceptuel vers le schéma logique pour un schéma modifié. Les modifications sont représentées par *Mod* et *Mod'*. *Mod'* est l'ajout nécessaire à la traduction du schéma conceptuel vers le schéma logique pour refléter la modification apportée au schéma conceptuel initial.

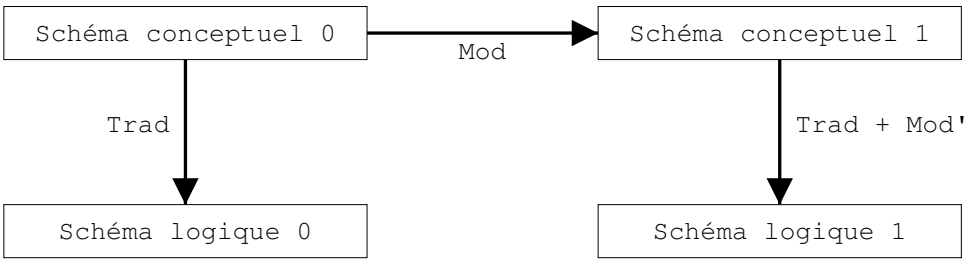

FIG. 3.5 – Processus d'évolution d'un schéma

Historique

Si l'on se reporte à la figure 3.5, on peut remarquer qu'après avoir modifié un schéma conceptuel, il est nécessaire de propager l'information vers le schéma logique. Pour ce faire, on utilise l'ensemble des règles de traduction appliquées au schéma initial augmenté d'une règle de traduction dépendant de la modification appliquée (notée ici *Mod'*).

Il est donc nécessaire de garder une trace des règles de traductions appliquées afin d'être capable de reconstruire à l'identique le schéma logique à partir du schéma conceptuel. Cette trace

est stockée dans l'historique. On y ajoutera la trace de toutes les modifications apportées. On pourra ainsi modifier plusieurs fois successivement la base de données.

Dans certains cas, les modifications apportées sont trop importantes pour pouvoir utiliser totalement l'historique. Dans ces cas-là, on rejouera une partie de l'historique afin de faciliter la tâche de l'utilisateur.

3.2.3 Impact des modifications sur les programmes

Il ressort des différentes sections ci-dessus qu'il est possible de modifier le schéma conceptuel d'une base de donnée et de propager ces modifications vers le niveau physique et donc vers le SGDB sous-jacent.

Cependant, les modifications apportées à un schéma de base de données n'ont pas que des impacts sur ce qui est stocké dans le SGDB. Si on reprend l'exemple 3.2.1 donné ci-dessus, on peut remarquer que les requêtes ont changées, ainsi que la manière d'insérer les données. Les modifications auront donc un impact sur les applications utilisant cette base de données.

Les développeurs de ces applications devront donc adapter leurs programmes pour pouvoir utiliser la nouvelle version de la base de données. Pour leur faciliter la tâche, l'outil d'évolution de base de données peut par analyse syntaxique pointer les endroits susceptibles de devoir être modifiés. Un ensemble de modules doit donc être créé afin que l'outil d'évolution comprenne les différents langages de programmation susceptibles d'être utilisé.

Une autre solution consiste à séparer la manière dont est stocké l'information de la manière dont elle est utilisée dans le programme. On utilise alors ce que l'on appelle un *wrapper*, un petit programme faisant le lien entre la représentation interne dans la base de donnés de l'information et la représentation utilisée dans le reste du programme. Ce type de solution est utilisé le plus souvent lors de l'intégration de bases de données.

3.3 Intégration et fédération de bases de données

Les bases de données ayant fait leurs preuves pour le stockage d'informations, le nombre d'applications spécifiques reposant sur des bases de données a explosé. Il n'est pas rare de voir, au sein d'une même entreprise, un ensemble de bases de données contenant des informations similaires, mais stockées de manières différentes et utilisées par différentes applications.

Une solution radicale dans ce genre de cas serait de créer une nouvelle base de données, reprenant toutes les informations provenant des différentes bases de données disponibles. C'est ce que l'on appelle l'intégration des bases de données.

Une autre solution, qui évite de devoir réécrire toutes les applications utilisant ces bases de données, est de créer une base de données virtuelle. Cette base de données n'est qu'une vue, permettant à de nouvelles applications d'accéder à l'ensemble des données disponibles. L'accès à ces données se fait via un logiciel d'intégration qui se charge d'aller récupérer les informations dans les différentes bases de données disponibles. Cette solution est appelée fédération de bases de données.

D'un point de vue conceptuel, ces deux solutions sont identiques. Elles nécessitent toutes deux la création d'un schéma conceptuel réunissant les points communs des bases de données à unifier, et la traduction de ce schéma vers d'autres niveaux d'abstraction. La différence se situe au niveau physique où la solution d'intégration créera une nouvelle base, tandis que la solution de fédération créera un logiciel permettant d'accéder aux données.

Je commencerai pas exposer ci-dessous la manière de créer le schéma conceptuel de telles bases de données, suivront ensuite quelques notes pour chacune des deux solutions.

3.3.1 Création du schéma intégré

Le processus d'intégration de schémas de bases de données en vue de créer un schéma intégré est complexe et demande de nombreuses interventions humaines permettant de résoudre les conflits qui peuvent apparaître lorsque l'on tente d'unifier la manière de représenter une même information.

Afin de simplifier quelque peu la description du processus de création, je supposerai que le gestionnaire des bases de données a en sa possession l'ensemble des schémas conceptuels décrivant la structure des bases de données devant être intégrées.

A partir de ces schémas conceptuels, il faudra passer par trois grandes étapes représentées figure 3.6.

Ces trois étapes sont :

Transformation des schémas Les bases de données à intégrer pouvant provenir d'horizons différents, leur schéma conceptuel peut être exprimer dans différents modèles. Afin de pouvoir manipuler les différents concepts définis dans ces schémas, il est nécessaire de tout traduire dans un modèle unique qui sera similaire à celui utilisé pour décrire le schéma intégré.

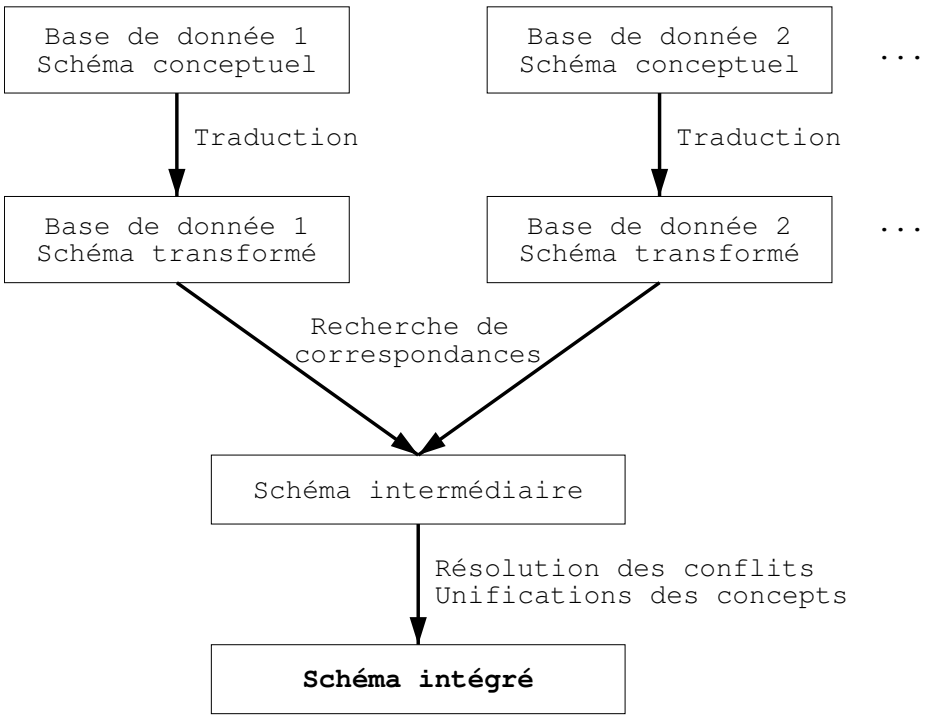

FIG. 3.6 – Tâches à effectuer pour intégrer des schémas de bases de données

Recherche de correspondances Le but étant d'intégrer les différentes bases de données, il faut rechercher, parmi les différentes bases des données disponibles, les concepts similaires et les faire correspondre entre eux. On crée ainsi un schéma intermédiaire contenant les concepts provenant de toutes les bases de données à intégrer, ainsi que les liens unissant ces concepts.

Création du schéma intégré Afin de créer un schéma final, il reste à unifier les concepts similaires et à résoudre les conflits pouvant apparaître entre les différents concepts. Certaines modifications peuvent être apportées au passage, par exemple de renommer certaines entités ou supprimer celles qui ne seront plus utiles par la suite.

Une fois le schéma intégré, il ne reste plus qu'à le mettre en production soit en intégrant les bases de données, soit en les fédérant.

3.3.2 Intégration de bases de données

L'intégration de bases de données est le processus qui du point de vue de l'information stockée est le plus simple. L'information est, en effet, copiée des anciennes bases de données vers la nouvelle, les seules difficultés provenant de la traduction des données dans leur nouveau format de stockage.

Une nouvelle base de données étant créée, les anciennes deviennent désuètes et peuvent donc être supprimées.

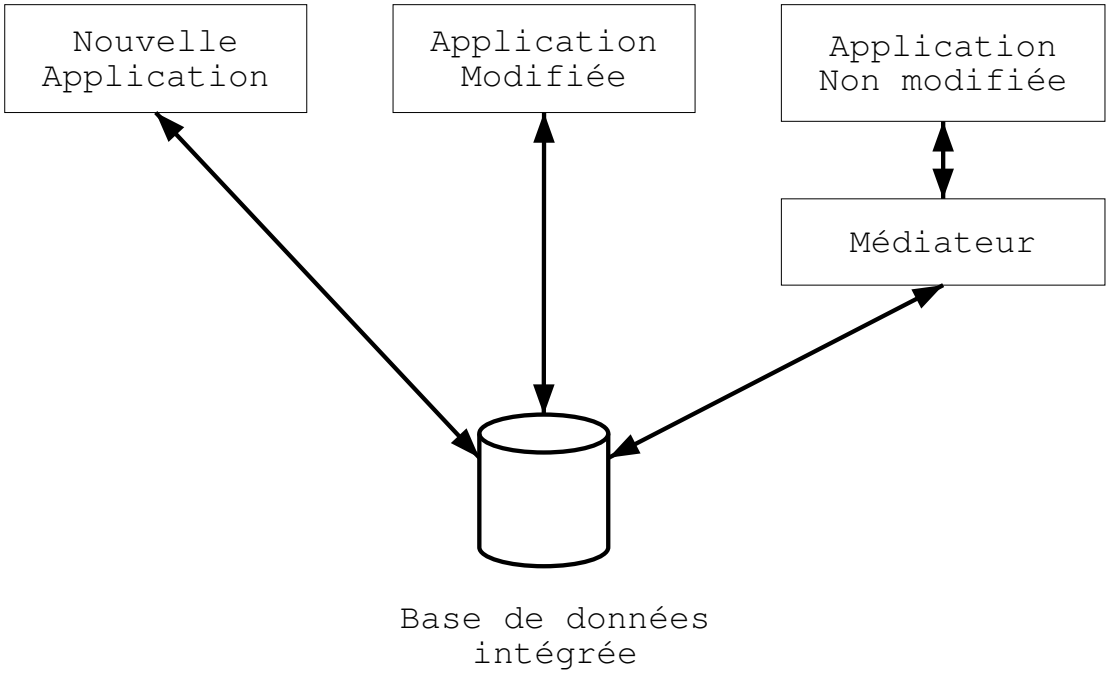

FIG. 3.7 – Accès des applications à la base de données intégrée

Le format de stockage des données étant modifié, la manière d'accéder aux données change aussi. Il faudra donc adapter d'une manière ou d'une autre la partie accès à la base de données. Pour ce faire, trois options sont possibles (voir figure 3.7) :

Créer une nouvelle application Soit il n'existe pas d'application répondant à un besoin, soit cette application est totalement désuète et doit être réécrite. Dans ce cas, il est envisageable d'écrire une nouvelle application répondant aux besoins de l'ancienne et ajoutant les nouvelles fonctionnalités nécessaires. Cette nouvelle application utilise directement la nouvelle structure.

Modifier une ancienne application Si le code source d'une ancienne application répondant encore aux besoins est disponible et suffisamment documenté, il peut être envisagé de modifier cette application afin qu'elle utilise directement la nouvelle structure.

Créer un médiateur Si le code source d'une ancienne application répondant encore aux besoins n'est pas disponible, il n'est pas possible de modifier cette application. Dans ce cas, soit il faut réécrire l'application (et donc en créer une nouvelle), soit créer une application intermédiaire, appelée *médiateur*. L'ancienne application continuera à envoyer et recevoir des informations dans l'ancien format, le médiateur se chargeant de faire correspondre ces informations avec celles disponibles dans la nouvelle structure.

Quelque soit la solution choisie, l'ensemble des applications utilisant les anciennes bases de données doit être passé en revue afin d'en adapter le fonctionnement à la nouvelle structure. Un tel processus est coûteux tant en terme de temps qu'en terme financier. Il ne pourra être envisagé que dans des processus d'intégration à long termes visant à une réécriture complète des applications.

3.3.3 Fédération de bases de données

Contrairement à l'intégration des bases de données, le processus de fédération est plus complexe au niveau du stockage de l'information. Les informations restant stockées dans les anciennes bases de données, il est nécessaire, pour fédérer les bases de données, de mettre en place un système de gestion de l'information.

Ce système de gestion a pour tâche de veiller à la cohérences de l'information stockée en général plusieurs fois sous des formats différents dans les différentes bases de données composant la fédération.

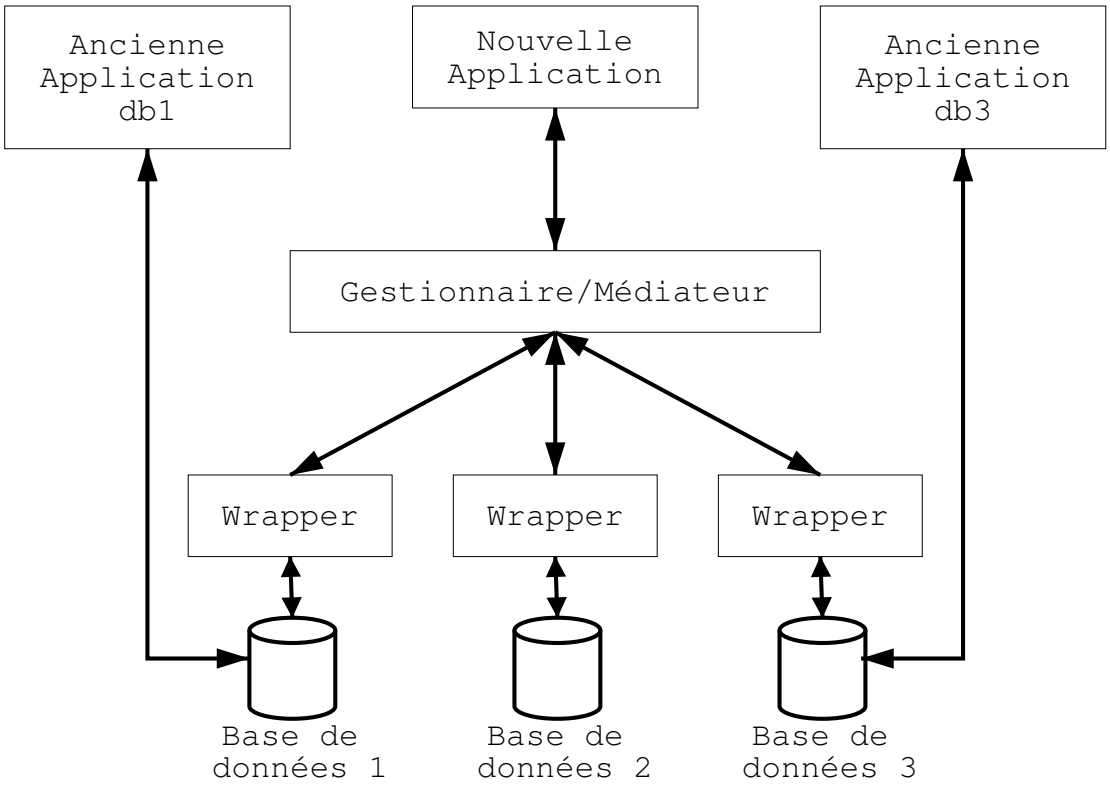

FIG. 3.8 – Accès des applications aux base de données fédérées

Le rôle du système de gestion est aussi de permettre l'accès aux informations de la base de données virtuelle reposant sur la fédération. La figure 3.8 montre différentes possibilités d'accès à la base de données fédérée :

Accès direct Les anciennes applications ne doivent pas être modifiées, elles continuent à accéder directement aux anciennes bases de données. Dans ce cas, lorsque une information est modifiée, on s'arrange pour que le système de gestion de la base de données concernée prévienne le gestionnaire de la fédération. Le gestionnaire veille alors à la cohérence des données en modifiant si nécessaire le contenu des autres bases de données.

Accès via le gestionnaire de fédération Les nouvelles applications ou les applications modifiées peuvent faire appel à des informations contenues dans la base de donnée virtuelle que constitue la fédération. Ces applications font appel à un médiateur (couplé éventuellement avec le gestionnaire de la fédération) qui se charge de recomposer l'information souhaitée à partir des données stockées dans les différentes bases de données constituant la fédération.

L'architecture proposée figure 3.8 ajoute une couche *wrapper* entre les bases de données et le médiateur. Ces programmes ont pour but de standardiser, au niveau du gestionnaire de fédération, les accès aux différentes bases de données. De tels programmes sont écrits une fois pour toute pour un SGDB particulier, ils peuvent donc être réutilisée dans d'autres fédérations.

Le gestionnaire de la fédération quand à lui devra être adapté à chaque cas, certaines parties pouvant être générées automatiquement à partir du schéma conceptuel de la base de données fédérée.

Dans l'architecture présentée ci-dessus, toute l'intelligence se trouve au niveau du gestionnaire de la fédération de bases de données. Il n'est donc pas utile, lors de la mise en place d'une fédération, de passer en revue toutes les anciennes applications, il en découle un gain de temps conséquent et donc une réduction de coûts.

Un approche intéressante possible dans le cas d'une fédération de bases de données est la création de toutes pièces du schéma de base de données qui serait utile à une application. A partir de ce schéma, on essaye de trouver les différentes informations nécessaires dans différentes bases de données existantes et on génère les correspondances entres ces bases de données et le schéma créé. L'intérêt est de rester indépendant de la manière dont l'information est déjà stockée et donc d'éviter de réutiliser des défauts présents dans les bases de données en place.

Chapitre 4

Outils d'aide à l'implémentation de traducteurs

Ce chapitre a pour but d'exposer différentes technologies permettant de faciliter la tâche du programmeur lorsqu'il doit implémenter un traducteur. Parmi ces technologies, on peut citer :

Les outils méta-CASE qui offrent un environnement complet pour la conception d'éditeurs de schémas. Offrant par là même un ensemble de facilités utiles dans la programmation de traducteurs.

Les analyseur grammaticaux et analyseurs syntaxiques qui permettent d'analyser un document textuel en vue d'un traitement particulier.

Les documents XML qui permettent d'exprimer de manière textuelle n'importe quel arbre. Il existe un ensemble d'outils non négligeable permettant de manipuler de tels documents.

Certaines de ces technologies ont été utilisées durant le développement des différents traducteurs du projet MurMur. Les autres technologies seront probablement utilisées dans des développements futurs.

4.1 Les outils Méta-CASE

Les outils méta-CASE sont des outils permettant à un informaticien de créer des outils CASE. De par leur nature, ces outils offrent, outre l'environnement nécessaire à un outil CASE, des possibilités d'extension qui sont intéressantes à exploiter dans les traductions de modèles.

Je passerai tout d'abord en revue les fonctionnalités de base de ces outils, ensuite je me pencherai sur un exemple intéressant, l'outil "DB-Main" développé par le Laboratoire d'ingénierie des applications de bases de données des Facultés Universitaires Notre-Dame de la Paix de Namur.

4.1.1 Fonctionnalités de base

Les outils méta-CASE partent du principe qu'il existe un ensemble commun de fonctionnalités nécessaires à la réalisation d'un outil CASE. Ces fonctionnalités sont bien entendu implémentées dans l'outil méta-CASE et sont modifiables via un langage propre à chaque outil.

On peut trouver parmi ces fonctions de base :

Une gestion de l'interface graphique Le moteur graphique est la partie la plus visible d'un outil CASE puisqu'il est le lieu des interactions avec l'utilisateur. Un bon moteur graphique doit offrir une grande souplesse de personnalisation au programmeur mais aussi rester simple d'utilisation pour l'utilisateur final. Le moteur graphique est généralement accompagné d'une série de figures prédéfinies qui permettent de définir simplement et rapidement des nouveaux modèles.

Un ou plusieurs langages de programmation Les différents langages doivent permettre de définir un nouveau modèle, de définir l'apparence des objets graphiques, d'étendre l'outil avec de nouvelles fonctions (affichage de boîtes de dialogue, gestion de la logique sous-jacente ...)

Une librairie de modèles Il est nécessaire de pouvoir stocker les modèles déjà définis. Ils peuvent soit servir à l'utilisateur final qui les utilisera dans ses problèmes de modélisation.

Le fait de posséder un langage de programmation interne permet d'ajouter des fonctionnalités intéressantes aux outils CASE résultant tel les évolution de schémas ou la rétro-ingénierie.

A titre d'information, on pourra consulter quelques documents décrivant quelques-uns des outils disponibles. Par exemple, JKogge [4], DB-Main [8] ou encore ConceptBase.

4.1.2 DB-Main

DB-Main est un des outils Meta-CASE disponible sure le marché. Il m'a paru intéressant à plusieurs point-de-vue :

- Il existe depuis plusieurs années et est toujours développé. Le logiciel continue donc à être amélioré et est tenu à jour.
- Il a été conçu au départ pour pouvoir implémenter des modèles de base de données.

- Des extensions dans les domaines de la rétro-ingéniérie et de l'évolution de schémas(ce point a déjà été évoqué en 3.2) ont été ajoutées.
- L'outil est développé en Belgique, les contacts avec les concepteurs du projets en sont facilités.

Architecture

L'outil DB-Main est composé de deux couches :

Une couche basique implémentant toutes les fonctionnalités de base nécessaires au fonctionnement d'un outil CASE. Comme DB-Main s'adresse en particulier aux bases de données Objet/Relationnel, et afin d'optimiser quelque peu l'outil, les concepteurs ont décidé d'intégrer à la couche basique des fonctionnalités et des concepts communs à tous les modèles objets/relationnels.

Une couche dynamique (ou couche méta), qui se charge du stockage et de l'exécution des modèles définis par l'utilisateur. Cette couche est elle-même divisée en trois parties :

Le langage et la machine virtuelle Voyager2 C'est le langage fonctionnel de DB-Main. Il permet d'étendre les possibilités du programme. Ce langage utilise une syntaxe proche de celle du Pascal. Ce langage présente certaines caractéristiques propres à son objectif :
- Types étendus, tel le type lambda permettant de charger une fonction au moment de l'exécution.
- Gestion de la mémoire à l'aide d'un ramasse-miettes.
- Fonctions permettant d'interagir avec la gestion de l'interface graphique.
- Possibilité d'interfaçage avec d'autres langages tels C++.
- Chargement dynamique de modules.
- Typage dynamique.
- Constructions particulières pour pouvoir créer des requêtes dans l'ensemble des classes instanciées.

Le moteur graphique Grasyla Le moteur graphique se charge à la fois de la gestion de l'interface avec l'utilisateur et de l'interprétation du langage de description graphique des objets.

L'interpréteur de méthodes Les méthodes dont on parle ici sont des méthodes d'analyse, la manière d'utiliser un modèle. DB-Main possède un langage de description de méthodes appelé assez simplement MDL (Method Description Language). L'interpréteur est donc charger d'exécuter ce qui a été décrit par le concepteur du modèle. Pour plus d'informations concernant ce langage, on peut consulter [19].

La partie statique de DB-Main est reflétée en partie dynamique afin d'avoir un outil entièrement modifiable.

Rétro-Ingénierie

L'outil DB-Main intègre des possibilités de rétro-ingéniérie de bases de données. Cette partie de l'application permet de tenter de récupérer le schéma conceptuel d'une base de données à partir d'une base de données relationnelle, ou même à partir d'une basse de données hiérarchique, écrite en COBOL.

Je n'exposerai ici que le principe théorique permettant de passer du schéma physique au schéma conceptuel, le reste sortant de l'objectif de ce travail. On pourra trouver quelques informations complémentaires dans [10].

Pour reconstruire un schéma on part de l'application existante pour remonter vers le schéma physique, ensuite le schéma logique pour terminer par le schéma conceptuel. Ce cycle est exactement le cycle inverse du processus de modélisation.

Pour passer d'un schéma à l'autre, on utilise différentes transformations. Chaque transformation se doit de respecter la sémantique du schéma. Dans le cas de DB-Main, puisque le but final est de pouvoir modifier le schéma conceptuel, chaque transformation se doit aussi d'être réversible. Toutes ces transformations sont stockées afin de pouvoir les réappliquer en cas de modification du schéma.

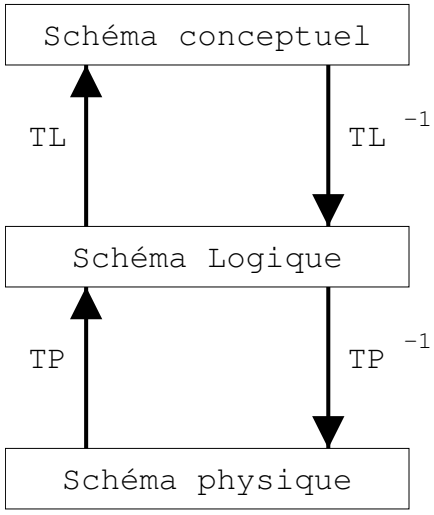

FIG. 4.1 – Rétro-ingéniérie

On peut ainsi voir à la figure 4.1, la remontée du schéma physique vers le schéma conceptuel. Les différentes opérations effectuées sur le schéma sont notées TP (transformation depuis le schéma physique) et TL (transformation depuis le schéma logique). Les transformations inverses sont respectivement notées TP^{-1} et TL^{-1}.

4.1.3 Conclusion

Les outils méta-CASE sont des outils très puissants pouvant toucher de nombreux domaines de d'ingénierie de bases de données. Des outils tels DB-Main qui intègrent de nombreux services connexes à l'édition de schéma permettent au concepteur de modèles de se concentrer sur son modèle et non sur la programmation de celui-ci.

A partir de ces outils, un ensemble de recherches ont été faites dans le domaine des transformations de schémas. L'étude de ces recherches permet de mieux comprendre les transformations. Il

faut les voir comme un ensemble d'opérateurs éventuellement réversibles. Ces opérateurs pouvant être classés selon la manière dont ils modifient la sémantique du schéma.

Les outils méta-CASE aident donc dans la conception, l'étude et l'utilisation de nouveaux modèles.

4.2 Analyseurs grammaticaux et analyseurs syntaxiques

Les analyseurs grammaticaux et analyseurs syntaxiques sont des outils classiques dans le monde informatique lorsque l'on tente d'analyser le contenu sémantique d'un fichier. On les utilise principalement dans les compilateurs, mais aussi dans toutes les applications qui demandent à l'utilisateur de rentrer de manière interactive des commandes.

Ces outils trouvent leur place dans le monde des bases de données lorsqu'il est nécessaire de traduire un ensemble de commandes vers un langage compréhensible par une machine. Ces commandes seront, par exemple, des commandes SQL de création de tables, des commandes de requêtes,. . .

Ils seront probablement utilisés dans les développements futurs (voir point 5) pour traduire, par exemple, des requêtes MADS vers un fichier XML. Le pourquoi de l'utilisation de fichiers XML sera abordée dans la section suivante(4.3).

4.2.1 Lex et Yacc

Parmi les grands classiques dans le mondes des analyseurs grammaticaux et analyseurs syntaxiques, on retrouve l'analyseur grammatical Lex et l'analyseur syntaxique Yacc. Pour une bonne introduction à l'utilisation de ces outils, on pourra consulter [11].

Ces deux outils ont été développés au départ pour des plates-formes Unix et pour le langage C, mais il existe maintenant des versions capables de créer des analyseurs grammaticaux et des analyseurs syntaxiques pour à peu près n'importe quelle plate-forme et n'importe quel langage.

Les versions GNU de ces outils(Flex[7] et Bison[6]), permettent d'exporter du code C ou C++ pour les plates-formes Unix, Windows, Macintosh, OS/2 . . .

Il existe d'autre part des versions spécialement écrites pour le langage Java par exemple.

4.3 Les documents XML

Les documents XML et les technologies connexes forment un ensemble émergent touchant de plus en plus de domaines. Basiquement, XML n'est qu'une manière de stocker des informations sous forme textuelle en y ajoutant une information sémantique traduite par des balises. Un exemple simple de document XML pourrait être :

<**livre**>
  <**auteur**> J.R.R Tolkien </**auteur**>
  <**titre**> Le Silmarillion </**titre**>
  <**chapitres**>
    <**titre**> Genèse </**titre**>
    <**titre**> Quenta Silmarillion </**titre**>
    <**titre**> Akallabeth </**titre**>
    <**titre**> Les Anneaux du Pouvoir et le Troisième Âge </**titre**>
  </**chapitres**>
</**livre**>

On peut remarquer les balises (en **gras**) fournissant une information sémantique sur le texte qu'elles entourent. Ainsi nous pouvons déduire, sans difficultés, de l'exemple précédent, que le titre du livre est "Le Silmarillion". Toujours en observant l'exemple, on peut remarquer que les balises peuvent contenir d'autre balises. Ainsi, le livre possède un ensemble de chapitres et l'ensemble de chapitres possède un ensemble de titres. Cette architecture de balises imbriquées permet de facilement définir des structures en arbre.

Les spécifications d'XML[23] sont une recommandation du W3C qui indique comment écrire un document correct. Ces indications sont assez simples, elles concernent la définition de ce qu'est une balise et de quelques autres éléments secondaires non abordés ici.

D'autres recommandations entourent la définition des documents XML. Ainsi, on peut trouver la définition d'un langage de parcours des documents, appelé XPATH[24].

Une autre recommandation intéressante est XMLSchema[25]. Cette recommandation détermine la manière de définir les balises utilisables et le contexte dans lequel elles peuvent être utilisées. L'utilisation d'un schéma avec l'exemple qui précède permettrait de définir que :

- tout livre doit comporter un titre
- tout livre doit comporter au moins un auteur
- que l'ensemble des chapitres est optionnel
- que l'ensemble des chapitres contient une série non vide de titres

On peut voir XMLSchema comme un outil de définition d'une grammaire dans le cadre des documents XML. On utilise ces schémas afin de valider un document, cela permet de s'assurer que le document est bien structuré comme on le souhaite et simplifie les traitements ultérieurs en enlevant une bonne partie de la gestion des erreurs.

Il existe un autre langage de définition de structure de document XML appelé DTD. Ce langage est plus ancien que XMLSchema, mais il est moins expressif ; il est donc appelé à être progressivement remplacé par des définitions XMLSchema.

Parmi les recommandations, on peut trouver des outils de manipulation et d'interrogation des documents XML. Outre ces recommandations, des outils plus particuliers ont vu le jour, se focalisant sur un type particulier de document ou sur un langage de programmation particulier. Quelques-uns de ces outils sont abordés dans les sections suivantes.

4.3.1 XSLT

XSLT[26] est un langage de transformation de documents XML en documents XML (bien qu'une sortie vers les formats texte et HTML soit possible). Le langage s'exprime lui-même dans la syntaxe XML en définissant un ensemble de balises particulières.

Une transformation XSLT décrit un ensembles de règles pour transformer un arbre XML. Ces règles sont constituées d'un modèle et d'un ensemble d'actions. La transformation s'effectue en essayant d'appliquer les modèles à l'arbre d'entrée. S'il y a une correspondance entre l'arbre d'entré et le modèle, l'ensemble d'actions associé au modèle est appliqué à l'arbre d'entrée. Ces actions soit génèrent une partie de l'arbre de sortie, soit tentent d'appliquer les modèles disponibles à un sous-arbre généré à partir de l'arbre d'entrée. L'arbre de sortie est indépendant de l'arbre d'entrée, mais généré à partir des règles appliquées à l'arbre d'entrée.

XSLT excelle dans les transformations simples, par exemple, renommer un noeud, ajouter, supprimer des noeuds, réordonner des noeuds se situant sur un même niveau, ... XSLT est idéal lorsqu'il ne faut parcourir les données qu'une seule fois.

Lorsqu'il est nécessaire de recourir à des parcours multiples dans la structure, il devient difficile d'exprimer les transformations à partir d'XSLT (bien qu'une nouvelle version de ce langage soit actuellement à l'étude). On recourt dans ces cas-là à des langages de programmation plus classique tels C, C++, Java aidés par une librairie de programmation XML, par exemple DOM qui sera abordée au point suivant.

XSLT n'en reste pas moins un outil utile, ne serait-ce que pour la communication entre logiciels ou la présentation des données.

4.3.2 DOM

DOM[22] est encore une spécification décrite par le W3C. Il s'agit d'une interface de programmation, basée sur des fonctions permettant de manipuler des documents XML.

La particularité de cette interface est d'avoir été définie dans un langage abstrait et d'avoir ensuite été traduite vers des langages de programmation courant. Ces traductions sont standardisées et on retrouve ainsi des implémentations de DOM dans différents langages, l'interface de programmation restant la même (à l'exception des modifications nécessaires afin de conserver une syntaxe correcte)quelque soit le langage utilisé. Comme cette interface est standardisée, elle est largement

diffusée et utilisée. On peut donc trouve, pour les langages supportés des implémentations fiables de DOM.

A partir de cette interface, il est possible de :
- Créer un arbre complet, ou un sous-arbre
- Ajouter et supprimer des noeuds dans un arbre
- Naviguer dans l'arborescence
- Récupérer les valeurs stockées dans les noeuds

Toutes ces manipulations sont faites dans le langage de programmation choisit par le programmeur. Ces manipulations peuvent donc être combinées avec d'autres traitements possible dans le langage choisit. C'est avec une implémentation Java de DOM que les différents traducteurs du projet MurMur ont été implémentés.

DOM n'est cependant pas la réponse à tous les problèmes de manipulation de documents XML. En effet, sa très grande généralité ne lui permet pas d'utiliser les finesses du langage dans lequel il est implémenté. DOM demande parfois d'écrire des lignes de code complexes, alors qu'il existe des mécanismes plus simples, mais particuliers à un langage. C'est pour cette raison que des interfaces de programmation différentes ont été développées entre autre pour le langage Java ; trois d'entre-elles seront décrites dans la section suivante.

4.3.3 Outils Java

Java[12] est un langage de programmation développé par Sun Microsystems. Java est un langage orienté objets épuré des concepts mal définis ou posant problèmes, comme par exemple l'héritage multiple.

Java est aussi une plate-forme de développement pour laquelle un ensemble étendu de librairies de programmation a été défini et standardisé. Ces librairies sont par ailleurs fournies en standard avec les outils de développement Java distribués par Sun Microsystems. Parmi les différentes librairies, on retrouve une gestion d'interfaces graphiques, des systèmes de connexion aux réseaux et une implémentation de DOM.

Java est encore une plate-forme d'exécution ; en effet, un programme écrit en Java est compilé pour une machine virtuelle et non pour une machine réelle. Le code ainsi compilé est appelé "bytecode". Ce bytecode ne pouvant être exécuté directement sur l'ordinateur de l'utilisateur, on a recours à une machine virtuelle capable de traduire le bytecode en un code compréhensible par l'ordinateur.

Cette manière de faire est un équilibre entre la portabilité d'un langage interprété et la vitesse d'exécution un langage compilé pour la plate-forme cible. Elle permet d'avoir des applications tournant sur n'importe quelle plate-forme pour laquelle une machine virtuelle Java a été développée. Parmi ces plate-forme, on retrouve : Windows, Linux, Solaris et MacOS. Autrement dit, les plates-formes les plus utilisées aujourd'hui.

Comme annoncé ci-dessus, des interfaces de programmation alternatives à DOM on été spécialement écrites pour le langage Java. On retrouvera parmi elles JDom et dom4j, deux interfaces concurrentes et JAXB une manière originale d'aborder l'utilisation des documents XML dans une application.

JDom

JDom[2] est un projet libre visant à réaliser une interface proche de celle de DOM, mais utilisant les propriétés du langage et de la plate-forme Java.

Au point de vue de l'implémentation, JDOM utilise des classes Java plutôt que les interfaces préconisées par DOM. Cela entraîne une plus grande facilité d'utilisation pour un programmeur habitué au langage.

JDom utilise aussi les "Collections", un ensemble de classes prédéfinies, dans les librairies Java, permettant de manipuler des ensembles d'objets de manière simple et rapide. L'utilisation des "Collections" ajoute, à l'usage, simplicité de programmation et vitesse d'exécution.

dom4j

dom4j[1] est également un projet libre, démarré à partir de JDom. L'objectif de dom4j est double :

- Améliorer le plus possible la flexibilité. Pour ce faire, à l'inverse de JDom, dom4j utilise des interfaces avec une implémentation sous-jacente par des classes.
- Ajouter un ensemble d'outils connexes à DOM : support du langage XPath, support de grands documents, ...

L'interface de programmation de dom4j est donc plus complexe que celle de JDom, mais elle offre plus de fonctionnalités. Au point de vue des fonctionnalités communes, les deux API sont aussi simples l'une que l'autre et se ressemblent fortement. On pourra trouver dans la bibliographie, une comparaison entre ces deux outils au point de vue facilité de programmation [14], ainsi qu'une comparaison au point de vue des performances [13].

Aucun de ces deux outils n'a été utilisé lors des développements du projet MurMur, tant pour des raisons de standardisation entre les différents partenaires, que parce que ces outils sont encore en développement et que leur API varie d'une version à l'autre.

Les API s'étant stabilisées depuis, ces outils peuvent être envisagés sans craintes pour des développements futurs.

JAXB

Contrairement au deux outils précédents, JAXB n'est pas une autre forme de DOM. Le but de JAXB est d'arriver à transformer le contenu d'un fichier XML en une structure utilisable via le langage Java, tout en conservant la sémantique du fichier initial.

A partir de l'exemple de document XML 4.3, on peut représenter (voir figure 4.2) les structures de données utilisées par des outils tels que DOM.

Si on compare cette structure avec celle réalisée par JAXB (voir figure 4.3), on remarque tout de suite l'intérêt de cette solution.

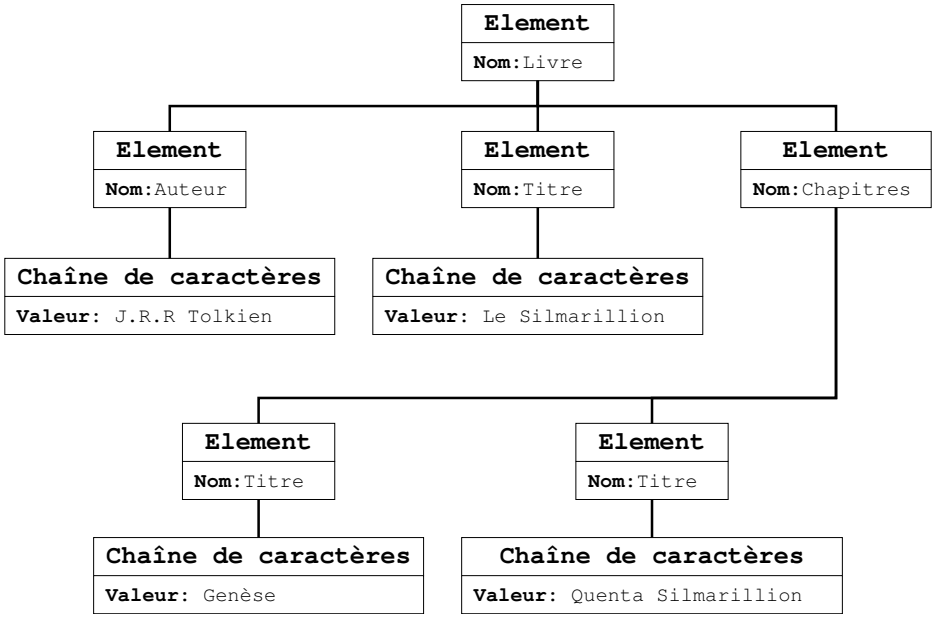

FIG. 4.2 – Représentation DOM du document XML 4.3

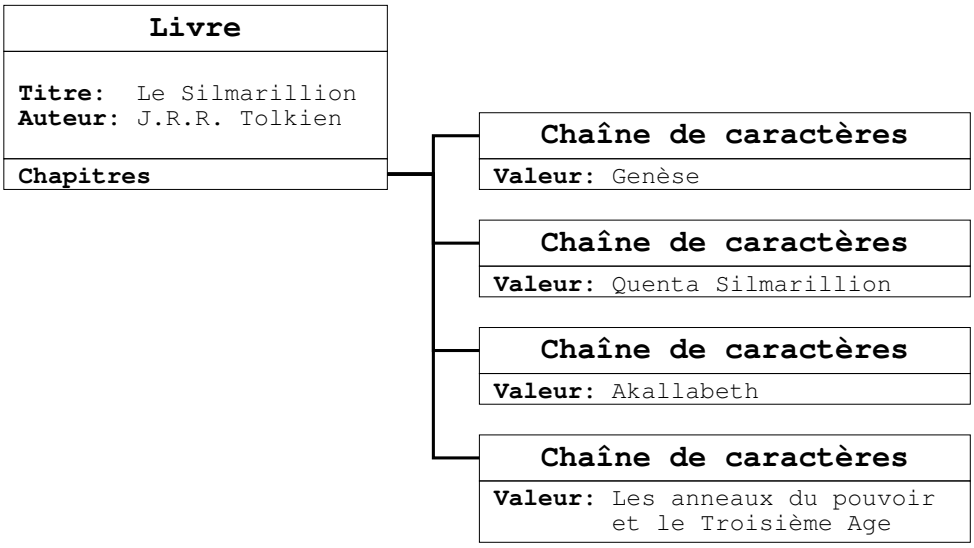

FIG. 4.3 – Représentation créée par JAXB du document XML 4.3

La nouvelle structure créée est similaire à celle qu'aurait créé un programmeur Java s'il avait dû l'implémenter directement. La structure ainsi créée est immédiatement utilisable dans le langage sans plus aucune référence à la structure du document qui contenait l'information.

JAXB est en réalité un générateur de code qui se base sur deux fichiers d'entrée : le schéma du document XML et un fichier de correspondance. Ce dernier permet de paramétrer la manière dont la traduction s'effectue et de modifier la traduction pour arriver à un code correspondant plus à une structure idéale au sens Java du terme. Cet outil rejoint donc la manière traditionnelle de traiter les fichiers comme un moyen de stockage de la structure représentée en mémoire. L'originalité du projet vient de l'utilisation du fichier pour définir la structure mémoire et de l'automatisation du processus.

Cet outil bien qu'étant très puissant est hélas encore en développement. La version actuelle n'accepte en entrée que des définitions par DTD et non par XMLSchema, privant par là les utilisateurs d'une part d'expressivité.

Il est à espérer qu'un tel outil arrivera rapidement à maturité afin de pouvoir être utilisé dans des projets à long terme. L'idée sous-jacente de ce projet, de représenter les documents XML en gardant leur sémantique, est intéressante et à exploiter même sans outil de génération automatique de code.

Chapitre 5

Recherches futures

Des divers exemples de traduction présentés dans ce travail, on peut déduire que :
- Les processus de traduction sont fortement présents dans la modélisation de bases de données.
- La manière de traduire un concept vers un autre est souvent bien connue.
- Les traductions des concepts d'un modèle vers un autre font en général appel à une série de règles simple et similaires, se combinant les unes aux autres.

Malgré tous ces développements autour des traductions, il n'existe que peu ou pas d'outils pouvant aider le programmeur à créer assez aisément des traducteurs.

Les recherches futures auront dès lors comme but pratique la création d'outils d'aide à la génération de traducteurs. Afin de simplifier au maximum le travail du futur utilisateur d'un tel outil, il sera nécessaire de décrire les différents éléments nécessaires à traduire un concept vers un autre. Une partie plus théorique consistera donc à créer un modèle permettant la description de ces différents éléments.

5.1 Recherche théorique

Les recherches futures pouvant être effectuées dans le domaine des traducteurs s'orienteront autour de deux axes. Le premier axe est la recherche de moyens pouvant décrire de manière globale la traduction d'un modèle vers un autre. Le second sera la recherche d'un modèle de description des traducteurs.

5.1.1 Premier axe

Le premier axe de recherche s'orientera vers l'étude du processus de traduction dans sa globalité. La traduction ne sera plus envisagée comme un ensemble de modules permettant de traduire les schémas, les requêtes, les contraintes d'intégrité, ... d'un modèle vers un autre. La traduction sera envisagé comme un ensemble de règles permettant de traduire un concept et les fonctionnalités entourant ce concept.

Si l'on reprend le modèle MADS que l'on voudrait traduire vers le modèle relationnel, une règle de traduction pourrait concerner les attributs de types intervalle. L'étude de cette règles comportera entre autre :

- La traduction de la structure d'un intervalle vers une table relationnelle (point de vue du schéma)
- La traduction des requêtes devant retourner cet intervalle comme résultat (point de vue des requêtes)
- La traduction des fonctions acceptant des intervalles comme paramètres, par exemple les fonctions : intersection, coïncidence, précède, suit, ... (point de vue fonctionnel)
- La traduction des contraintes d'intégrités faisant intervenir des intervalles (point de vue contraintes)

L'étude de cette règle fera apparaître des liens entre les différents point de vue.

Le premier axes tentera de dégager, à partir de différents cas pratiques, les liens unissant les différentes parties d'une règle de traduction. Le premier cas pratique envisagé sera le modèle MADS puisque les règles de traductions de ce modèle ont été développées par le service informatique de la faculté des sciences appliquées.

5.1.2 Second axe

Le deuxième axe de recherche s'articulera autour de la création d'un modèle de description des traducteurs entre modèles de bases de données. Cette partie du développement théorique sera fortement liée aux développements pratiques visant à implémenter des outils d'aide à la création de traducteurs.

Le modèle ainsi créé devra proposer :

- Une manière de décrire le fonctionnement global d'un traducteur
- Une manière de décrire l'enchaînement des règles composant ce traducteur
- Une manière de décrire les règles utilisées par le traducteur

Plusieurs formalismes sont à envisager comme base pour la conception d'un tel modèle. Parmi ceux-ci on trouve :

- Les diagrammes d'état-transition ; ces diagrammes permettent de décrire un enchaînement d'actions avec d'éventuelles bifurcations si nécessaire.
- Les "Use Case" d'UML qui peuvent servir de base pour la définition des règles de traduction.
- Les opérations mathématiques qui permettraient de voir les règles comme des opérateurs agissant sur un objet. L'enchaînement des règles pourrait alors être vu comme une composition de ces opérateurs.

Parallèlement au modèle de description visuel des traducteurs, un langage informatique de description des traducteurs devra être développé. Ces langage sera utilisé dans le développement des outils de création de traducteurs.

5.2 Applications pratiques

A partir des recherches théoriques décrites ci-dessus, il devrait être possible d'implémenter des outils d'aide à la conception et à la programmation de traducteurs de modèles.

Ces outils devraient idéalement répondre à certains critères, parmi lesquels :
- Rester indépendant de la plate-forme informatique utilisée
- Utiliser des bibliothèques standard et de préférences libres
- Être rapides, efficaces et d'utilisation aisée
- Le code généré devrait pouvoir être exprimé dans un ensemble de langages courants tels C/C++ et Java.
- Rester le plus modulaire possible et proposer un système simple d'extension
- Être totalement documentés

Le développement de ces outils se fera de manière itérative, en commençant par l'implémentation de fonctions minimales et par ajout successifs de fonctionnalités. Cette approche permettant de coupler une partie de l'étude théorique avec le développement des outils et ainsi de valider par la pratique les options choisies.

5.3 Conclusion

Les différents développements proposés ci-dessus permettraient aux programmeurs de gagner du temps lors de la création de traducteurs de modèles. Ces développements permettraient aussi une mise en place plus rapide de nouveaux modèles plus adéquats à certaines situations.

Les outils tels qu'envisagés ici sont orientés dans un premier temps vers la traduction entre modèles de bases de données, mais ils pourraient s'appliquer par extension à d'autres domaines de l'informatique.

Lexique

API
Application Programming Interface
Abréviation anglophone pour “Interface de programmation d’une application”.

CASE
Computer Aided Software Engineering.

DOM
Document Object Model
Interface de programmation standardisée pour la manipulation de documents XML.

DTD
Document Type Definition
Langage de spécifications de structure pour des documents XML.

GNU
GNU is Not Unix
Projet de développement en sources libres d’un système Unix complet et des outils de développement nécessaires.

UML
Unified Modeling Language
Modèle créé pour pouvoir décrire des applications en utilisant les concepts de l’Orienté Objet.

SGDB
Système de Gestion de Bases de Données.

SQL
Structured Query Language
Langage de requêtes standardisé pour les bases de données relationnelles.

W3C
World Wide Web Consortium
Organisme chargé de la définition des standards utilisés sur internet.

XML
eXtended Markup Language

Formalisme de stockage de données qui se base, à l'instar du format HTML, sur du texte dans lequel est inséré un ensemble prédéfini de balises.

XSLT

XML Stylesheet Transformation

Langage de transformation des documents XML

Table des matières

Liste des figures

Liste des tableaux

Bibliographie

[1] dom4j.
URL : `http://dom4j.org`.

[2] Jdom.
URL : `http://jdom.org`.

[3] Richard T. Snodgrass (chair), Ilsoo Ahn, Gad Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna Kulkarni, T. Y. Cliff Leung, Nikos Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo, and Suryanarayana M. Sripada. Tsql2 language specification, September 1994.

[4] Jürgen Ebert, Roger Süttenbach, and Ingar Uhe. Jkogge : a component-based approach for tools in the internet. In *Proceedings STJA '99*, Erfurt, September 1999.
URL : `http://www.uni-koblenz.de/~ist/retrieve/stja99.ps.gz`.

[5] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings, 1999.

[6] Free Software Foundation. Bison.
URL : `http://www.gnu.org/software/bison`.

[7] Free Software Foundation. Flex.
URL : `http://www.gnu.org/software/flex`.

[8] Jean-Luc Hainaut and Vincent Englebert. Db-main : A next generation meta-case. *Information Systems Journal*, June 1999.
URL : `http://www.info.fundp.ac.be/~ven/PAPERS/jis.pdf`.

[9] Jean-Marc Hick. *Évolution d'Applications de Bases de Données Relationnelles. Méthodes et Outils*. PhD thesis, FUNDP, 2001.

[10] Jean-Marc Hick, Jean-Luc Hainaut, Vincent Englebert, Didier Roland, and Jean Henrard. Stratégies pour l'évolution des applications de bases de données relationnelles : l'approche db-main. In *XVIIe congrès INFORSID*, 1999.
URL : `http://www.fundp.ac.be/recherche/publications/fr/35840.html`.

[11] Bert Hubert. Lex and yacc primer/howto, 2002.
URL : `http://ds9a.nl/lex-yacc`.

[12] Sun Microsystems. Java programming language.
URL : `http://java.sun.com`.

[13] Dennis M.Sosnoski. *XML in Java : Document models, Part1 : Performance*. IBM developerWorks.
URL : `http://www-106.ibm.com/developerworks/xml/library/x-injava`.

[14] Dennis M.Sosnoski. *XML in Java : Document models, Part2 : Usage*. IBM developerWorks.
URL : http://www-106.ibm.com/developerworks/xml/library/x-injava2.

[15] Christine Parent. Workpackage 3 - deliverable 8 : Query language specification. Technical report, MurMur Project, 2001.
URL : http://www.mur-mur.org/download/MM-WP3-DLA-008-V6_Deliverable8.zip.

[16] Christine Parent and Stefano Spaccapietra. Issues and approaches of database integration.

[17] Christine Parent, Stefano Spaccapietra, and Esteban Zimányi. Murmur : Database management of multiple representations, 2000.
URL : http://lbdsun.epfl.ch/e/publications/articles.pdf/AAAI-STgranularity.pdf.

[18] Christine Parent, Esteban Zimányi, Mohammed Minout, and Ali Aissaoui. *Traité IGAT sur la représentation multiple et la généralisation*, chapter Implantation d'un modèle conceptuel avec multi-représentation. 2002.

[19] Didier Roland and Jean-Luc Hainaut. Database engineering process modeling. In *Int Conference on The Many Facets of Process Engineering*, 1997.
URL : http://www.fundp.ac.be/recherche/publications/fr/39230.html.

[20] Stefano Spaccapietra. Workpackage 2 - deliverable 5 : Data model specification. Technical report, MurMur Project, 2001.
URL : http://www.mur-mur.org/download/MM-WP2-DLA-005-V4_Deliverable5.zip.

[21] Ph. Thiran and J.L. Hainaut. Interoperability of legacy databases, 2000.

[22] W3C. Documents object model (dom).
URL : http://www.w3.org/DOM.

[23] W3C. Extensible markup language (xml).
URL : http://www.w3.org/TR/REC-xml.

[24] W3C. Xml path language (xpath).
URL : http://www.w3.org/TR/xpath.

[25] W3C. Xml schema.
URL : http://www.w3.org/XML/Schema.

[26] W3C. Xsl transformations (xslt) version 1.1.
URL : http://www.w3.org/TR/xslt11.