UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

Visualisation de requêtes SOLAP : Une approche services Web

Mémoire présenté en vue de l'obtention
du grade de Licencié en Informatique

Grégory BABUSIAUX
Année académique 2006–2007

Remerciements

Je tiens tout d'abord à remercier mon directeur de mémoire, Monsieur Esteban Zimányi, pour sa disponibilité et les nombreux conseils prodigués.

Je tiens ensuite à saluer les membres du jury, les Professeurs Gianluca Bontempi et Joël Goossens, pour le temps qu'ils passeront à lire ce travail.

Merci à Géraldine, pour ses nombreuses relectures et corrections. Plus encore, merci pour ton soutien, ton optimisme et ton réconfort.

Enfin, j'adresse mes plus vifs remerciements à mes parents pour leur relectures, mais surtout pour leur support, indéfectible, lors de toutes ces années. Les parents ne peuvent donner que deux choses à leurs enfants : des racines et des ailes. Vous avez excellé. Merci.

Table des matières

1 Introduction **1**

2 Les entrepôts de données **3**

2.1 Introduction 3

2.2 L'informatique décisionnelle 3

2.2.1 L'informatique décisionnelle spatiale 5

2.3 Les entrepôts de données 6

2.3.1 Historique 6

2.3.2 Présentation 7

2.3.3 Définition 8

2.3.4 Architecture 9

2.3.5 Agrégation des données 10

2.3.6 Opérations sur les entrepôts de données 11

2.4 Conclusions 14

3 Systèmes SOLAP **15**

3.1 Introduction 15

3.2 GIS 15

3.2.1 Définition 15

3.2.2 Motivations 16

3.2.3 Architecture 18

3.2.4 Modélisation des données 18

3.2.5 Outils GIS 19

3.3 OLAP 19

3.3.1 Génèse des systèmes OLAP 19

3.3.2 Types d'OLAP 21

3.3.3 Outils OLAP 23
3.4 SOLAP 23
3.4.1 Présentation 23
3.4.2 Définition 24
3.4.3 Vision 24
3.5 Conclusions 25

4 Le méta-langage XML **27**
4.1 Introduction 27
4.2 Historique 27
4.3 Principes de base 28
4.4 Motivations 29
4.5 Le document XML 29
4.6 Validation d'un document XML 31
4.6.1 Le DTD 32
4.6.2 Le langage XML Schema 33
4.7 Les analyseurs XML 35
4.7.1 XSLT 37
4.8 Conclusions 38

5 Le langage GML **39**
5.1 Introduction 39
5.2 Présentation 39
5.3 Description 41
5.4 Evolutions 44
5.5 Le mapping 45
5.6 Critiques du langage 45
5.6.1 Qualités 45
5.6.2 Défauts 46
5.7 Conclusions 46

6 Les outils utilisés **47**
6.1 Introduction 47
6.2 Le GML Viewer 47
6.2.1 Présentation 47

6.2.2 Conception d'ensemble 50
6.2.3 Fichiers principaux 51
6.2.4 Choix en faveur de GML Viewer 62
6.3 La suite Oracle 64
6.3.1 Présentation d'Oracle 64
6.3.2 Oracle 10*g* 65
6.3.3 Oracle Spatial 66
6.4 Conclusions 67

7 Mise en oeuvre 68
7.1 Introduction 68
7.2 Présentation du travail effectué 68
7.3 Fichier de données XML 69
7.4 Export en XML dans Oracle 70
7.4.1 Extraction de données en XML 70
7.4.2 Génération du fichier de données XML 72
7.5 Préparation du GML Viewer 73
7.5.1 Création d'un modèle propre 73
7.5.2 Définition des couches en GML 74
7.6 Transformation du fichier source en HTML 75
7.7 Interaction avec le GML Viewer 80
7.8 Limites 84
7.9 Conclusions 85

8 Conclusions et développements futurs 86
8.1 Conclusions 86
8.2 Développements futurs 87

Bibliographie 89

A Liste des acronymes rencontrés 92

B Exemples du chapitre 4 94
B.1 Les DTD's 94
B.2 XML Schema 95

B.3 Feuille de style XSLT 97

C Mise en oeuvre **98**

C.1 Feuille de style XSLT 98

C.2 Fonctions javascript 100

C.3 Fichier GML de description des frontières. 101

C.4 Fichier de données généré 103

C.5 Feuille de style CSS 104

C.6 Fichier config.xml 105

Chapitre 1

Introduction

Ce mémoire traite de l'informatique décisionnelle et, plus précisément, des systèmes d'aide à la décision. Ces systèmes revêtent aujourd'hui une importance particulière dans le monde informatique, ce qui rend ce travail tout à fait attrayant.

Le but de ce mémoire est d'imaginer une solution pour l'affichage de données provenant d'un système Oracle. Cette solution devra comprendre une visualisation des données rehaussée d'une carte géographique. Ce type de solutions est généralement appelé solution "SOLAP[1]".

Etrangement, Oracle ne dispose pas à ce jour de solution SOLAP proprement dite. Il s'agira d'une solution "simple", non-destinée à l'affichage de données proprement dites sur la carte (via des graphiques, bâtons ou camemberts par exemple) ; ces solutions faisant généralement appel à des logiciels propriétaires.

Dans le cadre de ce travail, nous nous contenterons d'afficher les données à côté d'une carte et nous établirons un lien entre carte et données. Un clic de souris sur les données provoquera l'apparition sur la carte de la zone géographique concernée. Cette réalisation fait appel à de nombreux concepts et techniques, que nous détaillerons dans ce mémoire.

Le chapitre deux sera consacré à une brève présentation de l'informatique décisionnelle, et plus particulièrement à ce que l'on appelle les "systèmes d'aide à la décision". Nous présenterons leur utilité et expliquerons les raisons de leur vif gain d'intérêt dans le monde de la *Business Intelligence* de nos jours. Nous brosserons le portrait des architectures que ces systèmes sous-tendent et évoquerons une sous-catégorie de ces systèmes, les SDSS ou systèmes d'aide à la décision spatiaux.

Ensuite, nous présenterons les entrepôts de données, qui sont une technique de stockage de données, bien souvent complémentaire aux bases de données transactionnelles classiques. Ces entrepôts sont très couramment utilisés dans les systèmes spatiaux d'aide à la décision.

[1] Afin d'aider le lecteur, un récapitulatif des acronymes cité dans ce mémoire est proposé en première annexe.

Le troisième chapitre présentera les systèmes SOLAP, qui permettent la représentation de données dans un contexte spatial. Pour ce faire, nous introduirons les systèmes GIS dont ils sont en partie issus, et nous présenterons les systèmes OLAP, fournissant des outils dédiés à l'analyse de données stockées sous forme multi-dimensionnelle. Nous énoncerons succinctement les différents types d'OLAP existants et présenterons ensuite les systèmes SOLAP proprement dits.

Le quatrième chapitre traitera du méta-langage XML. Ce langage nous sera utile à maintes reprises dans le cadre de ce mémoire. Nous allons en effet représenter nos données en XML, et ensuite utiliser plusieurs langages dérivés du XML, tels que XML Schema, XSLT, et GML, qui est écrit dans une syntaxe XML.

Dans ce chapitre, nous détaillerons donc le XML et les langages dérivés dont nous nous servirons dans la réalisation de notre travail.

Le cinquième chapitre sera consacré au langage GML, utilisé dans le cadre de ce mémoire pour afficher, sur une carte, la représentation de la zone géographique concernée par les données. Un bref historique sera présenté, et nous évoquerons quelques-unes des possibilités offertes par ce langage.

Le sixième chapitre sera consacré au logiciel utilisé pour effectuer notre représentation sur carte. Il s'agit d'un GML Viewer s'exécutant au sein d'un navigateur internet. De plus, nous présenterons brièvement la suite d'outils Oracle dont nous nous sommes servis dans le cadre de ce mémoire.

Le septième chapitre présentera les différentes implémentations effectuées au cours de ce travail. Nous illustrerons à l'aide de brefs morceaux de code les différentes techniques mises en place.

Le huitième et dernier chapitre énoncera une conclusion sur le travail effectué, et émettra quelques hypothèses quant aux possibles développements futurs.

Chapitre 2

Les entrepôts de données

2.1 Introduction

Dans ce chapitre, nous présenterons l'informatique décisionnelle ; nous tenterons de décrire à quel point elle est devenue incontournable dans le monde de la "*Business Intelligence*"[1].

Nous présenterons un outil mis en place afin de répondre aux besoins grandissants dans ce domaine, à savoir les entrepôts de données, en anglais : "*Data Warehouses*". Nous expliquerons leurs concepts, et illustrerons par des exemples leurs diverses possibilités.

2.2 L'informatique décisionnelle

En préambule à ce mémoire, nous allons vous présenter une branche en plein essor dans l'industrie informatique : l'informatique décisionnelle. Concrètement, il s'agit de ce que l'on appelle les systèmes d'aide à la décision ou "*Decision Support Systems*", également appelés DSS. Nous utiliserons à l'avenir cet acronyme pour désigner ces systèmes.

Les DSS peuvent prendre des formes très variées. On peut dire qu'il s'agit d'un système informatisé qui aide à la prise de décisions efficaces. En effet, via des collections et accumulations de données, les DSS permettent d'avoir une vision claire du passé, et donc d'estimer avec plus précision et de sûreté le futur. Cette accumulation de données, et surtout, les requêtes que l'on peut effectuer sur celles-ci, permettent d'obtenir des informations très précieuses sur un secteur d'activité précis, concernant une période donnée, dans un laps de temps choisi ...

Cette branche de l'informatique revêt donc une importance primordiale. Nombreux sont les secteurs demandeurs. Il est particulièrement intéressant pour les décideurs de pouvoir mettre en exergue des tendances, qui par ailleurs ne

[1] Aucune traduction n'a été jugée satisfaisante

pourraient pas apparaître sans l'aide de ces systèmes. En effet, c'est en interrogeant de gros volumes de données que nous pouvons dégager des faits. Et c'est l'agrégation de ces données et les outils permettant de les interroger qui rendent possible l'apparition de ces tendances.

Il serait absolument impossible d'avoir une analyse aussi fine du passé sans ces outils. Or, c'est grâce à ces analyses, que nous pouvons émettre des hypothèses sur le futur, et ce à court terme ou à long terme selon le secteur d'activité. Il est aisé de comprendre qu'une entreprise commerciale ne s'aviserait pas à émettre des hypothèses d'évolution de marché sur dix ans, si son domaine est l'informatique par exemple. Il s'agit là d'un secteur en constante évolution, et il serait bien trop dangereux de se baser sur le passé pour faire des projections sur dix ans. Par contre, ces mêmes entreprises peuvent tout à fait cibler leur marketing sur base de l'analyse des différentes évolutions du passé.

Mais, dans le secteur de la recherche climatologique, par ailleurs grande utilisatrice de ce genre de systèmes, l'étude des données climatologiques passées peut conduire à des hypothèses sur le relativement long-terme, partant du principe qu'il s'agit là de phénomènes cycliques dans le temps. De gros budgets ont été consacrés dans ce domaine à l'informatisation de toutes les données récoltées ces dernières décennies, afin d'arriver à des modèles de simulation puissants.

Dans [1], Efraïm Turban, professeur de systèmes d'informations à l'université d'état de Californie, décrit les DSS comme :

> "*un système d'information informatisé interactif, flexible, et adaptable, spécialement développé pour apporter une solution à un problème de management non-structuré pour une meilleure aide à la décision.*"

Les recherches portant sur les DSS remontent aux années soixante. Les DSS trouvent leurs origines dans les recherches sur les décisions organisationnelles et sur les systèmes informatiques interactifs. Le concept est devenu un champ de recherche à part entière dans les années septante, et a connu un vif gain d'intérêt dans le courant des années quatre-vingt[2]. L'avènement des entrepôts de données et de l'*On-line Analytical Processing* (OLAP) n'a fait qu'accélérer ce mouvement. Nous reviendrons ultérieurement sur ce qu'est OLAP.

On peut distinguer plusieurs types de DSS, tels que décrits dans [2] :

DSS basés sur un modèle : manipule des données statistiques, financières, ou provenant d'une simulation. Il utilise des données fournies par l'utilisateur pour assister la prise de décision.

DSS basé sur la communication : plusieurs personnes travaillant sur une même tâche aident à la prise de décision.

DSS basé sur les données permet, via la collection de données en grosse quantité, des comparaisons et suppositions aidant aux décisions.

[2] `http://dssresources.com/history/dsshistory.html`

DSS basé sur les documents gère, retrouve et manipule des informations non structurées sur support électronique et permet une utilisation efficace de celles-ci.

DSS basé sur la connaissance favorise la résolution de problèmes à partir des faits, règles, procédures déjà établies et connues.

Dans ce mémoire, nous traiterons des DSS basés sur les données. Et plus précisément des SDSS (*spatial Decision Support Systems*).

2.2.1 L'informatique décisionnelle spatiale

La recherche portant sur les SDSS est développée en parallèle aux DSS. Elle touche plus précisément à la dimension spatiale des données récoltées. Un système de décision spatial est généralement constitué de trois éléments :

1. Un système de gestion de la base de données qui gère les données géographiques, qu'on appelle GIS (*Geographical Information System*), nous y reviendrons plus tard.
2. Des modèles potentiels permettant de prévoir les conséquences des décisions à prendre.
3. Une interface aidant les utilisateurs à interagir avec le système et à apprécier les résultats des analyses effectuées.

Un cas concret. Voici un cas concret illustrant l'importance des SDSS aujourd'hui et ne laissant aucun doute sur l'intérêt croissant porté à ce domaine de recherche. Il s'agit de ce qu'on appelle le marketing ciblé (*target marketing*). Le domaine de recherche de ce mémoire se prête particulièrement bien à cet aspect du commerce.

Une entreprise peut collecter, au grès des mois qui passent, toutes sortes de données relatives à ses clients : quel produit est vendu, dans quelle région, en quelle quantité, au cours de tel ou tel mois de l'année ...

Intéressons-nous maintenant à la dimension spatiale de ces données. Imaginons une société basée aux Etats-unis et évoluant dans le domaine du tourisme. Elle s'implante au Canada voisin, mais ne mesure pas les grandes disparités entre francophones et anglophones. Les québécois subissent des hivers particulièrement pénibles et ont pour habitude de partir une semaine dans les Antilles pour "couper" leur hiver. Pendant leurs vacances estivales, ils sont généralement plus friands de culture que les anglophones. Ils ont par exemple plus tendance à prendre des "pass musées" dans les villes qu'ils visitent. Après quelques mois de présence sur le marché canadien, notre entreprise peut consulter ses données, et éventuellement isolera le Québec des provinces anglophones. Les différentes habitudes de consommation en matière de tourisme seront alors évidentes. La société pourra alors développer, au Québec, un marketing ciblé, approprié et donc différent du reste du Canada.

Nous ajouterons que les DSS sont un domaine d'étude particulièrement passionnant, relativement récent, et se prêtent donc bien à un mémoire, tant les technologies sont novatrices, et les lacunes à combler encore bien présentes.

2.3 Les entrepôts de données

2.3.1 Historique

Le modèle de données relationnel, qui a été introduit par E.F. Codd en 1970 et qui lui a valu de remporter le Turing Award dix ans après, a servi de fondation à l'actuelle industrie des bases de données, pesant plusieurs milliards de dollars [3]. Une base de données relationnelle est vue comme un ensemble de type entité/relation que l'on peut interroger via le language SQL (*Structured Query Language*) notamment. Les requêtes soumises aux bases de données sont généralement du type : "lister les clients habitants Ixelles" par exemple. Parmi les grosses sociétés du secteur, on peut citer Oracle ou Microsoft qui, via sa suite Microsoft Office, distribue le programme Access de création et d'administration de bases de données ou encore son SQL server, pour des solutions plus professionnelles. Les transactions s'effectuent à l'aide de clés primaires et étrangères qui relient les différentes tables entre elles. Ceci permet d'éviter la redondance des informations dans les tables, réduit donc l'espace de stockage nécessaire et permet surtout d'effectuer des requêtes dont les réponses seront rapides, ce qui est absolument primordial au vu des milliers de transactions effectuées chaque jour sur ces données.

Cependant, il manquait à ces bases une dimension essentielle, celle du temps. En effet, les systèmes relationnels classiques n'étaient jamais qu'un reflet de l'état actuel de l'entreprise. La liste de clients courants, la liste des fournisseurs sous contrat, la liste des magasins, l'état des ventes, ...

Ne serait-il pas intéressant d'ajouter à ces données une dimension temporelle ? N'est-il pas dommage que des données soient effacées, parce que considérées comme périmées ou plus utiles au bon fonctionnement de l'entreprise aujourd'hui ?

Il est pourtant logique d'imaginer que nous pourrions faire quelque chose de ces données. L'accumulation, l'archivage et le stockage de telles quantités d'informations pourraient aider à se faire une idée de l'évolution de la clientèle, des ventes, ... On soupçonne aisément les analyses que l'on pourrait tirer de ces données, et leurs implications en terme d'aide à la décision.

Au cours de la décennie passée, le modèle multi-dimensionnel est apparu avec l'idée de profiter de ces informations. L'idée était de rajouter une dimension temporelle aux données et d'archiver celles-ci au sein de nouvelles structures rendant facile l'ajout, l'interrogation et le classement des données.

Un grand nombre de secteurs, commerciaux ou non, ne pourraient plus aujourd'hui se passer de bases de données multi-dimensionnelles. Les grandes surfaces

ont besoin de savoir d'où provient leur clientèle, afin de déterminer leur rayon d'action et de pouvoir mettre en place un marketing approprié. Les banques veulent évaluer toujours plus efficacement les risques pour accorder un prêt. Les sociétés d'assurances calculent leurs primes en extrayant des entrepôts de données des facteurs géographiques et temporels qui seront ensuite analysés. La recherche spatiale stocke un grand nombre de données qui permettent d'écarter ou de favoriser des pistes de recherche. L'organisme Eurostat[3], permet, en ligne, de compulser toutes sortes de données à caractère démographique, socio-économique, ... Le secteur de la recherche médicale se sert grandement de données multi-dimensionnelles. Elles permettent par exemple d'identifier des régions où les cancers sont plus fréquents, et donc de dégager des pistes quant aux causes probables ; ou encore de faire des comparaisons entre différentes décennies, et donc de pouvoir émettre des hypothèses quant aux conséquences de l'évolution de nos modes de vies.

Voici un cas concret, que nous tenons d'une conversation avec un directeur financier de chez Fortis. Il y a quelques années, le groupe Fortis s'est implanté en Turquie, poursuivant sa volonté de diversifier son marché. Ils ont racheté une enseigne bancaire turque et, au bout de quelques mois de présence sur le marché, ont découvert que leurs activités concernant les cartes de crédits étaient déficitaires. Trop de gens ne parvenaient pas à rembourser leurs achats en fin de mois. Avant d'annuler tout bonnement et simplement cette activité, ils ont décidé de faire une analyse simple des chiffres. Ils ont identifié les succursales qui perdaient le plus d'argent. A l'aide des données encodées dans des entrepôts de données et d'outils de visualisation appropriés, ils ont découvert que les antennes du sud du pays "tiraient" les bons résultats du nord vers le bas. Ils ont alors arrêté de proposer ce service dans le sud, et ont renoué avec les bénéfices dans les mois suivants.

2.3.2 Présentation

Un entrepôt de données est donc un stockage de données en masse, avec différentes dimensions : spatiales, temporelles, ... On peut dire qu'il diffère du modèle bi-dimensionnel traditionnel par son historisation des données. Comme dit plus haut, une base de données n'est jamais que le reflet de la situation actuelle d'une entreprise. Typiquement, une liste des clients actuels, une liste de ventes de cette période comptable, la liste des fournisseurs en cours de contrat, ...

L'entrepôt de données, lui, permet de stocker l'ensemble de ces données, par exemple par mois, puis par année. On imagine donc aisément une relation client-ventes en deux dimensions en relationnel qui deviendrait un cube avec une historisation de ces données. dans le modèle multi-dimensionnel. Voici un exemple (figure 2.1).

[3] `http://ec.europa.eu/eurostat`

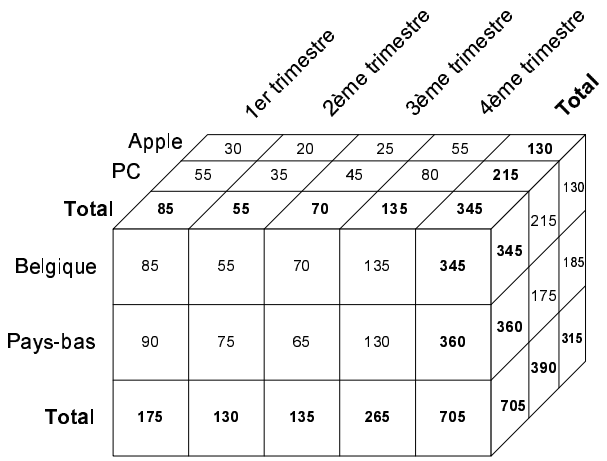

FIG. 2.1 – Représentation cubique des données

2.3.3 Définition

Bill Inmon, considéré comme étant le père fondateur des entrepôts de données et créateur de la "*Corporate Information Factory*"[4], première entreprise évoluant dans ce domaine, les définit comme suit :

> *"Un entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles, historisées et organisées pour la prise de décision."*

Dans [4], Inmon précise cette définition :

- **Orientées sujet** : La base est organisée de telle manière que les données afférent aux mêmes événements ou matières soient liées entre-elles. Ceci afin d'éviter aux données concernées par plusieurs sujets d'être dupliquées.
- **Intégrées** : Les données proviennent généralement de plusieurs départements au sein d'une organisation. Elles sont donc par essence hétérogènes, et il est nécessaire d'effectuer une gestion préalable pour uniformiser leur format et les intégrer à l'entrepôt de données.
- **Non-volatiles** : les données ne sont jamais réécrites ou effacées. Une fois enregistrées, elles sont archivées d'une manière statique, en lecture-seule et sont gardées organisées pour d'éventuelles consultations ultérieures.
- **Historisées** : Les changements sur les données sont journalisés et enregistrés de telle manière que l'on puisse consulter des historiques de changements au cours du temps.

Cette définition suppose que les données sont sauvegardées à leur niveau le plus élémentaire pour un usage basique et plus flexible rendant facile et rapide l'analyse d'informations.

Nous avons vu ici une approche des entrepôts de données se résumant au stockage et à l'organisation des données garantissant un accès à l'information passée. Il existe cependant une autre approche à la conception intrinsèque de ces

[4] `http://www.inmoncif.com/home/`

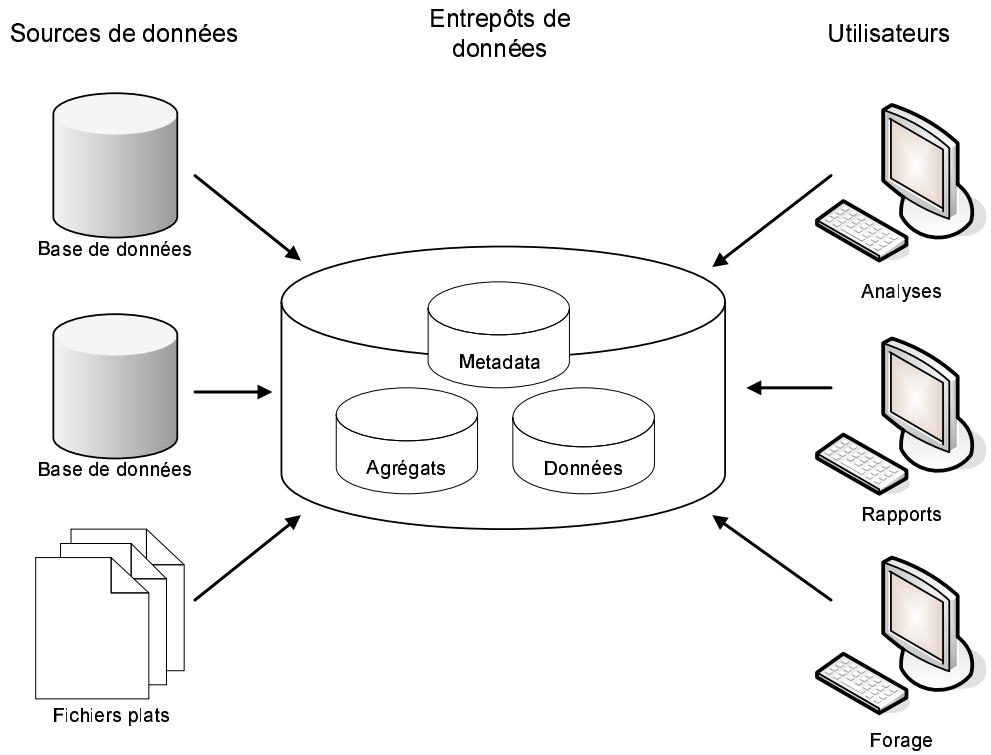

FIG. 2.2 – Architecture classique des entrepôts de données

entrepôts. Cette approche est définie dans [5], il s'agit de l'approche dimensionnelle. Dans cette approche, on privilègie les groupements facilitant les analyses futures. Par exemple, on divise des ventes en plusieurs dimensions telles que prix, nombre, lieu de ventes ... L'approche dimensionnelle est donc totalement différente de l'approche normalisée décrite par Bill Inmon.

2.3.4 Architecture

Voici un exemple (figure 2.2) classique d'architecture d'un entrepôt de données[5]. Ce schéma est largement répandu dans la communauté, on le trouve sur de nombreux sites traitant du sujet.

On peut y distinguer diverses sources de données, externes ou internes, pouvant provenir de plusieurs systèmes. Ensuite, des outils sont disponibles et permettent d'interroger ces entrepôts de données. Il peut s'agir d'outils permettant des analyses à l'aide de requêtes, d'outils permettant de faire des rapports (rapports financiers annuels par exemple) ou encore d'outils permettant de faire du forage (*Data Mining*).

Enfin, on distingue l'architecture des entrepôts de données. Ils sont composés de metadata, qui ne sont pas des données proprement dites, mais plutôt des

[5]source : `http://www.stanford.edu`

informations de gestion permettant une vision multi-dimensionnelle logique des données. Ils sont également composés d'agrégats, qui sont des calculs effectués sur les données et dont les résultats sont stockés dans la base. Ensuite viennent les données proprement dites.

2.3.5 Agrégation des données

Une agrégation est obtenue en effectuant une opération, souvent la somme, de plusieurs données détaillées [5]. Des données, et c'est encore plus vrai pour des données spatiales, présentent plusieurs niveaux de granularité possibles et jouer sur ces niveaux permet d'affiner nos résultats ou d'identifier plus précisément ou plus localement la source de problèmes. Par exemple, une chaîne possédant 30 magasins en Belgique s'inquiète de voir son chiffre d'affaire national en baisse continuelle. Elle peut alors voir les résultats province par province et identifier la province dont le chiffre chute drastiquement. Elle peut encore avoir une vision ville par ville et identifier le magasin posant un problème plus sérieux que les autres. Nous sommes donc descendus dans les niveaux de granularité et avons pu identifier la source du problème. En terme d'entrepôts de données, ces opérations s'appellent *roll-up* et *drill-down* et sont décrites dans la section suivante.

Afin d'accélérer le traitement sur des données agrégées, on propose d'effectuer un pré-calcul de ces agrégats et d'en stocker le résultat dans la base de données. Le temps de réponse sur les requêtes de ce type s'en voit fortement accéléré. Mais cette solution n'est pas sans poser des problèmes d'espace.

En effet, ces agrégats doivent également être stockés dans le système. En calculer trop serait pénalisant en terme d'espace disque et de temps de calcul nécessaire.

Il convient donc de rester prudent ! En effet, ne pas calculer les agrégats serait coûteux en temps de réponse, calculer tous les agrégats possibles serait coûteux en temps de pré-calcul et en volume du cube de données, et donc en espace disque. On estime que l'espace requis pour enregistrer ces agrégats est de 200 à 500 fois l'espace demandé par les données elles-mêmes [6]. La solution est donc, comme toujours, un compromis entre les deux solutions extrêmes. On calcule les agrégats dont on a de bonnes raisons de penser qu'ils seront souvent exploités, et on observe les requêtes fréquentes (*Frequently Asked Queries*). On peut aussi privilégier les agrégats qui sont coûteux en calculs mais pas nécessairement accédés fréquemment. Les autres agrégats sont calculés à la volée et, périodiquement, on se réserve le droit de mettre à jour le cube de données en ajoutant le résultat de nouveaux calculs, résultants de requêtes particulièrement demandées.

Il est intéressant enfin de remarquer que lors d'une modification d'un cube (hyper-cube), par exemple l'ajout d'une dimension, il faut recalculer les agrégats. Evidemment, les agrégats calculés à la volée ne sont pas concernés par ces modifications. Afin d'accélérer la mise à jour des agrégats, une méthode est proposée dans [7].

Elle consiste à établir des seuils de tolérance au-delà desquels les données ne

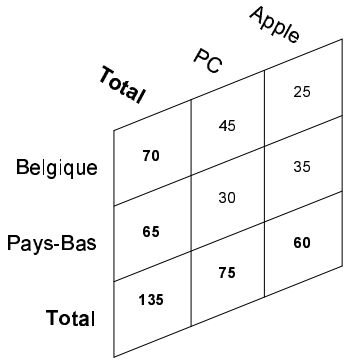

FIG. 2.3 – Opération *slice*

sont plus représentatives de la réalité. Une fois le seuil de tolérance franchi, les agrégats sont recalculés. Ces seuils de tolérance peuvent être variés, et peuvent se référer, par exemple, au nombre de changements successifs. Cette méthode est dite "*Lazy*".

2.3.6 Opérations sur les entrepôts de données

En *Data Warehousing*, on modélise les données sous forme d'un data-cube, si les données n'excèdent pas trois dimensions. Chaque dimension est représentée par une table. Reprenons notre exemple de représentation cubique (figure 2.1).

Il est à noter que ces cubes s'appellent hyper-cubes s'ils ont plus de trois dimensions. Voici les opérations courantes effectuées sur ces cubes afin d'en récupérer efficacement des informations. L'opération *slice* permet de sélectionner une dimension sur un cube. Intuitivement, il s'agit de couper une tranche du cube ; on peut alors observer les données de la dimension désirée. L'opération slice est représentée à la figure 2.4. Il s'agit d'un *slice* du troisième trimestre du cube précédemment illustré (figure 2.3).

On peut également avoir à sélectionner plusieurs trimestres à la fois, il s'agit alors de l'opération *dice*, qui extrait un "sous-cube" du cube principal. L'opération *dice* est illustrée à la figure 2.5.

Afin d'en clarifier la vision, on veut également pouvoir représenter les dimensions selon les axes que nous choisissons. Nous pourrions par exemple vouloir mettre pour le cube de notre exemple la dimension spatiale sur l'axe Z. Cette opération s'appelle le *pivot* et est illustrée à la figure 2.6. Il s'agit là d'un échange des axes "Produit" et "Localisation" consécutif à l'opération *dice*.

Deux opérations de manipulation de données sont encore à présenter. La première, *roll-up*, permet d'agréger des données selon une dimension. On peut alors remonter dans la hiérarchie d'une dimension. Par exemple, on peut vouloir agréger les données de deux trimestres et les représenter en un semestre.

Cette opération est illustrée à la figure 2.7. La seconde opération fait exactement

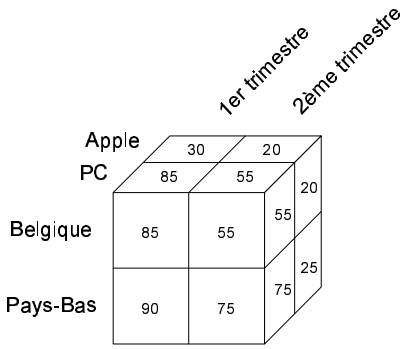

FIG. 2.4 – Opération *dice*

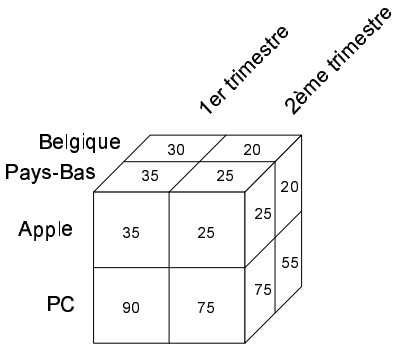

FIG. 2.5 – Opération *pivot*

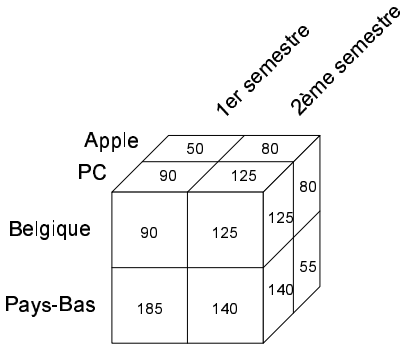

FIG. 2.6 – Opération *roll-up*

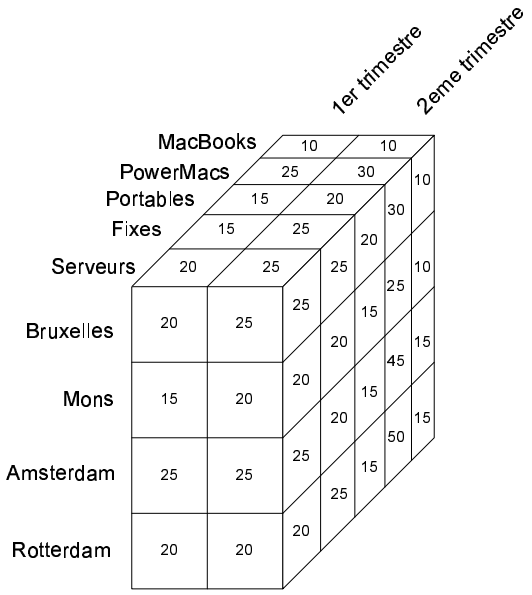

FIG. 2.7 – Opération *drill-down*

l'inverse. Elle permet de descendre dans la hiérarchie d'une dimension. Il s'agit en quelques sortes d'une spécialisation du cube, c'est l'opération *drill-down*. Un exemple est illustré à la figure 2.8. Ces illustrations sont inspirées de [8].

Afin de tirer profit des possibilités des entrepôts de données, nous devons introduire des méthodes d'organisation, d'extraction, et d'analyses de données. Les questions typiquement posées à un cube de données sont, nous l'avons vu, de type : "*Quels sont les revenus totaux du troisième trimestre pour l'ouest du pays ?*". Ces analyses de données ont permis de faire de grands pas en avant dans plusieurs domaines. Mais nous voudrions aller plus loin. Nous voudrions pouvoir poser des questions telles que "*Que se passerait-il si nous augmentions nos prix de 50 cents l'unité, tout en réduisant nos frais de transport de cinq cents ?*" Ces questions impliquent une analyse poussée des données, que l'on appelle OLAP (*On-line Analytical Processing*). L'OLTP (*Online Transaction Processing*) que nous connaissons est basé sur le modèle relationnel, là où les systèmes OLAP utilisent une vue multi-dimensionnelle des données qui procure un accès rapide à l'information stratégique permettant des analyses plus précises. Le chapitre trois du présent mémoire est consacré à une présentation de l'OLAP et à SOLAP, dont le but premier est de favoriser la visualisation des résultats provenant de requêtes OLAP.

Ceci est en effet une dimension importante dans ce domaine. Nous travaillons généralement avec une telle quantité de données, qu'il est nécessaire d'utiliser des outils de visualisation puissants. Les systèmes SOLAP sont plus particulièrement dédiés à l'affichage de données sur des cartes géographiques.

2.4 Conclusions

Dans ce chapitre, nous avons présenté l'informatique décisionnelle, et avons évoqué l'informatique décisionnelle spatiale. Nous avons vu à quel point cette discipline était devenue indispensable sur bien des plans, et qu'elle augurait à l'avenir des recherches passionnantes.

Nous avons ensuite présenté les entrepôts de données et montré qu'ils étaient indispensables aux SDSS. Nous avons montré quelques techniques pour optimiser les temps de réponse des requêtes sur les cubes de données et avons ensuite présenté les opérations courantes sur les entrepôts de données. Dans la suite, nous préciserons les outils pouvant tirer efficacement profit de ces nouvelles structures de données.

Chapitre 3

Systèmes SOLAP

3.1 Introduction

Dans ce chapitre, nous présenterons les systèmes SOLAP, définirons les outils mis à leur disposition et évoquerons les qualités essentielles dont doivent disposer ces systèmes.

Auparavant, nous évoquerons le monde des GIS (*Geographical Information Systems*) qui sont un ensemble d'outils divers et variés touchant à la représentation sur cartes géographiques de données.

Nous présenterons également les systèmes OLAP, qui permettent d'interroger efficacement des données selon une vue multi-dimensionnelle de celles-ci. Les systèmes SOLAP sont en fait une union des capacités analytiques d'OLAP et des pouvoirs de visualisation des GIS.

3.2 GIS

3.2.1 Définition

La définition suivante est largement admise dans la communauté, et tirée de [9].

> "*Un GIS, ou Geographical Information System, est un système informatique spécifique, qui permet la capture, la modélisation, la sauvegarde, la récupération, le partage, la manipulation, l'analyse et la présentation de données géospatiales.* "

Il est important de ne pas confondre GIS et SIS (*Spatial Information Systems*) ; les SIS font référence aux données spatiales au sens large, qui peuvent tout aussi bien concerner les configurations moléculaires par exemple.

Un GIS est donc constitué de plusieurs composantes essentielles [9] :

- **Une unité de traitement de données** : les logiciels GIS doivent posséder un ensemble de fonctionnalités complet permettant des analyses de haut niveau et une aide à la décision efficace au sein d'un domaine d'application. Par exemple, des applications géologiques requerront des opérations tri-dimensionnelles.
- **Une unité permettant la sauvegarde et la récupération des données** : les requêtes sur les données doivent pouvoir être effectuées rapidement, au sein de structures optimisées et de représentations claires.
- **Une unité de partage de données** : les GIS sont composés de différents logiciels (par exemple un serveur de requêtes, un navigateur internet, un logiciel de représentation de cartes, ...) Tous ces logiciels doivent se partager les données, cela implique une rigueur dans les accès, une gestion des droits, ...
- **Une unité de présentation des données** : les analyses effectuées par ces systèmes, et l'aide à la décision qu'elles apportent ne sont utiles que si on peut les visualiser sur des cartes géographiques, bi- ou tri-dimensionnelles, ou sur des graphiques particulièrement parlants.
- **Une unité spatio-temporelle** : les données sont spatiales et temporelles. Il faut permettre aux systèmes GIS de sauvegarder et traiter des données qui peuvent changer au cours du temps.
- **Un ensemble de règles** permettant de garantir l'exactitude, la précision et la fiabilité des données.

3.2.2 Motivations

L'avènement des GIS est, nous le pensons, une conséquence logique de la généralisation des entrepôts de données. Comme nous l'avons évoqué au chapitre précédent, les composantes spatiales et géographiques des données sont essentielles à l'informatique décisionnelle, et une vision claire de celles-ci permet d'en tirer la pleine quintessence. Le succès actuellement rencontré par les systèmes GIS tient au pouvoir de représentation des données qu'il offre. Les cartes géographiques sont un excellent support pour afficher de grosses quantités d'informations, tout en faisant abstraction de leur complexité. Par exemple, un simple traceur mis sur un véhicule d'une société de transports de fonds, va permettre d'afficher sa position sur une carte en temps réel. Ceci est plus intéressant qu'il n'y paraît. En effet, afficher une représentation de ce véhicule sur la carte va matérialiser, en un seul coup d'oeil, sa position, son orientation, sa proximité avec la banque, sa proximité avec d'autres camions, la présence ou non de commissariats environnants, ... Plus encore, on peut très bien concevoir le système (et c'est généralement le cas), tel qu'une petite zone de texte accompagne l'icône du véhicule, indiquant le numéro de contact, l'argent transporté, la prochaine banque à livrer, ...

Ceci illustre très bien le pouvoir de la représentation géographique. Ces systèmes se sont rendus indispensables aujourd'hui. Un autre exemple, très parlant également, est la facilité à pouvoir déterminer la couverture qu'une société occupe sur un territoire donné. Par couverture, nous entendons le nombre de succursales.

FIG. 3.1 – Couverture des succursales de la Bank of America dans la ville de New York.

Il convient pour une compagnie de choisir des sites appropriés pour l'implantation de nouveaux bâtiments sachant qu'une liste des adresses déjà occupées n'est pas révélatrice. Voici comment, à titre d'exemple, la *Bank of America* détermine sa couverture dans les villes américaines, ici Manhattan, NY (figure 3.1)[1]. Les cercles représentent les implantations. Plus le diamètre du cercle est grand, meilleure est la couverture de ce secteur.

Mais laissons de côté le domaine strictement mercantile, les GIS sont également très utilisés dans les études sur la biodiversité où ils permettent de voir clairement l'évolution de la variété des espèces au sein d'un périmètre. Et ce faisant, ils permettent d'émettre des hypothèses sur les causes d'éventuels problèmes liés à la survie d'une espèce. Dans le secteur de la recherche médicale, les entrepôts de données permettent de collecter des données sur plusieurs années et donc de tirer des conclusions quant à l'évolution de la santé publique. On imagine aisément l'aide que doit apporter la représentation de ces évolutions sur des cartes afin de déterminer la part de responsabilité des facteurs géographiques et démographiques. Le secteur du commerce équitable, en plein essor, mais à la rentabilité encore précaire, utilise des systèmes GIS dans les vérifications de viabilité des choix de producteurs. Mieux encore, car il s'agit de recherches très en vogue, on utilise des GIS en climatologie, afin d'aider à déterminer d'où proviennent les boulversements climatiques et leurs possibles conséquences futures. Nous pourrions encore citer les apports des GIS en génie civil, urbanisme, arrêtons ici une liste que nous aurions bien du mal à énoncer de manière exhaustive.

[1] source : `http://www.gis.com`

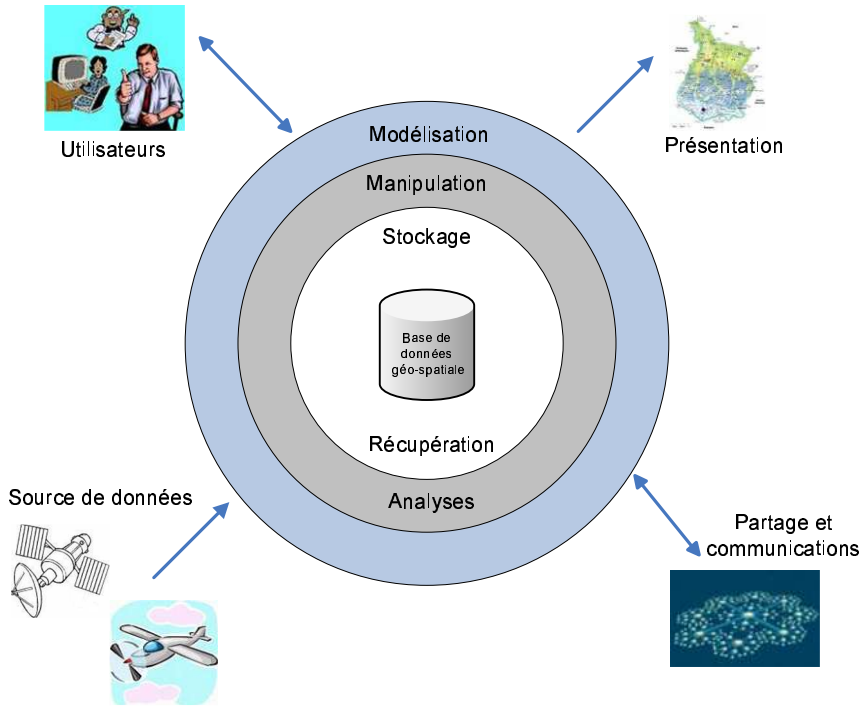

FIG. 3.2 – Architecture d'un système GIS.

3.2.3 Architecture

Alors que les logiciels GIS sont très spécialisés, le hardware nécessaire est lui essentiellement le même que dans les systèmes d'informations traditionnels. Il est entendu cependant que ces logiciels utilisent bien souvent des téraoctets de données et qu'il convient dès lors de fournir des solutions de stockage appropriées. Autre outil peu commun mais très utilisé dans ce domaine, le digitaliseur (*digitizer*). Il s'agit d'une grande plaque tactile sur laquelle on peut, à l'aide d'un stylet, dessiner et reproduire des formes. Cet outil est particulièrement utile quand il s'agit de numériser des cartes.

Voici un schéma (figure 3.2) reprenant les différentes interactions possibles avec un système GIS, tiré de [9].

3.2.4 Modélisation des données

Lorsque l'on veut représenter des données géographiques, deux conceptions s'affrontent : la conception en *objet* (*object-based model*), et la conception en *champ* (*field-based model*).

Conception en objet La conception en objet considère l'espace comme étant peuplé d'entités discrètes et identifiables, chacune ayant une référence spatiale. Une entité doit :

– être identifiable,
– être utile,
– possèder des caractéristiques permettant de la décrire.
Ces entités sont représentées à l'aide de vecteurs, de points, ou encore d'arcs et de cercles qu'il convient de discrétiser. C'est la représentation dite "vecteur" (*vector*).

Conception en champ La conception en champ traite les données géographiques comme un schéma, que l'on peut diviser et assigner, sans se soucier des objets présents sur la carte. Par exemple, une carte que l'on représente à l'aide d'un quadrillage. C'est ce que l'on appelle la représentation en trame (*raster*). Les cartes sont formées à partir d'une matrice dont chaque cellule contient une information ; la taille de la cellule équivaut à une distance sur le sol. L'information comprise dans les cellules est généralement une couleur, délimitant le terrain. Afin d'afficher toutes sortes d'informations complémentaires, autres que les délimitations des frontières, les cartes forment une architecture en couches (*layers*). Une couche peut représenter les routes, une autre les cours d'eau, une autre délimite les parcelles, ...

3.2.5 Outils GIS

Parmi les produits phares du marché, on retrouve la société ESRI et sa suite logicielle ArcGIS[2], la société MapInfo[3], Microsoft MapPoint[4] et plusieurs solutions open-sources, dont les plus répandues sont : MapServer[5], GrassGIS[6] et citons également GMLViewer[7], qui présente l'avantage de s'afficher via un plugin dans un navigateur internet et dont nous nous sommes servis dans le cadre de ce mémoire. Nous aurons l'occasion de la présenter plus en détails dans un chapitre ultérieur.

3.3 OLAP

3.3.1 Génèse des systèmes OLAP

Avec l'avènement des entrepôts de données et l'enthousiasme grandissant de l'industrie pour l'informatique décisionnelle, le besoin d'outils analytiques puissants s'est fait sentir. On parlait alors d'analyses de données multi-dimensionnelles mais ce terme était quelque peu réducteur et on lui préféra le terme d'*Online Analytical Processing* (OLAP). En effet, l'analyse multi-dimensionnelle n'est qu'une de ses caractéristiques. C'est Edgar F. Codd qui a introduit le nom

[2] `http://www.esri.com/software/arcgis/index.html`
[3] `http://www.mapinfo.com/mipro`
[4] `http://www.microsoft.com/mappoint/default.mspx`
[5] `http://mapserver.gis.umn.edu/`
[6] `http://en.wikipedia.org/wiki/GRASS_GIS`
[7] `http://demo.communitymapbuilder.org/gml/gmlviewer-1_0/`

OLAP dans [10]. Il y définissait 12 règles pour qu'un produit soit estampillé OLAP. Mais ces règles furent jugées trop nombreuses et peu crédibles. En effet, Edgar Codd travaillait à l'époque pour la société Arbor Software (devenue Hyperion), et était donc tenté d'énoncer des critères favorisant les produits proposés par la société qui l'employait. En 1994 se créa le "*OLAP Report*" qui regroupait des chercheurs totalement indépendants de toute entreprise commerciale. Ils se sont penchés sur une définition plus courte et plus générique de ce que devait être un système OLAP. Leur définition tient en cinq mots-clés [8] :

Fast Analysis of Shared Multidimensional Information – FASMI

Cette définition est largement admise dans la communauté OLAP et est citée sur 120 sites web dans 30 pays[8]. Précisons cette définition. Les applications OLAP doivent correspondre à cinq caractéristiques :

- **Fast** — Rapide : signifie que le système est pensé pour délivrer les réponses dans les cinq secondes. Les analyses très simples sont délivrées en moins d'une seconde et le temps de réponse des requêtes complexes doit rarement dépasser les vingt secondes. Les produits actuellement sur le marché tentent de parvenir à ce résultat grâce à des hardwares spécifiques et des pré-calculs. Notons que sur de gros volumes de données, les résultats ne sont pas encore satisfaisants.
- **Analysis** — Analyse : le système doit offrir des outils analytiques à l'utilisateur, et ces outils doivent pouvoir s'adapter aux besoins du domaine d'activité considéré. De plus, ils doivent être ergonomiques, facilement paramétrisables et intuitifs. Ces systèmes doivent enfin proposer une facilité et une flexibilité de calcul, sans avoir recours à des programmes supplémentaires.
- **Shared** — Partagée : le système doit être partagé et doit donc mettre en place un mécanisme de protection lors d'accès multiples en écriture par exemple. En effet, de plus en plus de solutions logicielles autorisent la réécriture des données et nombre d'entre-elles ne proposent pas de protections satisfaisantes. Il est nécessaire que ces mécanismes sécuritaires soient paramétisables, afin de garantir une sécurité adaptable aux besoins de chacun.
- **Multidimensionality** — Aspect multi-dimensionnel : l'élément clé des solutions OLAP. Le système doit fournir une vue conceptuelle, multi-dimensionnelle des données et supporter de multiples niveaux de hiérarchisation. L'*OLAP Report* ne précise pas quels outils doivent être utilisés, mais ces derniers doivent garantir une conception multi-dimensionnelle des données.
- **Information** : représente toutes les informations disponibles qui doivent être accessibles pour les analyses. On peut mesurer l'efficacité d'un système de par la quantité d'informations qu'il peut gérer, avec quelle quantité de RAM, l'espace disque dont il a besoin, ... Ceci est un aspect fondamental et crucial. Nous avons énoncé plus haut que parmi ces systèmes, nombre d'entre eux sont confrontés à de très larges volumes de données. Il est donc nécessaire de prendre cet aspect en considération lors de la conception de tels systèmes.

[8] `http://www.olapreport.com/fasmi.htm`

Les informations disponibles dans la base de données devront être accessibles facilement, et le temps de réponse des requêtes doit être réaliste en terme de coût, qu'il s'agisse de coût en matériel ou en temps.

3.3.2 Types d'OLAP

Vu les nombreux domaines d'application des technologies OLAP, plusieurs types existent. Ils sont définis en catégories. Celles-ci se rapportent aux domaines d'application, aux techniques utilisées pour le stockage des informations et à l'utilisation que l'on pourra en faire.

MOLAP

La première de ces catégories est MOLAP, (*Multidimensional On-Line Analytical Processing*). C'est la manière la plus traditionnelle de faire de l'OLAP. Les données sont stockées dans un cube multi-dimensionnel, dans un format propriétaire. Cet emploi de cube est très productif en terme de gain de performances. En effet, ils sont conçus et optimisés pour retrouver rapidement l'information demandée. Intuitivement et comme nous l'avons montré, les opérations *slice*, *dice* sont particulièrement aisées sur un cube de données. De plus, les cubes supportent des calculs complexes et rapides, via les techniques de pré-agrégats que nous avons évoqués au chapitre précédent.

Il convient cependant de nuancer cette affirmation. En effet, quand les données sont volumineuses, le calcul d'agrégats était très coûteux, et donc le temps de réponse des requêtes sur des données non-agrégées peuvent être longues. De plus, ces agrégats sont ajoutés aux données déjà présentes, augmentant considérablement l'espace disque nécessaire.

Enfin, les systèmes permettant de stocker des données sous forme multi-dimensionnelle sont propriétaires. Cela pose plusieurs problèmes :

- l'investissement initial peut être très lourd ...
- ... dissuadant le passage chez les concurrents,
- les technologies propriétaires requièrent du personnel qualifié, qu'il faut former à des coûts bien souvent prohibitifs,
- les besoins d'une société ne sont pas toujours en adéquation avec la suite d'outils proposée par le vendeur.

ROLAP

Les outils ROLAP (*Relational On-Line Analytical Processing*) proposent, en opposition aux technologies MOLAP, de stocker les informations dans des bases de données relationnelles classiques, et fournissent des outils pour les interroger selon une vue multi-dimensionnelle. Ceci comporte plusieurs avantages.

La technologie des bases de données relationnelles est éprouvée et possède de nombreux outils avec lesquels beaucoup de systèmes sont familiers. L'exemple

le plus flagrant est le langage SQL (*Structured Query Language*), qui peut être utilisé pour interroger ces données. Les différentes opérations réalisables sur les cubes peuvent être effectuées en SQL à l'aide de la clause `WHERE`. Les informaticiens capables d'utiliser ces systèmes sont nombreux, ce langage étant un standard de facto. Il n'est donc pas nécessaire de former des spécialistes aux systèmes propriétaires.

Un autre avantage indéniable est la taille de ces bases de données. Les systèmes ROLAP en tant que tels n'imposent pas de limitation quant aux volumes de données autorisés. La seule limitation provient de la taille maximale de la base de données sous-tendante. Les systèmes relationnels sont beaucoup plus performants pour de gros volumes que les systèmes nativement multi-dimensionnels.

Ensuite, les bases de données relationnelles proposent à l'origine des mécanismes sécuritaires plus poussés et on peut imaginer poser un filtre sur les requêtes, réduisant l'accès à certaines données à un groupe d'utilisateurs particulier.

Si cette catégorie a des qualités, elle a aussi des défauts. Et notamment des performances réduites par rapport au MOLAP. Mais cette discussion est sujette à polémique. En effet, les systèmes nativement multi-dimensionnels sont plus rapides, mais nous avons vu que cela dépendait de la quantité de données. La technologie ROLAP offre l'immense avantage de ne pas limiter la quantité de données admissible et s'avère donc indispensable lors du choix de solutions répondant à des environnements particulièrement gourmands en volume de données.

Enfin, les systèmes ROLAP sont freinés dans leurs possibilités par la technologie employée. Le langage SQL n'offre en effet pas toutes les possibilités dont disposent les requêtes sur les modèles multi-dimensionnels natifs. Cependant, de nouvelles fonctions sont apparues dans le langage SQL, notamment les fonctions *ROLLUP* et *CUBE* décrites dans le chapitre précédent. Ces fonctions ont été introduites dans *SQL server* depuis la version 6.5 et dans Oracle depuis la version 8i[9].

HOLAP

La catégorie d'outils HOLAP (*Hybrid On-Line Analytical Processing*) est, comme son nom l'indique, un hybride entre les deux systèmes cités plus haut. Ces systèmes ont tenté de garder les avantages des deux techniques dont ils sont issus. Concrètement, les données sont stockées dans une base de données relationnelle classique et les agrégats sont stockés dans un modèle multi-dimensionnel propriétaire. Ceci peut se faire selon plusieurs critères choisis par l'utilisateur final.

Le problème de cette famille est que peu d'outils sur le marché supportent à la fois les deux technologies, et les systèmes HOLAP sont donc généralement propriétaires. Ce qui amène plusieurs inconvénients dont nous avons parlé à propos des outils MOLAP.

[9] `http://databases.about.com/od/sql/l/aacuberollup.htm`

Il existe d'autres catégories telles que WOLAP (*Web-based OLAP*), RTOLAP (*Real-Time OLAP*) et bien d'autres encore mais il ne nous semble pas utile de les détailler dans le cadre de ce mémoire.

Ajoutons évidemment le SOLAP, que nous présentons dans la section suivante du présent chapitre.

3.3.3 Outils OLAP

Différentes solutions logicielles se partagent l'affiche dans ce marché en pleine expansion. La société Microsoft est actuellement leader[10] avec 31% de parts de marché, suivi par Hyperion Solutions[11] avec 18%. Parmi les grandes sociétés du secteur, on peut aussi citer SAP[12] avec 5,8 % et Oracle[13] crédité quant à lui de 2,8%[14]. Ceci s'explique par le fait qu'Oracle propose des solutions OLAP depuis sa version *9i*, sortie en 2001 et a mis du temps avant de proposer des solutions efficaces. Par exemple, Oracle ne fournit toujours pas de solution SOLAP telle quelle. Ce mémoire proposera des pistes pour combler cette lacune.

3.4 SOLAP

3.4.1 Présentation

Il est reconnu que 80% des données présentes dans les différentes bases de données mondiales ont un caractère spatial [11], mais il est sauvegardé sous forme textuelle dans les tables. Afin de tirer profit efficacement de cette manne extraordinaire d'informations, il a fallu créer de nouveaux outils dédiés. C'est le cas des GIS et, avec eux, certains outils d'analyse dédiés à la composante spatiale et à la combinaison de capacités analytiques et d'outils de représentation : les SOLAP (*Spatial Online Analytical Processing*).

> *"Il a été démontré que OLAP possède un certain potentiel pour supporter l'analyse spatio-temporelle. Cependant, sans un volet spatial permettant de visualiser et de manipuler la composante géométrique des données spatiales, l'analyse peut demeurer incomplète. Une nouvelle catégorie d'outils OLAP, les outils SOLAP, est introduite." [12]*

La réflexion sur ce besoin de nouveaux outils est très récente. A notre connaissance, le premier article présentant les concepts sous-tendants à SOLAP date de 2001 et était intitulé "*Toward better support for spatial decision making : Defining the characteristics of spatial On-line Analytical Processing (SOLAP).*"

[10] `http://www.microsoft.com/sql/default.mspx`
[11] `http://www.hyperion.com/`
[12] `http://www.sapdb.org`
[13] `http://www.oracle.com/technology/products/bi/olap/index.html`
[14] source : `http://www.olapreport.com/market.htm` — chiffres 2006

Depuis, et malgré le laps de temps très court, des solutions logicielles sont apparues au Canada, aux Etats-unis, en France, au Portugal et dans d'autres pays et de nombreux chapitres de livres ont été consacrés à ce sujet [13].

3.4.2 Définition

Le Dr. Yvan Bédard, professeur en GIS et en base de données géospatiales au centre de recherche en géomatique de l'Université de Laval au Québec, la référence en matière de SOLAP, les définit comme suit :

> "*A visual platform built especially to support rapid and easy spatio-temporal analysis and exploration of data following a multidimensional approach comprised of aggregation levels available in cartographic displays as well as in tabular and diagram displays.*"
>
> "Une plateforme visuelle conçue spécialement pour supporter des analyses spatio-temporelles rapides et faciles, et l'exploration de données selon une approche multi-dimensionnelle venant de niveaux d'agrégation disponibles autant dans les représentations cartographiques que dans les représentations en tableaux et diagrammes."

Il est à noter qu'entrent également dans la catégorie des SOLAP les outils qui affichent des données non spatiales sur des cartes géographiques. En effet, la dimension spatiale est bien présente, même si elle n'est pas reprise sous forme de coordonnées dans une base de données spatiale. Elle peut l'être sous forme textuelle représentant des pays, régions, adresses, ...

3.4.3 Vision

Les outils OLAP ne permettaient pas de visualiser sur des cartes les résultats des requêtes, ce qui rendait les analyses sur les dimensions spatiales quasi illisibles et peu révélatrices. Il est effectivement impossible de retirer des faits, des analyses, à partir de colonnes de chiffres, ou même de dizaines de graphiques. Dans beaucoup de cas, il s'avère nécessaire pour une bonne compréhension des faits, de pouvoir visualiser les données sur des cartes. Les tendances se dégageront de manière plus évidente que si l'on avait comparé beaucoup de nombres et de graphiques. Les cartes ont ce pouvoir d'abstraire toutes les données relatives à l'espace. Cela permet de ne plus se concentrer que sur les données non-spatiales, le support sur lequel elles sont affichées indiquant clairement d'où elles proviennent.

SOLAP doit répondre à ces besoins, et par ce biais aider à la prise de décision. La conception des systèmes SOLAP doit donc correspondre à un certain nombre de critères, énoncés ici.

Visualisation des données. Il s'agit là d'un principe de base des systèmes SOLAP. La visualisation des données doit être claire, flexible, et doit per-

mettre d'utiliser des outils autres que la représentation cartographique. Des tableaux, des outils statistiques, des graphiques sont autant de compléments d'informations souvent indispensables à la bonne compréhension des données. Il est donc important de pouvoir afficher plusieurs représentations simultanément. Par exemple, il serait intéressant de représenter sur une carte l'évolution de la couverture régionale d'une enseigne, et sur un graphique, l'évolution éventuelle du chiffre d'affaires de la société — le jeu en valait-il la chandelle ?

Personnalisation de l'interface. L'outil doit permettre une personnalisation poussée. L'utilisateur doit pouvoir choisir les couleurs représentant les divers composants (afin, par exemple, d'associer une couleur à un type de bâtiments au cours d'études urbanistiques), les éléments qu'il désire voir s'afficher, ... Les logiciels SOLAP étant, nous l'avons démontré, appelés à être employés dans un ensemble de domaines variés ; cette adaptation aux besoins de l'utilisateur est donc capitale.

Manipulation des données. Nous voulons pouvoir exploiter pleinement les possibilités de la représentation graphique des données à dimension spatiale. Il serait intéressant de pouvoir visualiser facilement les opérations *roll-up* et *drill-down*. Par exemple, passer d'une visualisation province par province à une visualisation ville par ville peut se faire très intuitivement en zoomant sur la carte, tout en conservant les mêmes critères de recherche. Nous pouvons ajouter à ces observations la nécessité d'une vision claire de l'évolution au cours du temps, vision qui devrait être disponible sous plusieurs formes afin d'en affiner la représentation. Un exemple évident à ce sujet nous vient des études démographiques. L'enjeu démographique est une préoccupation pour beaucoup de nations actuellement, et il serait utile de visualiser sur une carte l'évolution dans ce domaine au cours du temps. Pour plus de précision encore, on pourrait travailler de concert avec des graphiques sur dix, vingt ou cinquante ans. Les prévisions gagneraient en réalisme. Il est à noter que l'opération permettant de visualiser les données au cours de différentes périodes est parfois nommée dans la littérature *drill-accross*, ou forage horizontal.

3.5 Conclusions

Dans ce chapitre, nous avons introduit les systèmes SOLAP et énoncé le besoin grandissant pour cette catégorie de produits. Les systèmes SOLAP sont très variés, et peuvent aller de la représentation sur cartes géographiques, au logiciel avec carte interactive interrogeant directement la base de données, et ce qu'elle soit spatiale ou non.

Dans ce mémoire, nous allons réaliser la première approche, à savoir l'affichage de requêtes OLAP sur une carte géographique. Pour ce faire, nous allons notamment utiliser un langage récent, spécialement conçu pour véhiculer des informations à caractère géographique, le GML (*Geography Markup Language*).

Ce langage, et le méta-langage à partir duquel il s'est construit, le XML, font l'objet des deux chapitres suivants.

Chapitre 4

Le méta-langage XML

4.1 Introduction

Ce chapitre traitera du méta-langage XML. Ce langage est beaucoup utilisé dans ce mémoire et il nous paraît utile de le détailler. De plus, le langage GML, dont nous nous sommes également servi, est issu du XML. Il apparaît donc nécessaire, pour parfaire la présentation du GML, de présenter le langage XML, ainsi que les concepts sous-tendants à sa création. C'est le sujet du présent chapitre. Nous présenterons le langage, ses forces et ses faiblesses, avant de laisser place, au chapitre suivant, à la présentation du langage GML.

4.2 Historique

Vers la fin des années soixante, le besoin de représenter les données électroniquement se faisait sentir. Charles Goldfarb, alors employé chez IBM, au *Cambridge Scientific Center* fut chargé de se pencher sur le sujet. Il travaillait en collaboration avec Ed Mosher et Ray Lorie, notamment.

Le principal problème qu'ils rencontrèrent était que les programmes sur lesquels ils travaillaient nécessitaient chacun un système de balises différent. C'est ainsi que naquit le *Generalized Markup Language* (GML) en 1969[1]. Initialement, ils travaillaient sur la mise en forme de documents légaux, mais on s'aperçut vite que ce langage pouvait être généralisé au traitement de tous types de textes.

Charles Goldfarb continua le développement de GML et ajouta notamment un analyseur qui permettrait de valider un document GML. Ce fut là l'évolution principale vers le langage SGML. En fait de langage, il s'agit plutôt d'un méta-langage, permettant de définir des langages. Le premier *working draft* sur SGML fut publié en 1980 par l'ANSI (*American National Standards Institute*)[2] qui le

[1] L'appellation GML est ici sujette à confusion, et il convient de ne pas confondre le *Generalized Markup Language* et le *Geography Markup Language* auquel ce chapitre est consacré, le premier étant l'ancêtre du second.

[2] source : `http://www.sgmlsource.com/history/index.htm`

propulsa au rang de standard.

L'idée fondamentale de SGML est de séparer la structure logique d'un document (titre, paragraphes, listes, ...) de sa présentation, qui doit être confiée à un fichier externe décrivant couleurs, polices, grandeur de caractères, ... De part sa vocation générique, SGML fut utilisé dans des domaines très variés tels que les bibliothèques, l'ISO publishing department, les éditeurs de textes (Microsoft Word, Wordperfect) et tant d'autres encore[3].

En 1990, HTML fut créé dans le sillage du World Wide Web par Tim Berners-Lee. Il employa des balises alors utilisées au CERN où il travaillait. Le principal problème de l'HTML est qu'il s'écartait d'un principe fondateur de SGML : la séparation de la structure du document et de sa mise en forme. En effet, HTML permet, au sein du document, des balises telles que `<font>`, `<b>` qui rompent avec les spécifications du format SGML.

De plus, HTML n'est pas flexible et la traduction de documents complexes peut s'avérer difficile voire impossible en HTML strict. Ce langage est dédié à l'affichage de pages web, et ne permet en aucun cas la description de documents divers.

Enfin, le standard HTML n'est pas respecté par tous les éditeurs de navigateurs web. Des balises sont rajoutées telles que la balise `<marquee>` qui fut proposée au w3c (*World Wide Web Consortium*)[4] qui la refusa.

Cependant, HTML, langage très facile à utiliser, eut le mérite de faciliter la diffusion et la propagation (hyperliens) de contenus sur internet.

Le langage XML a été développé par un groupe de travail, le *SGML Editorial Review Board*, en 1996. La motivation initiale était de combiner la flexibilité de SGML et la simplicité d'HTML. Depuis, XML se généralise dans des domaines très divers et son utilisation ne cesse de croître. James Clark, un développeur open-source évoluant dans la sphère XML prétend que ce succès est dû **"au triomphe de la simplicité"**.[5]

4.3 Principes de base

XML est l'abréviation de *eXtensible Markup Language*. Il a été conçu selon les dix principes suivants [14] :

- XML devra être utilisable facilement sur Internet,
- XML devra supporter une grande variété d'applications,
- XML devra être compatible avec SGML,
- il devra être facile d'écrire des programmes gérant des documents XML,
- le nombre d'options disponibles dans XML devra être réduit au minimum, idéalement à 0,

[3] source : `http://wwwasdoc.web.cern.ch/wwwasdoc/WWW/publications/sgmlen/node3.html`

[4] `http://www.w3c.org`

[5] source : `http://www.ddj.com/184404686`

– les documents XML devront être lisibles par l'homme et raisonnablement clairs,
– le design d'XML devra être fait rapidement,
– le design d'XML devra être formel et concis,
– les documents XML devront être faciles à créer,
– la concision dans les balises est d'une importance minime.

4.4 Motivations

XML est né du manque de solutions faciles à utiliser, génériques, flexibles en matière de description de documents. Il permet de stocker d'importants volumes de données sous forme textuelle. Ceci présente l'avantage d'être lisible par les hommes comme par les machines.

Du point de vue de l'utilisateur, un fichier XML est parfaitement lisible car les noms des balises sont souvent éloquents et permettent leur compréhension. De plus, la structure même du fichier permet de comprendre la manière dont les données sont imbriquées.

Du point de vue de la machine, de tels fichiers texte laissent libre choix aux programmes quant à leur utilisation. Partons d'un fichier XML relatant un agenda. Ce même fichier pourra être partagé entre deux programmes différents de gestion de calendrier. L'affichage des informations ne sera pas identique, mais pour peu que les programmes utilisent les mêmes formalismes XML, nous avons là un bon moyen d'échange d'informations.

Les documents XML étant par définition extensibles, ils peuvent s'adapter à tous types d'applications et ce, dans n'importe quel domaine. Une application qui nous intéresse particulièrement est le RSS (*Really Simple Syndication*) que l'on peut traduire par "souscription vraiment simple". Les flux RSS tiennent une grande part de responsabilité dans le succès de ce qu'il convient d'appeler le web 2.0 et ils se trouvent être codés en XML. Le programme de traitement de texte Microsoft Word utilise pour ses fichiers une représentation en XML, de manière native, et ce depuis 2003. Les exemples à ce sujet ne manquent pas !

Un autre aspect intéressant d'XML est son universalité. La norme XML impose en effet que les sets de caractères UTF-8 et UTF-16 soient supportés. Il s'agit là de la norme unicode qui vise à regrouper tous les caractères de toutes les formes de langages en une seule et même norme[6].

4.5 Le document XML

Comme nous l'avons expliqué dans la section précédente, le XML est un langage de balises, comme l'HTML. A ceci près que les balises ne sont pas pré-définies. Il convient au programmeur de définir ses balises, leur hiérarchie et ce qu'elles

[6] `http://www.w3.org/TR/2006/REC-xml-20060816/#charsets`

peuvent contenir, mais nous y reviendrons. Voici un exemple de code XML, décrivant un classique annuaire :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Description de mon annuaire -->
<annuaire>
      <contact id="a1">
           <non>Dion</nom>
           <prenom>Caroline</prenom>
           <adresse>
                <pays>Canada</pays>
                <province>Quebec</province>
                <ville>Montreal</ville>
                <zip>H1W3Y5</zip>
                <rue>La Fontaine Ouest</rue>
                <num>1550</num>
           </adresse>
           <tel>514-738-2167</tel>
      </contact>
      <contact>
           ...
      </contact>
</annuaire>
```

Analysons ce document XML basique. Il est constitué de plusieurs éléments distincts :

- La première ligne constitue le **prologue**, permettant d'identifier la version utilisée du langage, le jeu de caractères d'encodage et éventuellement de spécifier si le fichier est *standalone* c'est-à-dire dépendant d'autres fichiers ou pas.
- La deuxième ligne constitue un **commentaire**.
- La troisième ligne débute le document XML proprement dit. Il s'agit d'un ensemble de balises appelées **éléments**, qui constitue le squelette du document. La première balise, `<annuaire>` est appelée **racine**. C'est la balise qui contiendra toutes les autres.
- La balise `<contact id="a1">` diffère légèrement des autres en ce qu'elle contient un **attribut**. Il s'agit en l'occurrence de l'identifiant du contact.
- Entre les balises figure ce que l'on appelle le **texte**.

Un élément d'un fichier XML est donc constitué d'un nom, d'un éventuel attribut, et d'une liste de fils, qui sont également des éléments. Intuitivement, on peut donc modéliser cette structure sous la forme d'un arbre ; ce qui est d'ailleurs l'objet d'une recommandation très précise du W3C, le DOM (*Document Object Model*).

Le DOM est une représentation informatique d'un document XML. Il permet une standardisation pour la lecture et la manipulation des documents par le

biais d'un programme. Ceci inclut : la création de documents, la navigation au sein de sa structure, l'ajout, la modification et la suppression des éléments.

Le W3C, dans sa recommandation[7], définit le DOM comme suit (traduit de l'anglais) :

> "*Une interface indépendante du langage et de la plate-forme qui permet aux programmes et aux scripts d'accéder et de mettre à jour de manière dynamique le contenu, la structure et le style des documents. Le DOM procure un ensemble d'objets standardisés pour représenter des documents XML et HTML [...].*"

Nous aurons l'occasion d'y revenir plus tard dans ce chapitre.

4.6 Validation d'un document XML

Afin de garantir le fonctionnement d'un document XML, il est nécessaire de le valider, c'est-à-dire de s'assurer qu'il se conforme bien à des règles pré-définies (contraintes, conditions, ...).

A ce sujet, il convient de distinguer deux notions importantes :

- Un document XML est **valide** s'il satisfait aux contraintes imposées (par exemple : le numéro de rue ne peut contenir de lettres, un élément doit avoir au moins trois fils *fils1*, *fils2*, *fils3*).
- Un document XML est **bien formé** s'il correspond à la spécification du langage. Plus particulièrement :
  - Le fichier XML doit être un document, au sens de la spécification du W3C. Soit il contient un élément, la racine, ou plusieurs éléments, embrigadés dans la racine, définis à l'aide d'une balise ouvrante et d'une balise fermante du même nom.
  - Il doit rencontrer toutes les contraintes définies dans la spécification.
  - Chaque entité analysée doit être bien formée au sens de la spécification.

Pour être valide, un document XML doit tout d'abord être bien formé, et donc correspondre à la spécification. Le processus de validation d'un document est sensiblement plus compliqué que son caractère bien formé.

Afin de préciser ce qu'un document peut ou doit contenir, on a besoin d'exprimer des conditions. L'analyseur XML pourra ensuite valider le document s'il rencontre la grammaire préalablement définie. Il existe plusieurs langages permettant de définir ces conditions, ils sont généralement appellés *Schemas Languages*, ou langages de schémas. Ces différents langages ont leurs forces et leurs faiblesses. Nous allons ici nous concentrer sur les deux langages les plus couramment utilisés, généralement compris par les outils de validation XML : les DTD (*Document Type Definition*) et le XML Schema.

[7] `http://www.w3.org/TR/REC-DOM-Level-1/`

4.6.1 Le DTD

Le DTD se concentre sur la structure même du document ; il en définit la syntaxe. Le DTD est natif au XML, inclut dans la spécification du W3C. La définition d'un document XML peut se faire via un DTD en interne (les définitions sont incluses dans le fichier XML proprement dit) ou en externe (les définitions sont dans un fichier séparé, accessible depuis le document XML via une URL spécifiée).

Le DTD est simple d'application, il permet de vérifier rapidement la validité d'un document. Cependant, il a montré plusieurs faiblesses dont certaines se sont révélées problématiques.

Premièrement, il ne supporte pas les nouvelles fonctionnalités du XML, notamment les *namespaces*. Les namespaces sont une extension de la norme 1.0 qui permettent de résoudre des conflits d'homonymie de balises dans les documents XML. En effet, pour souci de modularité, si un vocabulaire de balises (*Markup Vocabulary*) est déjà créé pour un domaine particulier, avec des programmes déjà conçus permettant leur interprétation, on veut pouvoir le réutiliser, c'est-à-dire utiliser les mêmes formalismes au niveau des balises. Il pourrait alors se produire des collisions si, lors de l'inclusion de ces fichiers XML, d'autres balises présentes antérieurement présentaient les mêmes noms, mais décrivant des objets différents. Pour utiliser des namespaces déjà existants, il suffit de préciser au document XML une URL via le mot clé `xmlns`. On peut alors réutiliser ces balises et les distinguer des balises présentes dans notre propre fichier XML. Lors de l'écriture d'une balise, on précise si elle provient du namespace ou pas.

Deuxièmement, on ne peut pas tout représenter dans le format DTD. Certains formalismes XML ne sont tout simplement pas représentables et le typage des données est très généraliste.

Enfin, le DTD n'a pas une syntaxe XML, il ne s'agit pas d'un document XML bien-formé ; sa syntaxe ne correspond pas à la norme XML.

Voici, à titre d'exemple, une portion de code DTD qui valide le code XML de notre exemple :

```
<!ELEMENT annuaire (contact+)>
<!ELEMENT contact (nom, prenom, adresse, tel)>
<!ATTLIST contact id ID #REQUIRED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT adresse (pays, province, ville, zip, rue, num)>
<!ELEMENT pays (#PCDATA)>
<!ELEMENT province (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
[...]
<!ELEMENT tel (#PCDATA)>
```

Quelques précisions sur ce document DTD :

- Pour chaque élément (ou balise), défini par `!ELEMENT`, on précise son nom, et ses éventuels fils, énoncés entre paranthèses. `contact+` signifie que l'élément peut avoir un ou plusieurs fils de ce type.
- Les attributs sont définis à l'aide du mot-clé `!ATTLIST` qui précise le nom de l'attribut et la valeur que l'on attend. En l'occurrence, il existe un mot réservé `ID` dans DTD qui permet de préciser que la valeur attendue doit être unique. Ceci permet d'affiner le fonctionnement de l'analyseur. Si nous avions entré un autre contact avec le même ID, le document XML n'aurait pas pu être validé.
- Pour chaque élément, nous devons préciser le type d'entrée dont il va s'agir. `PCDATA` signifie que l'on attend une chaîne de caractères. Nous ne détaillerons pas ici les différents types possibles.

Ceci démontre clairement une des plus grandes faiblesses de DTD ; son manque de souplesse. En effet, on ne peut préciser quel type de chaîne de caractères est attendue à un endroit spécifique. Il pourrait être intéressant par exemple de préciser quel format doit avoir la chaîne de caractères compris entre les balises `<zip>`. S'agissant, par exemple, d'un fichier exclusivement canadien, les codes postaux doivent être de la forme LCLCLC.

Dans la section suivante, nous décrivons le langage XML Schema, qui pallie notamment à ce problème.

4.6.2 Le langage XML Schema

Le XML Schema a été décrit pour la première fois dans la recommandation du W3C en Mai 2001. Il est grandement inspiré de ses prédécesseurs, tels que SOX, XML-DATA et XDR. Il s'agit en fait d'un compromis des différents langages de schémas précédents.

Il introduit notamment le typage des données. Les booléens, entiers, ... sont représentables avec le XML Schema. Les `namespaces` sont également reconnus et la syntaxe XML Schema est une syntaxe XML bien formée.

Autre apport intéressant, XML Schema permet l'héritage. Des éléments peuvent être définis à partir d'autres éléments, héritant de leur contenu et éventuels attributs.

Voici quelques morceaux choisis du code XML Schema validant notre exemple XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="nom" type="xsd:string"/>
[...]
<xsd:element name="num" type="xsd:int"/>
[...]
```

```
<xsd:element name="zip">
<xsd:simpleType>
     <xsd:restriction base="xsd:string">
          <xsd:pattern value=red"[A-Z][0-9][A-Z][0-9][A-Z][0-9]"/>
     </xsd:restriction>
</xsd:simpleType>
</xsd:element>

<xsd:element name="adresse">
<xsd:complexType>
     <xsd:sequence>
          <xsd:element ref="pays"/>
          [...]
          <xsd:element ref="num"/>
     </xsd:sequence>
</xsd:complexType>
</xsd:element>
[...]
<xsd:element name="annuaire">
<xsd:complexType>
     <xsd:sequence>
          <xsd:element ref="contact" maxOccurs="unbounded"/>
     </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Regardons plus en détails la structure de ce morceau de code XML Schema :

- Les documents XML Schema répondent à la syntaxe XML. A ce titre, ils utilisent pleinement les *namespaces*. Ce schéma de validation fait appel au namespace pré-défini par la norme XML Schema du W3C. Tous les éléments préfixés de la variable `xsd` sont donc définis dans le namespace du W3C.
- En XML Schema, on définit les éléments et on leur associe un type, soit pré-défini, soit défini à la volée. En l'occurrence, ce schéma de validation s'attelle à définir les types dits "simples", c'est-à-dire les types pré-définis par la norme. Il est de bon usage de procéder comme cela, car vient ensuite la possibilité de définir des types dits "complexes" à partir des éléments déjà définis. Il est intéressant de remarquer que l'apport de ce *casting* apporte beaucoup plus de précision et de souplesse quant à la définition du document XML que les DTD. En effet, dans le cas qui nous intéresse, nous sommes capables de préciser que la balise `<nom>` comprendra obligatoirement une chaîne de caractères, et que la balise `<num>` s'attend à contenir exclusivement des entiers.
- Illustrons maintenant la possibilité qu'offre le langage XML Schema de préciser le format d'une chaîne de caractères que l'on attend. Il s'agit ici de la définition de la balise `zip`. Prenons comme postulat que le carnet d'adresses que nous tentons de définir en XML sera utilisé uniquement au Canada.

Dans ce cas, nous pouvons tenir compte, dans notre validation, du format du code postal et imposer qu'il corresponde bien au format canadien, à savoir LCLCLC. A l'aide d'XML Schema, nous pouvons préciser qu'une chaîne de caractères est attendue, et lui associer une restriction, en l'occurrence que la chaîne de caractères doit bien correspondre au format LCLCLC.

– La section suivante du code ici représenté définit un type complexe, en l'occurrence la balise `<adresse>`. Celle-ci contient exclusivement des éléments XML préalablement définis dans le code. On précise, à l'aide du type complexe via le mot clé `xsd:sequence`, la séquence d'éléments que doit contenir la balise `<adresse>`. Chaque élément de la séquence fait référence à un élément défini plus haut dans le code.
– La dernière section du code présenté définit l'élément racine, ainsi nommé car compris dans aucun autre élément. Il s'agit de la balise `<annuaire>`. Notons que nous pouvons préciser le nombre d'occurrences possibles d'un élement au sein de la séquence. Cela se fait à l'aide du mot-clé `maxOccurs`. L'attribut `unbounded` signifie que la balise `<contact>` peut se répéter indéfiniment au sein de la balise `<annuaire>`. En XML Schema, lorsque l'on définit une séquence, les attributs `minOccurs` et `maxOccurs` ont par défaut la valeur 1.

Nous pouvons observer que le langage XML Schema palie à beaucoup de faiblesses des DTD. La principale de ses qualités est de permettre une définition précise des données attendues. Avec les DTD, nous n'aurions pu, par exemple, préciser que la chaîne de caractères `HF56HG` n'est pas un code postal canadien valide.

Hormis les deux langages précédemment définis, on trouve plusieurs alternatives, dont les principales et les plus couramment utilisées sont le langage *Relax NG* et le langage *Schematron*, que nous ne présenterons pas dans le cadre de ce mémoire.

4.7 Les analyseurs XML

Dès lors que nous avons présenté le langage XML, énoncé ses concepts, décrit ses documents et présenté la manière dont nous nous assurons qu'ils sont valides, nous pouvons aborder l'étape ultime : l'analyse ou *parsing*.[8]

L'analyseur est chargé de lire le document XML, et de le valider sur base du XML Schema ou du fichier DTD fourni en paramètre. Les schémas peuvent être locaux ou distants, via les namespaces, et les DTD peuvent être inclus dans le fichier XML ou pas.

En terme d'analyseurs XML, deux standards s'affrontent :

1. **SAX** — ou *Simple API for XML* — est une méthode qui consiste à traiter les éléments du document les uns après les autres au fur et à

[8] Afin d'éviter la redondance de termes anglais dans cette section, nous utiliserons analyse pour parsing et analyseur pour parser.

mesure du parcours du fichier XML. Lorsqu'il identifie une unité lexicale, une fonction de callback est appelée et l'événement est traité.

2. **DOM** — ou *Document Object Model* — fonctionne d'une manière tout-à-fait différente. Il représente le document XML dans un arbre, utilisant la structure d'arbre intrinsèque des documents XML.

Les deux techniques ont donc une approche radicalement différente et, par voie de conséquence, ne seront pas utilisées dans les mêmes circonstances.

La méthode SAX va parcourir le fichier XML et va appeller la fonction correspondante lorsqu'elle rencontre une unité lexicale. Ceci est avantageux, à plusieurs titres. Premièrement, cette manière de procéder ne prend que très peu d'espace mémoire. En effet, on ne traite qu'une seule (et petite) partie du fichier simultanément. Deuxièmement, si l'information que nous sommes en train de traiter est une information particulièrement demandée, nous y gagnons en terme de suppression de parcours inutiles, de swap-in et swap-out en mémoire. Enfin, le document commence à être analysé dès que l'analyseur démarre.

De son côté, DOM a aussi des avantages. Le principal étant que le document entier est chargé en mémoire. Les applications peuvent donc parcourir le document, entièrement ou non, dès qu'elles le veulent. Cependant, s'il s'agit d'un avantage, il s'agit aussi d'une faiblesse. Charger tout un document en mémoire peut s'avérer très problématique si les données sont très volumineuses. Et cela arrive fréquemment en XML, nous en parlerons dans la prochaine section de ce chapitre.

Nous pouvons bien sûr minimiser ce problème avec des techniques de type *lazy*, mais le problème n'est pas résolu si nous voulons accéder fréquemment à des informations diverses. Or, c'est dans ce cas précis que la méthode DOM paraît la plus efficace de prime abord. De plus, il se peut que le programme opérant sur un gros fichier XML n'ait besoin que d'une petite partie de celui-ci. Quel est alors l'intérêt d'occuper tout cet espace mémoire ?

La méthode choisie dépendra donc du contexte d'utilisation. Il faudra trouver un compromis, car aucune solution n'est parfaite. Et les tâches à accomplir sont rarement d'un type ou de l'autre, mais bien d'un mélange des deux. A ce titre, la société SUN, via son équipe de développement JAVA a fait des tests en la matière[9].

Ce test s'est fait dans le cadre du développement d'API java. Le document XML représentait un ensemble de configurations possibles aux échecs. Ils ont testé les fichiers XML comprenant de 10 à 5000 configurations, afin de dégager l'influence de la taille du fichier sur ces méthodes.

Le test montre clairement que plus la taille du fichier augmente, plus SAX est efficace par rapport à DOM. Gageons que si la quantité de données avait été relativement réduite, et les accès au sein du fichier divers et variés, DOM aurait montré de meilleurs performances.

[9]source : `http://java.sun.com/developer/technicalArticles/xml/JavaTechandXML_part2/`

D'autres méthodes existent, citons StAX, qui utilise une technique de récursivité descendante, mais nous choisissons de ne pas les détailler ici.

4.7.1 XSLT

Nous ne pouvons quitter cette présentation du XML sans aborder le XSLT. En effet, nous allons largement utiliser ce langage dans le cadre de ce mémoire.

XSLT signifie *eXtented Stylesheet Language Transformations*. L'objectif de ce langage est de transformer un document XML en un autre document XML, du XHTML par exemple. Il est un dérivé de la famille des langages XSL, qui comprend le XSLT, le XSL-FO (qui permet de formater un document XML, il est notamment utilisé pour la génération des documents PDF) et le langage XPath, qui permet de parcourir un document XML à travers ses différents noeuds.

Une feuille de style XSLT prend donc en paramètre un document XML, et produit un autre document (XML, HTML) dans un formalisme différent. Ceci est particulièrement utile lorsque l'on veut qu'un site internet affiche un document XML. Nous pouvons ajouter à l'XML brut une mise en page, à l'aide d'une feuille externe CSS par exemple.

Voici un exemple de l'utilisation du XSLT. Considérons à nouveau notre exemple XML de la page 30, et transformons-le en langage HTML. Voici le code XSLT effectuant cette transformation :

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/annuaire">
   <html><head><title>Exemple de tranformation</title></head>
   <body><h1>Mon annuaire</h1>
   <table border="1">
   <xsl:apply-templates/>
    </table></body></html>
<xsl:template/>
<xsl:template match="contact">
   <tr>
   <td><xsl:value-of select="nom"/></td>
   <td><xsl:value-of select="prenom"/></td>
   <td><xsl:value-of select="adresse/num"/>,
   <xsl:value-of select="adresse/rue"/><br/>
   <xsl:value-of select="adresse/zip"/> 
   <td><xsl:value-of select="tel"/></td>
   </tr>
</xsl:template>
</xsl:stylesheet>
```

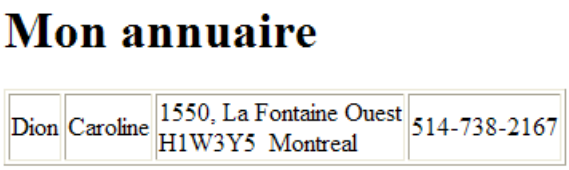

FIG. 4.1 – Aperçu de la transformation XSLT dans Firefox

Le langage XSLT est relativement intuitif à comprendre. Tout d'abord, nous voyons qu'il ne s'agit ni plus ni moins que d'un document XML. Nous précisons le format de sortie, en l'occurrence le HTML. Ensuite, nous nous déplaçons à travers le fichier source XML à l'aide de la syntaxe XPath. Rappelons que ces langages sont étroitement liés. Quand nous faisons un "match" d'un noeud du document XML, nous devons préciser quelle sera la sortie HTML correspondante.

Ici, quand nous commençons la lecture de notre document XML, au niveau de la balise `<annuaire>`, nous écrivons l'en-tête du fichier HTML, et un titre. Ensuite nous commençons un tableau. Le mot clé XSLT `apply-templates` signifie que nous allons insérer à cet endroit précis le code généré par les fils du noeud courant, en l'occurrence, le code produit par la transformation de notre balise `<contact>`. Ensuite, nous pouvons fermer les balises précédemment restées ouvertes.

Le "match" de la balise `<contact>` produit simplement une ligne de tableau par contact. Si plusieurs contacts avaient été présents dans le fichier, plusieurs lignes du tableau auraient été rajoutées.

Le caractère ` ` représente un espace en XML. Il s'agit du numéro du caractère espace en unicode. Etant donné que notre output est en HTML, le processeur XSLT le remplacera automatiquement par ` `, qui est l'espace insécable de l'HTML. La page HTML ainsi produite est illustrée à la figure 4.1.

4.8 Conclusions

Dans ce chapitre, nous avons présenté le langage XML. Il sera particulièrement utilisé dans le cadre de ce mémoire. Premièrement, nous travaillerons avec des données exprimées en XML, et nous les traiterons à l'aide d'une feuille de style XSLT. Deuxièmement, nous identifierons une zone géographique à l'aide d'un document GML, qui est un dérivé du XML.

Le prochain chapitre présente le langage GML, du moins certains de ses aspects, utiles à la bonne compréhension du travail réalisé.

Chapitre 5

Le langage GML

5.1 Introduction

Le langage GML, ou *Geography Markup Language*, est relativement récent ; il a moins de dix ans. Il a été imaginé et proposé par l'OGC, l'*Open Geospatial Consortitum*. Il est écrit en XML et, à ce titre, est validé par un schéma XML. Ce langage permet de décrire différentes entités géographiques, et est de plus en plus utilisé en représentation spatiale.

Son caractère ouvert, et sa compréhension relativement aisée, en fait une nouvelle alternative face aux techniques propriétaires. Nous allons le présenter, énoncer ses concepts fondateurs mais aussi discuter des avantages et inconvénients allant de paire avec son caractère textuel.

5.2 Présentation

Le langage GML est né du besoin de fournir aux compagnies et organisations une possibilité de faire évoluer leurs données géographiques. En effet, avant l'an 2000, les organisations ayant recours à la géomatique étaient cantonnées à un format de stockage défini, propriétaire. Cela avait des conséquences problématiques.

Premièrement, ces données ne pouvaient être analysées qu'avec un nombre limité d'outils, car seuls les outils proposés par le fournisseur étaient compatibles avec le format de données.

Deuxièmement, la transposition des données vers un autre format était particulièrement fastidieuse, excluant dans beaucoup de cas la possibilité de changer de système propriétaire [15]. Les compagnies étaient donc contraintes de garder leur système, et ce, même s'il ne correspondait plus à leurs desiderata.

Enfin, cette situation de monopole rendait impossible le partage de données entre organisations ce qui, compte tenu du volume considérable de données disponibles, était un inconvénient majeur. Parmi les formats propriétaires po-

pulaires, on peut citer SHAPE (développé par ESRI) et MIF(*Mapinfo Interchange Format*) [16] développé par MapInfo Corporation, aujourd'hui détenue par Pitney Bowes.

Il était donc nécessaire que des compagnies intéressées par le traitement de données géo-spatiales s'unissent et se concertent afin de dégager des pistes de travail tendant vers une harmonisation du secteur. C'est dans ce contexte que fut créé le langage GML.

Le langage GML a été introduit par l'OGC qui est l'acronyme d'*Open Geospatial Consortium*. Il regroupe plus de 250 compagnies, agences gouvernementales et universités, ayant pour but de développer un langage qui permet le traitement, le transport, la modélisation et le stockage d'informations à teneur géographique. L'OGC définit lui-même le langage comme suit[1] :

> "*Le GML est un langage écrit en XML pour la modélisation, le transport et le stockage d'informations géographiques incluant les propriétés spatiales et non-spatiales inhérentes aux données géographiques*".

Le langage est basé selon plusieurs principes fondateurs, parmis lesquels[1] :

- procurer un framework ouvert, indépendant pour la définition des schémas d'application géo-spatiaux,
- supporter le stockage et le transport de schémas d'application et d'ensembles de données,
- augmenter les capacités qu'ont les organisations de partager des schémas d'application et l'information qu'ils décrivent,
- supporter la description de schémas d'application géo-spatiaux pour des domaines spécialisés et des communautés d'information,
- permettre l'existence de profils composés de sous-ensembles des capacités descriptives du GML.

Il convient ici de préciser deux termes employés dans le paragraphe précédent. Un schéma d'application GML est un ensemble de règles GML dédiées à un domaine particulier. Le schéma d'application permet de valider le document GML (il n'est autre qu'un XML Schema, dont nous avons parlé dans le chapitre précédent) et il permet au logiciel considéré d'interpréter le fichier GML. Ceci constitue à notre sens une force mais aussi une faiblesse du langage GML, mais nous y reviendrons plus tard.

Contrairement aux schémas d'application, qui vont extraire du langage GML les seules possibilités qui les intéressent, les profils sont un ensemble pré-défini de fonctionnalités du GML. Il s'agit en fait de restrictions sur le langage, particulièrement efficaces car le langage est conçu de manière à toucher le plus grand nombre de domaines d'application possibles. Ces profils sont publics et dédiés à des usages spécifiques, rendant plus simple et plus attirant l'emploi du GML. Parmi les profils les plus connus, on peut citer le *point profile*, concernant les applications avec données géométriques qui n'ont pas besoin de toute la

[1] source : `http://www.opengeospatial.org/standards/gml`

grammaire GML, *simple geometry*, dont la grammaire contiendra seulement les figures géométriques basiques du GML, et le *web publishing*, reprenant quelques outils utiles à la diffusion sur internet.

Le GML est relativement récent. Il est tiré de l'*OpenGIS abstract specification*[2], qui définissait, de manière conceptuelle et abstraite, les fondements qui devaient conduire à un projet d'inter-opérabilité des données géographiques. Ces recommandations ont servi de modèle de référence à la création du GML.

Le GML est né de la collaboration de deux compagnies, toutes deux membres de l'OGC : LizardTech, basée à Seattle aux Etats-Unis et Galdos Systems Inc., basée à Vancouver au Canada. Chacune de ces compagnies a apporté son expérience dans un domaine de compétence particulier ; la première était spécialisée dans la compression pour les technologies d'imagerie géo-spatiale basée sur JPEG2000 (qui est une spécification de format de fichier améliorant le format JPEG) et la deuxième développait GML en vue de parfaire les systèmes d'information géographique, ou GIS, dont nous avons parlé précédemment dans ce mémoire.

5.3 Description

Nous l'avons dit plus haut, le but premier du GML est de décrire le monde réel, notamment suivant des critères géométriques et topographiques. Il permet de décrire des entités géographiques (un building, une autoroute, une route, un fleuve, ...), des informations de couverture, des observations, des unités de mesure (longueur, température, temps, ...), ou encore un système géodésique (axes de références).

L'élément principal d'un document GML est la "*feature*". Elle décrit un élément du monde réel (un bâtiment par exemple) et on peut lui associer un ensemble d'attributs (nom, valeur, type). Certaines propriétés peuvent être géométriques comme ses coordonnées, ou la couverture de cette feature. En effet, une feature peut être composée d'un ensemble de features. Par exemple, une ville comprenant routes, maisons, rivières, ... est aussi une feature et on imagine aisément qu'on lui associe une couverture, qui est définie géométriquement.

Le GML permet également de représenter plusieurs figures géométriques, telles que des sphères, des cylindres, des triangles, ... Les surfaces peuvent être définies selon des figures pré-établies, ou des polygones, qui consistent en un ensemble de segments à définir via un système de coordonnées.

Autre concept important du GML, les observations. Une observation représente ce que l'on a pu capter, par exemple à l'aide d'une caméra. L'observation est une feature à laquelle on associe une date, et l'observation ne vaut que pour cette date.

Voici un exemple succinct de code GML qui illustrera quelques-unes des possibilités offertes par ce langage et qui permettra de se rendre compte de ce

[2] `http://www.opengeospatial.org/standards/as`

qu'est un schéma d'application. Cet exemple est basé sur le schéma d'application `city.xsd`, provenant des exemples proposés par l'*Open Geospatial Consortium*, dont nous avons parlé plus haut. Ce schéma d'application s'attelle à fournir les outils nécessaires pour décrire une ville. Des libertés ont été prises par rapport au schéma originel, ceci afin d'illustrer la notion de couverture, dont nous nous sommes servis dans ce mémoire.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemple de code GML -->
<CityModel xmlns="http://www.opengis.net/examples"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/examples City.xsd">
```

Cette partie de code illustre très bien ce qu'est le GML et comment il fonctionne. La première ligne est l'en-tête classique d'un document XML, ce qui démontre que le GML n'est ni plus ni moins qu'un document écrit en XML. A ce titre, il peut être validé à l'aide des analyseurs XML, il suffit de préciser le schéma face auquel nous désirons valider le document.

Ensuite commence notre document GML proprement dit. Cette première balise, `<cityModel>` est particulièrement intéressante. Ses attributs font appel aux *namespaces* (voir chapitre précédent) qui permettent d'employer les balises GML au sein du document. Toutes les balises commençant par `<gml:>` seront vérifiées et validées à l'aide des schémas du langage GML. La dernière ligne permet de lier le présent document à un XML Schema qui validera ce document.

Il est intéressant de remarquer que la première balise n'est pas une balise GML. Ceci illustre véritablement la manière dont on se sert du GML. Chaque domaine d'application aura ses propres balises XML, validées par un schéma correspondant. C'est cela que l'on entend par schéma d'application. Au sein de ces balises XML, qui seront comprises et interprétées par une ou plusieurs applications dédiées, on trouvera des balises GML, pré-définies par la norme du langage, qui serviront à décrire le plus fidèlement possible des données géographiques.

```
 <gml:name>Montreal<gml:name
 <gml:boundedBy>
<!--City Bounding Box-->
   <gml:box srsName="http://www.opengis.net/gml/srs/
       epsg.xml4326">
     <gml:coord><gml:x>-75.0</gml:x><gml:y>46.0</gml:y>
     <gml:coord><gml:x>-73.0</gml:x><gml:y>45.0</gml:y>
  </gml:box>
   <gml:polygon>
     <gml:outerBoundaryIs>
       <gml:LinearRing>
```

```
            <gml:coordinates xmlns:gml="http://www.opengis.net/
                gml" decimal="." cs="," ts=" ">
            -73.33,45.33 -73.27,45.28 -73.18,45.15 -73.0,45.13
            -72,94, 45.0 -72.89,45.76<!--...--> -73.33,45.33
            </gml:coordinates>
          </gml:LinearRing>
        </gml:outerBoundaryIs>
    </gml:polygon>
</gml:boundedBy>
```

Nous rentrons ici dans la description (raccourcie) de notre ville, conformément à ce qui est défini dans le fichier distant `city.xsd`. Nous précisons le nom de la ville, Montréal, et définissons son "*Bounding Box*", que l'on pourrait traduire par rectangle recouvrant. Il est de bon usage de recourir aux rectangles recouvrants afin d'accélérer la recherche des zones à considérer. En effet, si nous définissons la ville uniquement à l'aide d'un polygone épousant ses contours, la zone à afficher par le logiciel sera moins facilement et moins rapidement identifiée. En précisant un rectangle recouvrant, nous pouvons afficher ce rectangle de carte, aller chercher les données s'y rapportant plus rapidement, ... Un inconvénient de cette méthode est que nous allons inclure des informations dont nous n'avons pas besoin, mais la pratique reste efficace.

Ensuite vient la définition des limites de la ville proprement dites, à l'aide d'un polygone (abrégé ici). Comme cité plus haut, il existe en GML plusieurs manières de définir une zone de couverture. Dans le cas d'une ville, la plus précise est évidemment le polygone. On utilise pour cela la balise `<gml:LinearRing>`, qui est un ensemble de segments de droites formant un espace fermé, c'est-à-dire que le premier point du premier segment doit être le même que le dernier point du dernier segment. Nous pourrions, au besoin, définir d'autres zones, afin par exemple d'inclure des îles, ce qui serait particulièrement utile dans le cas de la ville de Montréal.

```
<citymember>
  <Road>
    <gml:name>Blvd.  St-Laurent</gml:name>
      <gml:linearGeometry>
        <gml:LineString
        srsName=http://www.opengis.net/gml/srs/epsg.xml4326">
          <gml:coordinates>-73.333,45.333 -73.339,45,331
          <!--...-->-73.322,45,330</gml:coordinates>
        </gml:LineString>
      </gml:linearGeometry>
    <gml:classification>Avenue</gml:classification>
  </Road>
</citymember>
```

Illustrons à présent la dernière partie de ce code GML. Il s'agit d'ajouter des éléments à la ville via la balise `<citymember>`. En l'occurrence, nous nous contenterons d'ajouter un boulevard, que nous représenterons à l'aide d'un succession de segments à l'aide de la balise `lineargeometry`.

Cet exemple GML illustre également bien la notion de "*features*". La balise `<citymember>` est une collection de features, alors que le boulevard décrit est une feature.

5.4 Evolutions

La première norme du langage, GML 1.0, fut proposée en l'an 2000. Depuis, le langage a subi beaucoup d'évolutions qui se sont avérées nécessaires à l'usage. La présente chronologie est notamment inspirée du site internet de la société Galdos Inc., créatrice du langage[3].

Le concept fondateur était de créer un langage capable d'exprimer, d'une manière universelle, tous les différents aspects géo-spatiaux que l'on pouvait rencontrer. On s'est assez vite aperçu que cela n'était pas réaliste. Selon les domaines d'application, les termes utilisés n'étaient pas les mêmes, et il paraissait irréalisable de contenter tout le monde.

On a donc, avec les normes 1.2 et 1.3, autorisé l'utilisateur à ajouter son propre vocabulaire, son propre contexte d'utilisation. Cela a pu s'opérer en séparant les schémas de validation du document proprement dit. Dans la norme 1.2, on utilisait les DTD, que nous avons présenté précédemment. Cela autorisait les utilisateurs à rajouter leurs propres balises et à se servir du GML au sein d'un environnement qui leur était propre. Dans la norme 1.3, on utilisa les RDF/S, un autre langage de schéma, que nous ne détaillerons pas dans le présent mémoire.

Le RDF/S est peu à peu tombé en désuétude et, dans la norme 1.3, le XML Schema que nous connaissons aujourd'hui a été introduit dans la norme. Comme nous l'avons évoqué lors de notre présentation sur le XML Schema, cela a permis encore plus de flexibilité dans les documents GML. Il devenait possible de se créer un environnement répondant fidèlement et précisément à ses besoins, au prix d'un écartement progressif des valeurs fondatrices. En effet, on s'éloigne du principe d'un schéma universel compris et géré par tout le monde, mais nous y reviendrons.

La norme 2.0, publiée par l'OGC en février 2001, voit peu de changements. Le XML Schema est adopté et on voit apparaître la gestion des *namespaces* provenant d'XML. La norme 2.0 prévoit trois schémas de base, toujours d'actualité. Le schéma *geometry* remplace le DTD de la version 1.0. Ce schéma permet d'ajouter des propriétés géométriques aux *features*. Un schéma de *features*, avec des propriétés plus avancées, et un schéma *Xlinks*, qui permet de créer des liens entre différentes *features* [17].

Très peu de changements sont à mettre au crédit des normes 3.0, 3.1.1(l'ac-

[3] `http://www.galdosinc.com/archives/315`

tuelle). Les changements concernent majoritairement la syntaxe, mais on s'est assuré que la norme 2.0 serait toujours comprise par la norme 3.0 (compatibilité vers l'arrière ou *backwards compatibility*).

5.5 Le mapping

Dans ce chapitre consacré au GML, nous avons décrit les puissantes possibilités qu'offre ce langage en terme de représentation de données géo-spatiales. Représentation oui, mais textuelle uniquement. En effet, le GML n'est ni plus ni moins qu'un fichier de texte, et il convient, afin d'en exploiter les possibilités, de mettre en place un mécanisme de représentation de ces données sur des cartes géographiques. Ces techniques font partie de ce que l'on appelle le *mapping*. Nous ne sommes pas parvenus à trouver une traduction satisfaisante de ce mot, nous continuerons donc à l'utiliser sous cette forme à l'avenir. Le mapping consiste donc à lire un fichier GML purement textuel, et à associer aux éléments lus des éléments d'affichage graphique dans des formats tels que SVG (*Scalable Vector Graphics*), VMLVector Markup Language, et d'autres encore.

5.6 Critiques du langage

Nous avons vu les possibilités indéniables de ce langage pour la description de données géographiques. Il convient cependant de rester critique, car si ce langage est un apport indéniable dans le monde des systèmes d'informations géographiques — nous allons en rappeler les raisons — il comporte aussi de nombreux défauts, qu'il est bon de garder à l'esprit lors de la mise en place d'un tel système. En effet, les systèmes de représentation géographique sont onéreux, traitent une quantité généralement phénoménale de données, et le choix de l'architecture et des outils à utiliser est capital lors de la conception de tels systèmes.

5.6.1 Qualités

Le GML permet l'affichage sur carte de manière très précise, au niveau des coordonnées par exemple. Les cartes générées à partir du GML sont fidèles à la réalité, et permettent une visualisation efficace des données.

Comme nous l'avons vu dans la section précédente, le code GML peut être lu par un navigateur internet via, par exemple, le format graphique SVG, ce qui permet de se passer de logiciels propriétaires, souvent onéreux et restrictifs. De plus, le potentiel de ces *Web Services* est énorme, et nous n'en sommes qu'au début. N'oublions pas aussi que le GML est un langage indépendant de toute société commerciale, ce qui permet de créer, avec l'aide d'autres outils, des solutions SOLAP totalement gratuites.

Le GML reste générique. Il précise quoi afficher, mais ne précise pas comment

l'afficher, ce qui laisse une complète flexibilité quant à la visualisation des données et à l'interaction que l'on peut avoir avec celles-ci.

Comme nous l'avons vu, un aspect très intéressant de ce langage est qu'il s'agit ni plus ni moins de code XML. Ceci est avantageux à plusieurs points de vue :

- le langage XML est populaire, connu de nombreux informaticiens. Quand bien même il ne serait pas maîtrisé, son apprentissage est des plus aisés.
- Beaucoup d'outils existent déjà. C'est le cas des analyseurs, étant donné que nous pouvons valider un fichier GML à l'aide des analyseurs XML.
- Le GML a donc hérité de la flexibilité du XML. On peut aujourd'hui représenter ce que l'on veut avec du GML, il s'applique à tous les domaines, pour peu qu'on se crée le schéma d'application correspondant à ses propres besoins.

5.6.2 Défauts

Cependant, cette faculté à pouvoir s'adapter à tous les domaines peut aussi représenter un défaut. En effet, on s'éloigne largement du concept fondateur qui aspirait à créer un langage universel permettant d'afficher tous types de données géographiques. En effet, avec l'arrivée des XML Schema dans la norme, chaque application a écrit son fichier GML, pouvant être lu dans un environnement bien particulier. Ceci aboutit à l'apparition de documents GML différents, et ce même s'ils décrivent les mêmes données. Ceci constitue un frein à l'interopérabilité et au partage des données. Un pré-traitement sera systématiquement nécéssaire. Les profils que nous avons évoqués plus tôt dans cette section ne résolvent que partiellement ce problème.

Un autre problème majeur du GML est qu'il s'agit d'un fichier texte. Or, les données géographiques sont souvent très volumineuses et la manipulation de tels fichiers est lente et pose de gros problèmes de mémoire. L'analyse de ces fichiers est très coûteuse comparée aux fichiers binaires.

5.7 Conclusions

Dans ce chapitre, nous avons présenté le GML, langage dont nous nous servirons pour l'application relative à ce mémoire. Armés de ces informations, nous allons pouvoir rentrer dans le vif du sujet, à savoir la création d'une application permettant, à l'aide de GML et du XML, la visualisation, sur une carte, de données extraites d'un système Oracle.

Le prochain chapitre présentera les différents outils sur lesquels nous avons travaillé à cette fin.

Chapitre 6

Les outils utilisés

6.1 Introduction

Ce chapitre nous permettra de présenter le logiciel dont nous nous sommes servis pour la représentation des données dans le cadre de ce mémoire. Nous présenterons la communauté qui a permis sa création, décrirons son architecture et les différents outils existants dont le logiciel tire les avantages.

Nous évoquerons ensuite les motivations qui nous ont conduits à choisir ce logiciel plutôt qu'un autre. Enfin, nous présenterons succinctement la suite d'outils Oracle, et énoncerons les lacunes qu'elle comporte en termes d'applications SOLAP.

6.2 Le GML Viewer

6.2.1 Présentation

Le programme dont nous nous sommes servis pour la représentation des données s'appelle "GML Viewer". Il s'agit en fait d'un programme permettant le rendu de fichiers GML au sein d'un navigateur internet. Il s'agit d'une application dérivée du logiciel MapBuilder, maintenu par la communauté MapBuilder (*Community MapBuilder*)[1].

Il est important à ce point de ne pas confondre la communauté précitée avec le programme MapBuilder également disponible sur internet[2]. Celui-ci permet l'affichage de cartes provenant de Google ou de Yahoo dans une page HTML. Différents outils de localisation sont proposés (sélection via une adresse, un code postal) et le logiciel permet également d'apposer des "punaises" (*pin*) sur la carte, afin de mettre en évidence une adresse particulière. Cet outil est particulièrement utilisé dans les sites commerciaux, afin, par exemple, d'identifier clairement la localisation d'un restaurant.

[1] `http://communitymapbuilder.org`
[2] `http://www.mapbuilder.net`

Le GML Viewer est distribué sous une licence LGPL (*Lesser General Public Licence*). Cette licence est une version "assouplie" de la GPL. La différence fondamentale tient en ce qu'il est possible, sous LGPL, d'utiliser des librairies propriétaires. Et il est également possible que le programme créé sous LGPL soit modifié et adapté aux besoins d'une société qui pourra alors commercialiser le produit ainsi créé. Nous avons perdu le caractère héréditaire de la GPL, qui exigeait que toutes les modifications effectuées sur un programme, une librairie, ... soient également distribuées sous licence GPL.

Cette forme de licence a été choisie car certaines librairies utilisées étaient sous LGPL. De plus, cela permet aux compagnies utilisant le logiciel de commercialiser un produit fini sans pour autant être contraintes de dévoiler les sources de leurs différents apports.

Les concepts sous-tendants à la création de MapBuilder sont, notamment[3] :

- la mise au point d'un logiciel de visualisation de cartes depuis un navigateur internet,
- le support des normes de l'OGC,
- la facilité d'utilisation,
- la rapidité et l'interactivité,
- la non-utilisation de plugins supplémentaires,

Ce programme est donc à voir comme un noyau de fonctions pré-définies, pouvant s'adapter aux besoins de chacun, les possibilités d'évolutions étant diverses et variées.

A ce jour, le programme est compatible avec Internet Explorer (version 6.0 ou supérieure), Firefox, et Mozilla (version 1.0 ou supérieure), ce qui représente 92.4% des navigateurs selon le W3C[4]. Ceci est un avantage absolument primordial en terme de coûts (nous n'avons pas recours à des logiciels propriétaires) et de portabilité (ces navigateurs étant disponibles sur une grande variété de plate-formes).

A titre d'exemple, voici une capture d'écran du programme, tel qu'il est disponible sur internet.

[3] `http://docs.codehaus.org/display/MAP/Home`

[4] `http://www.w3schools.com/browsers/browsers_stats.asp`

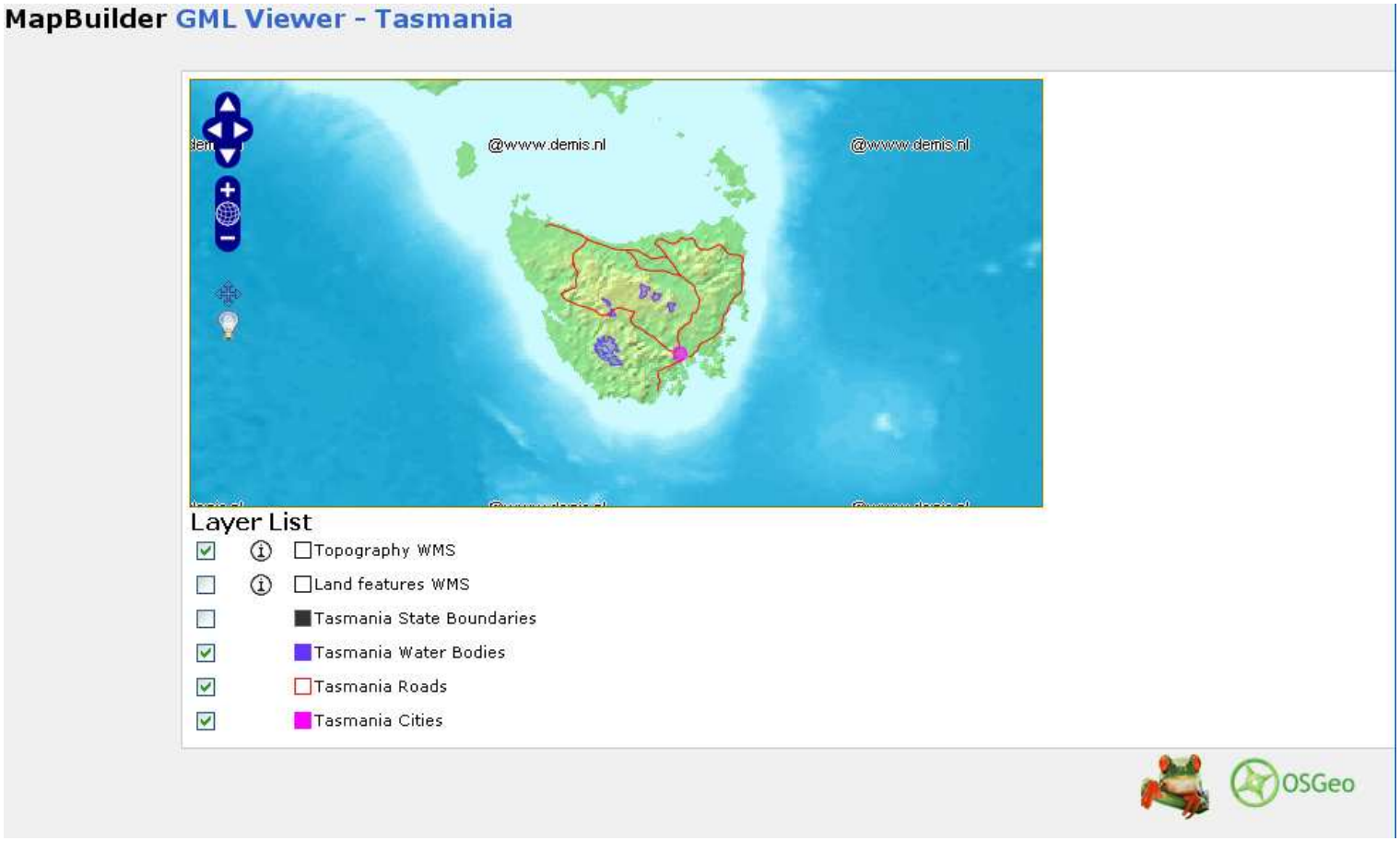

FIG. 6.1 – Capture d'écran de GML Viewer

6.2.2 Conception d'ensemble

Le programme est écrit à l'aide d'un ensemble de langages formant ce qu'il convient d'appeler désormais l'Ajax (*Asynchronus Javascript And XML*). Il ne s'agit pas d'une technologie proprement dite, encore moins d'un langage, mais plutôt de la réunion de plusieurs langages existants, libres, permettant la réalisation d'applications au sein de pages internet. L'Ajax regroupe l'utilisation des langages Javascript, XML, XSLT, HTML, CSS, DOM et l'objet XMLHttpRequest qui permet une interaction avec le serveur distant.

Le terme Ajax est relativement récent ; il a été proposé en 2005 par Jesse James Garrett, dans un article devenu célèbre[5]. Il convient de ne pas se méprendre ; si la terminologie est récente, les technologies utilisées, elles, sont éprouvées.

L'idée est venue d'un constat sans appel : internet évoluait à grande vitesse. Aux premières heures, les sites étaient statiques, on se déplaçait de pages en pages en cliquant sur des liens, ce qui demandait peu de ressources au serveur HTTP. L'avènement des sites dynamiques a singulièrement compliqué les choses à ce niveau. Les requêtes étaient beaucoup plus fréquentes, plus volumineuses, et pendant qu'une page se chargeait, l'utilisateur ne pouvait qu'attendre.

Pour palier à cette situation, on a imaginé un ensemble de techniques afin de soulager la charge du serveur. En effet, à l'aide des langages précités, il est possible de générer du contenu dynamique, évolutif, avec un ensemble de fichiers et de scripts présents sur la machine cliente. Ceci permet non seulement d'alléger le serveur, mais cela réduit fortement le temps de chargement de la page désirée. Les requêtes vers le serveur HTTP sont bien entendu toujours possibles, mais on évitera autant que faire se peut d'y avoir recours.

Le terme asynchrone provient du fait que les requêtes qui seraient habituellement effectuées vers le serveur sont exécutées localement ; et les nécessaires appels au serveur sont réalisés de manière asynchrone, sans que l'utilisateur final s'en aperçoive. Cette conception nouvelle des sites internet est particulièrement utilisée dans les applications à vocation géographique.

Un inconvénient notable de cette technique est qu'elle a recours à de nombreux scripts, aux effets parfois différents selon le navigateur utilisé. Mais on peut généralement contourner le problème, quitte à dédoubler une partie du code afin de rendre ces variations totalement invisibles aux yeux de l'utilisateur.

L'interface dont nous nous sommes servis dans le cadre de ce mémoire tire pleinement parti de cette nouvelle conception. A ce titre, elle fait appel à de nombreux fichiers, dont les plus importants sont expliqués dans la section suivante.

[5] `http://www.adaptivepath.com/publications/essays/archives/000385.php`

6.2.3 Fichiers principaux

Le programme dont nous nous sommes servis, GML Viewer, est, comme nous l'avons décrit plus haut, écrit en Ajax[6].

Ceci sous-entend un ensemble de relations entre différents fichiers de type javascript, css, html, xml et xslt. Nous allons ici tenter d'expliquer le fonctionnement interne du programme, à travers les différentes relations entre les fichiers principaux.

Dans un souci de concision et de clarté, seuls quelques fichiers primordiaux seront abordés, le code entier faisant appel à de nombreux autres fichiers et librairies.

L'architecture du GML Viewer est présentée sur le schéma de la figure 6.2. Les flèches décrivent les interactions entre les fichiers. Nous sommes conscients que ce schéma est incomplet, car il ne décrit pas toutes les relations et tous les fichiers, mais il permet néanmoins d'avoir une bonne vision d'ensemble des fichiers que nous allons expliquer dans cette section.

Notons que le fichier `gmlRenderer.xsl` n'est pas le seul à transformer le code GML. Différents fichiers permettent de transformer le GML afin de l'afficher dans la fenêtre du navigateur mais il serait long et fastidieux de les détailler ici. Retenons que le GML est interprété par une feuille de style qui le transforme en un format lisible par le navigateur.

index.html

Ce fichier est l'épine dorsale du programme ; il en décrit le "squelette". Son rôle est de :

- initier le bootstrapping en appellant le premier script qui lancera tous les autres,
- appeler les scripts externes de mise en page (scripts CSS),
- préciser l'endroit où se trouve le fichier de configuration que nous détaillerons plus tard,
- décrire quels sont les éléments du programme à afficher, et à quel endroit dans la page ils doivent s'afficher.

Voici quelques extraits de code qui aideront à la compréhension du fonctionnement du programme :

[6] Bien que cette terminologie ne soit pas totalement correcte (Ajax n'est pas un langage en soi), elle est largement répandue dans la littérature et nous choisissons de l'utiliser ici.

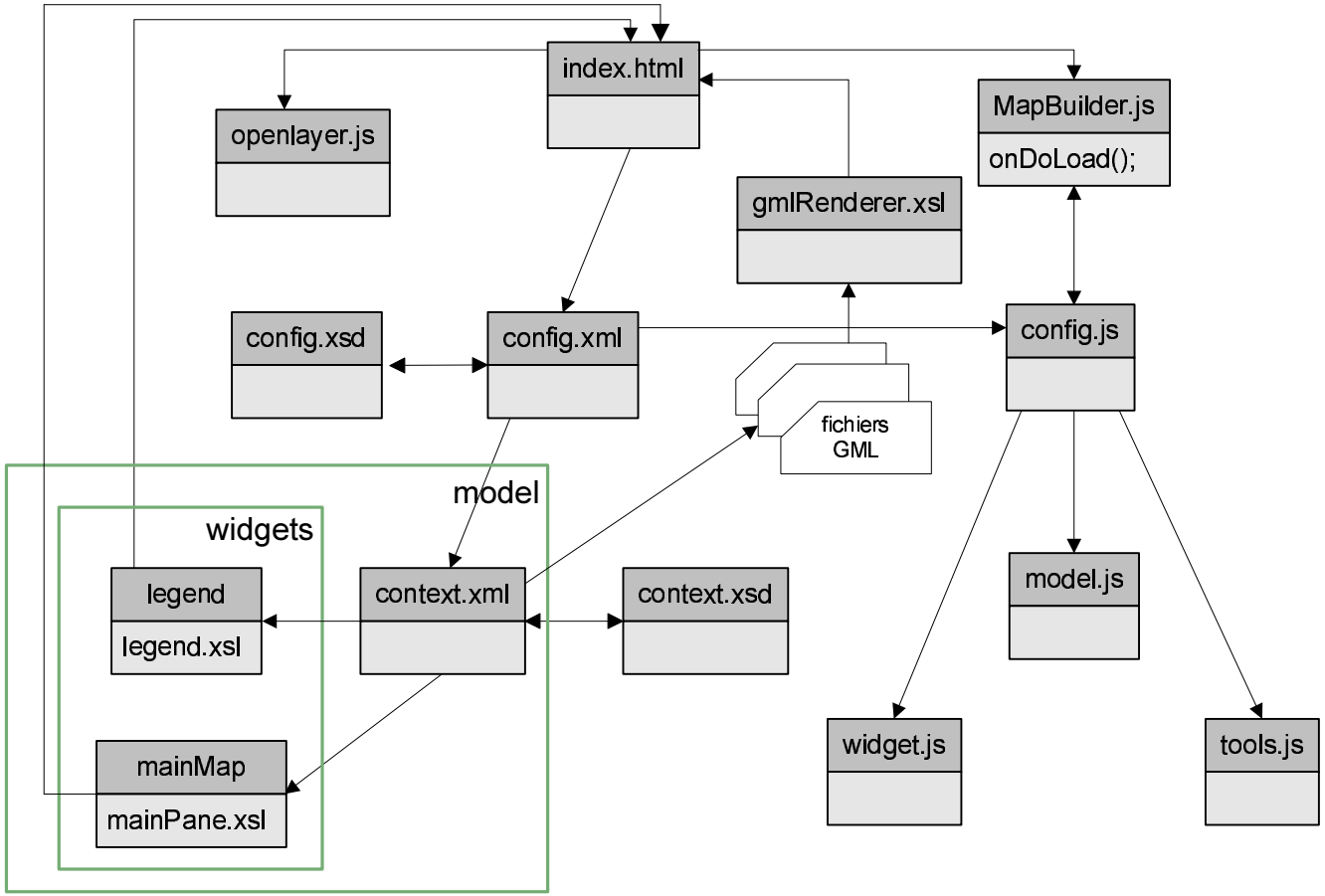

FIG. 6.2 – Fichiers principaux de GML Viewer

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<html xlmns="http://www.w3.org/1999/xhtml"
      xlmns:vml="urn:schemas-microsoft-com:vml"
      xlmns:svg="http://www.w3.org/2000/svg"
      xlmns:xlink="http://www.w3.org/1999/xlink">
<head>
<link rel='StyleSheet' type='text/css'
       href='./lib/skin/default/demoStyles.css'/>
<script src="OpenLayers.js"></script>
<script>var mbConfigUrl='belgiumConfig.xml';</script>
<script type="text/javascript" src="lib/MapBuilder.js"></script>
[...]
</head>
```

Ce morceau de code est relativement intuitif. Néanmoins, il convient de préciser certains points importants. Il s'agit d'un fichier XHTML, qui est une version du HTML où les balises sont insérées dans un fichier XML bien formé.

Dans la section `<head>` est spécifié le lien vers le fichier de feuilles de style CSS. Ce fichier comprend les styles associés aux balises présentes dans le document HTML. Enfin, des scripts sont exécutés afin de permettre le boostrapping dont nous parlons plus haut. Il est important de noter que la variable `mbConfigUrl` est assignée ; elle sera utile au script `MapBuilder.js`.

```
<body onload="mbDoLoad()">
[...]
  <div id="mainbody">
     <div id="mainMapPane"> </div>
     <div id="legend"> </div>
[...]
   </div>
</body>
```

Voici comment est articulé le corps de la page internet générée par GML Viewer. A ce niveau, il est intéressant de consulter la capture d'écran représentée en figure 6.1. On peut y voir une carte, affichée ici au sein de la balise `<div id="mainMapPane">` et une légende, affichée quant à elle dans `<div id="legend">`.

Dans le fichier de configuration que nous allons détailler, des objets sont créés via des formalismes XML, et on peut y préciser au sein de quelles balises `<div>` leur rendu devra se faire dans le fichier HTML. GML Viewer va donc insérer le code HTML généré depuis des fichiers XML via une feuille de style XSLT entre les balises `<div>` correspondantes.

C'est là tout le principe de GML Viewer, et de beaucoup d'applications Ajax. Le fichier HTML est le fichier principal, constitué de balises `<div>` chargées de déterminer où les données vont s'afficher. De plus, chaque balise `<div>` peut avoir un style différent, défini dans le fichier CSS externe.

Donc, des fichiers XML décrivent les données proprement dites, et, via XSLT, on génère la représentation des données en HTML, contenant des appels aux fonctions javascript. Ce code HTML vient s'insérer entre les balises `<div>` correspondantes précisées dans un fichier de configuration, et la page peut alors se charger.

openlayers.js

Ce script est en réalité une librairie entièrement écrite en javascript. Elle permet l'affichage de cartes dynamiques au sein d'un site internet, et ce de manière totalement indépendante du serveur qui héberge le site.

OpenLayers est un projet indépendant sponsorisé par MetaCarta[7]. La librairie est très puissante. Elle permet l'affichage d'une carte dans une page internet avec diverses fonctionnalités. Elle autorise notamment différentes couches, réactualise la carte lors de déplacements avec la souris, et permet de zoomer directement sur la carte.

Pour ce faire, elle se sert de WMS (*Web Map Service*)[8]. Ce service est proposé par l'OGC et consiste en un serveur de cartes géographiques. On envoie des requêtes selon un formalisme pré-établi, et le serveur envoie la partie de carte désirée, avec les couches demandées et ce dans une liste de formats disponibles.

De nombreuses organisations utilisent ce service et nombre d'entre-elles proposent un serveur de cartes géographiques. Citons en exemple "Ressources Naturelles Canada", qui met à disposition des utilisateurs, moyennant une inscription gratuite, une couverture complète du pays, comprenant 73 couches. D'autres couches sont disponibles encore pour certaines parties du pays seulement. La Nasa et la Marine Américaine proposent également ce type de service, gratuitement.

Le programme avec lequel nous travaillons fait appel à la société Demis, fournissant un WMS. L'obtention des cartes chez Demis est soumise à l'achat d'une licence, mais il est possible de les obtenir gratuitement, moyennant une mosaïque en surimpression sur la carte.

Plusieurs types de requêtes sont disponibles, nous allons illuster ici la requête `GetMap`, qui permet d'obtenir une partie de carte désirée.

Cette requête a été effectuée par le programme GML Viewer, alors que nous effectuions un zoom arrière.

[7] `http://www.metacarta.com`
[8] `http://www.opengeospatial.org/standards/wms`

```
http://www2.demis.nl/mapserver/request.asp?Service=WMS
&Version=1.1.0&Request=GetMap&BBox=-20,39,38,63
&SRS=EPSG:4326&Width=600&Height=300
&Layers=Countries,Coastlines,Waterbodies&Format=image/gif
```

Voici donc une requête typique effectuée vers un Web Map Service. Elle se décompose comme suit :

- Le service (WMS) est précisé, ainsi que la version employée. La version permet de préciser à quelle spécification on fait référence. Des différences apparaissent dans la formulation des requêtes selon les versions ou dans les possibilités que celles-ci offrent.
- Le nom de la requête est ensuite énoncé, en l'occurrence `GetMap`.
- Via le paramètre *BBox*, nous précisons le bounding box que nous désirons, c'est à dire la zone géographique qui doit être représentée sur la carte. En l'occurrence, nous voulons une carte couvrant la zone -20 ouest, 39 nord, 38 ouest, 63 nord.
- Nous précisons ensuite le SRS (*Spatial Reference System*), qui précise le système de coordonnées dans lequel nous nous exprimons. Les coordonnées exprimées pour le bounding box ne sont valables que dans ce contexte. Ici, nous utilisons la norme de coordonnées `EPSG:4326` (*European Petroleum Survey Group*)[9]. Ce système de coordonnées se base sur les latitudes et les longitudes définies par rapport au méridien de Greenwich et à l'équateur.
- Nous devons également préciser la taille de l'image que nous désirons recevoir. Cette taille est indépendante de la zone que l'on veut voir représentée sur la carte. Il se pourrait donc que l'on obtienne une carte déformée si le bounding box et la taille de l'image ne sont pas proprement accordés.
- Vient alors la description des couches que nous désirons. La liste de couches disponibles est généralement précisée sur le site internet du WMS.
- Enfin, nous précisons le format dans lequel nous désirons recevoir l'image. Ici, nous recevrons une image en format GIF, format idéal pour afficher l'image dans un navigateur internet.

mapBuilder.js

Ce script est principalement responsable du bootstrapping. Il assigne aussi les variables globales, et appelle les différents scripts nécessaires à la bonne exécution du programme. Le script est lancé via la fonction `onDoLoad() ;` depuis le fichier HTML principal. Il construit une instance de l'objet MapBuilder.

Voici quelques exemples de code choisis :

```
var config;
function MapBuilder(){
```

[9]Il s'agissait d'une organisation scientifique émanant du secteur pétrolier en Europe dont le but était de parvenir à la réalisation de cartes géodésiques aidant à la découverte de puits de pétrole. Cette organisation a cessé d'exister en 2005.

```
[...]
   this.loadScript(baseDir+"/util/sarissa/Sarissa.js");
   this.loadScript(baseDir+"/util/Listener.js");
   this.loadScript(baseDir+"/model/ModelBase.js");
   this.loadScript(baseDir+"/model/Config.js");
[...]
   config=new Config(mbConfigUrl);
}
```

Ce morceau de code illustre le bootstrapping dont nous parlions en début de chapitre. La fonction `MapBuilder`, qui crée une instance du programme, se charge de mettre en mémoire les scripts nécéssaires à son fonctionnement.

Le script `Sarissa.js` fait appel à la librairie Sarissa, qui permet de se déplacer au sein d'un document XML. Elle est particulièrement utile à la communauté de développement Ajax car elle règle beaucoup de problèmes d'incompatibilité entre les navigateurs.

Le script `config.js` est particulièrement important. A l'aide de la librairie Sarissa, il va parcourir le fichier de configuration, écrit en XML, détecter les objets présents dans ce fichier (modèles, widgets, ...) et fera ensuite appel au script `MapBuilder.js` pour charger les scripts correspondants à ces objets.

Nous pouvons voir que la variable `config` prend en argument la variable `mbConfigUrl` qui a été assignée au sein du document HTML. Cette variable contient l'url vers le fichier de configuration.

```
this.loadScriptsFromXpath=function(nodes,dir){
   for (var i=0; i<nodes.length; i++) {
      if (nodes[i].selectSingleNode("mb:scriptFile")==null){
         scriptFile = baseDir+"/"+dir+nodes[i].nodeName+".js";
         this.loadScript(scriptFile);
     }
   }
}
```

Ceci est la fonction qui charge les scripts correspondants aux objets déclarés dans le fichier de configuration XML.

Elle est appelée à partir du fichier `config.js` et elle reçoit en paramètres un ensemble de noeuds contenus dans le fichier XML et leur type. Elle est appelée au plus trois fois : une fois pour tous les modèles détectés, une autre fois pour tous les widgets détectés, et enfin une fois pour tous les outils détectés. Leur type (model, widget ou tool) renseigne également le répertoire dans lequel les scripts correspondants se situent. Par exemple, si le script `config.js` a détecté un "model" nommé `OwsContext`, qui est un modèle pré-défini, alors la fonction `loadScriptsFromXpath` chargera le script se trouvant à l'endroit

`./lib/model/OwsContext.js`. Les différences entre model, widget, et tool seront détaillées dans la suite.

```
if (navigator.userAgent.toLowerCase().indexOf("msie") > -1)
   checkIESecurity();
var mapbuilder=new Mapbuilder();
[...]
function mbDoLoad(){
   mbTimerId=setInterval('mapbuilderInit()',100);
}
```

La première partie de ce code détecte si le navigateur utilisé est Internet Explorer. Si c'est le cas, il convient d'appeler une fonction qui va prévenir l'utilisateur si les contrôles active X sont désactivés. Ceci montre le soin apporté à la portabilité lors de la conception de telles applications s'exécutant au sein de navigateurs internet.

Ensuite, une instance de l'objet `MapBuilder` est créée, ce qui appelle le constructeur qui charge les différents scripts comme nous l'avons vu plus haut.

La fonction `mbDoLoad()` est appelée depuis le fichier HTML. Elle s'assure toutes les 100 millisecondes que les scripts ont été chargés. Pour ce faire, elle appelle la fonction `mapbuilderInit()` qui interroge l'état de chargement via une variable globale booléenne.

Nous avons eu ici une bonne idée de la manière dont fonctionne le boostrapping du programme. Ce script travaille de concert avec le script `config.js` qui, lui, interroge le fichier `config.xml`. Voyons donc à présent comment se présente ce fichier de configuration.

config.xml

Ce fichier décrit les objets présents sur la page internet. Il ne décrit en rien l'endroit où ils se placent, ni la manière dont nous devons les afficher. Il sert de lien entre les différents objets présents, et les fichiers qui s'y rapportent.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<MapbuilderConfig version="0.2.1" id="simpleTemplate"
xmlns="http://mapbuilder.sourceforge.net/mapbuilder"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://mapbuilder.sourceforge.net/
mapbuilder lib/schemas/config.xsd">
   <models>
      <OwsContext id="mainMap">
         <defaultModelUrl>Context.xml</defaultModelUrl>
         <widgets>
            <MapPaneOL id="mainMapWidget">
               <htmlTagId>mainMapPane</htmlTagId>
```

```
            <mapContainerId>mainMapContainer</mapContainerId>
          </MapPaneOL>
          <Legend id="contextLegend">
            <htmlTagId>legend</htmlTagId>
            <stylesheet>./Legend2.xsl</stylesheet>
          </Legend>
        </widgets>
        <tools>
[...]
</MapBuilderConfig>
```

Ce fichier est probablement le plus important de tous. Une nouvelle fois, il convient de se reporter à la capture d'écran en figure 6.1. Chaque objet figurant sur la page internet doit être déclaré ici.

L'architecture est simple. On définit ce que l'on appelle un modèle ; ici le seul modèle présent est `OwsContext`. Les *OWS Context* sont des documents XML, correspondant à un schéma défini par l'OGC. Ils servent à préciser comment le navigateur doit afficher une carte, quelles couches il doit afficher par défaut, et permet de proposer différents outils d'interaction à l'utilisateur. Nous reviendrons sur ce point un peu plus tard dans cette section, dans la définition du fichier `context.xml`.

A chaque modèle doit être associé un `defaultModelUrl`. C'est le fichier XML qui va contenir toutes les données de ce modèle. Il sera transcrit en HTML via plusieurs feuilles de style XSLT, chacune correspondant à un *widget*[10].

Deux choses peuvent appartenir à un modèle : des *widgets* ou des *tools* (outils). Chacun dérive d'une classe de base, respectivement : `widgetBase.js` et `toolBase.js`.

Les *widgets* représentent des éléments de la page à afficher. Ils sont bien souvent constitués d'un fichier XML, transformé en HTML pour l'affichage via une feuille de style XSLT associée. Les outils, quant à eux, sont consacrés à la gestion des événements. Ils récupèrent les actions effectuées à la souris et au clavier sur les différents widgets présents.

Le présent modèle contient donc deux widgets : **MapPaneOL** et **Legend**. Le premier affiche la carte proprement dite, le deuxième, la légende. La capture d'écran est particulièrement parlante à ce niveau.

Chaque widget contient une balise `<htmlTagId>`. Il s'agit d'une référence vers la balise `<div>` du document HTML au sein de laquelle il faudra afficher le code HTML généré. Ceci permet d'afficher les éléments exactement où l'on veut qu'ils se situent, l'aspect étant pris en charge par une feuille de style externe déclarée au sein du fichier HTML principal.

Notons qu'il est possible de préciser une feuille de style bien précise pour un widget, cela se fait à l'aide de la balise `<stylesheet>`.

[10] Aucune traduction n'a été jugée satisfaisante, nous conserverons donc ce terme à l'avenir.

D'autres éléments sont présents sur notre page d'exemple, mais il ne nous a pas semblé utile de les détailler ici.

Ce fichier de configuration XML est validé par un schéma, nommé `config.xsd`. Le lien vers ce schéma est précisé dans le prologue du fichier (voir code).

config.js

Le fichier `config.js` est chargé de lire le fichier de configuration XML, d'identifier les éléments présents, et d'appeler les scripts correspondants. Il crée un objet `config` qui est le parent de tous les modèles. En paramètre, le constructeur prend le chemin vers le fichier de configuration XML décrit dans le paragraphe précédent. Chaque "modèle-enfant" (*child-model*) peut être référencé par `config.objects.modelId`. Par exemple, le widget chargé d'afficher la carte sera défini par `config.objects.mainMap`.

Voici quelques morceaux de code choisis, particulièrement révélateurs.

```
function Config(url) {
[...]
this.loadConfigScripts=function(){
mapbuilder.loadScriptsFromXpath(this.doc.selectNodes
       ("//mb:models/*"),"model/");
mapbuilder.loadScriptsFromXpath(this.doc.selectNodes
       ("//mb:widgets/*"),"widget/");
mapbuilder.loadScriptsFromXpath(this.doc.selectNodes
       ("//mb:tools/*"),"tool/");
```

Ce morceau de code fait appel à la fonction `loadScriptsFromXpath` que nous avons abordée précédemment. Pour ce faire, elle fait appel à la librairie Sarissa qui permet de parcourir un document XML.

Premièrement, tous les noeuds fils de la balise `<models>` sont identifiés et passés en premier argument à la fonction, ensuite on passe en deuxième argument le type de noeud dont il s'agit, dont nous avons vu qu'il permettrait aussi de déterminer dans quel répertoire aller chercher les scripts.

On procède de la même manière avec les fils des balises `<widgets>` et `<tools>`.

```
[...]
this.objects = new Object();
ModelBase.apply(this, new Array(modelNode));
[...]
```

Ceci permet de lier tous les objets créés au modèle `config` de base. Chaque nouvel objet créé dans la structure sera ajouté comme une propriété de

`config.objects`. Ensuite, nous permettons à l'objet `config` d'hériter de toutes les fonctions des objets de base, afin d'en faire un modèle à part entière.

Nous avons ici fait le tour des principaux fichiers grâce auxquels fonctionne le programme GML Viewer. Il nous reste à détailler un fichier, un des plus importants. Il s'agit du fichier de données XML correspondant au modèle.

context.xml

Le fichier `context.xml` est le fichier de référence de notre modèle `OwsContext`. Il décrit, en XML, selon les formalismes définis par l'OGC, ce qui doit se trouver sur la page HTML. Ce fichier sera lu par différents fichiers XSLT (chacun correspondant à type d'objet) et le rendu correspondant sera alors affiché dans les balises appropriées.

Voici quelques morceaux de codes pour aider à comprendre le fonctionnement de ce dernier fichier.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<OWSContext version="0.0.13" id="ows-context-tie"
xmlns="http://www.opengis.net/context"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sld="http://www.opengis.net/sld"
xmlns:ows="http://www.opengis.net/ows"
xmlns:param="http;//www.opengis.net/param"
xsi:schemaLocation="http://www.opengis.net/oc oc_0_0_13.xsd">
```

Ceci est le prologue du fichier `context.xml`. Il est important car il fait appel aux différents namespaces présents dans les fichier Ows Context provenant de la norme de l'OGC. Nous avons déjà abordé context et xlink, voyons les autres namespaces utiles au fonctionnement de notre programme. Parmi ces namespaces, l'un est particulièrement important, il s'agit du SLD (*Styled Layer Descriptor*) : c'est à nouveau une spécification de l'OGC allant de concert avec la norme WMS. Celle-ci permet d'obtenir une carte comprenant différentes couches, et la norme SLD permet de définir des propriétés de style sur chacune de ces couches. Par exemple, dans quelle couleur devra s'afficher la couche en question.

```
<General>
   <Window width="600" height="300"/>
   <ows:BoundingBox crs="EPSG:4326">
      <ows:LowerCorner>9.57875 48.26625</ows:LowerCorner>
      <ows:UpperCorner>0.03125 53.56375</ows:UpperCorner>
   </ows:BoundingBox>
[...]
</General>
```

Ceci décrit, au sein du contexte OWS, la zone d'affichage de la carte. Ces paramètres seront utilisés pour construire la requête que nous effectuerons vers le WMS que nous avons choisi, en l'occurrence `http://www.demis.nl`. Ces informations seront utilisées par le fichier `MapPane.xsl` et permettrons de générer une partie du fichier HTML comprenant une balise `<img>` dont la source sera la requête WMS ainsi construite. Les informations concernant les différentes couches désirées sont définies plus loin dans le fichier, mais le code étant particulièrement long, redondant, et de toute façon intuitif, il ne nous a pas semblé utile de le montrer ici.

```
<ResourceList>
   <FeatureType hidden="1">
   <Server service="OGC:GML" version="2.1.2" title="Local">
      <OnlineResource xlink:type="simple"
         xlink:href="Belgium/belgiumStateBoundaries.xml"/>
   <Name>belgium</Name>
   <Title>Frontiere belge</Title>
```

Au sein de la balise `<ResourceList>`, on précise les différentes couches qui vont apparaître sur la carte. Ce fichier est donc également interprété par le fichier `Legend2.xsl`, chargé de créer une légende HTML. Ceci est un morceau de code où nous avons rajouté notre couche à la carte. Le paramètre `hidden` permet d'afficher ou non la couche par défaut, au chargement de la page HTML.

Nous précisons alors le lien vers notre fichier source. Il s'agit d'une représentation de la Belgique selon un polygone contenue dans un fichier GML. Ce fichier GML sera parcouru par une feuille de style XSLT, intégrée au programme, et chargée de convertir le GML en HTML ou en SVG. La conversion de fichiers GML à l'aide du GML Viewer n'offre, à notre sens, pas assez de possibilités. Nous aurons l'occasion d'aborder le sujet dans la suite de ce mémoire.

Nous devons ensuite préciser le nom (en interne) de la couche, et le titre de la couche, devant apparaître sur la page internet. Il nous reste ensuite à spécifier les informations de style que nous désirons associer à cette couche, ce qui est fait dans la prochaine section de code.

```
<SLD>
   <sld:Fill>
      <sld:CssParameter name="fill">#333333</sld:CssParameter>
      <sld:CssParameter name="fill-opacity">0.3</sld:CssParameter>
   </sld:Fill>
   <sld:Stroke>
      <sld:CssParameter name="stroke">#666666</sld:CssParameter>
      <sld:CssParameter name="stroke-opacity">1</sld:CssParameter>
      <sld:CssParameter name="stroke-width">0.2</sld:CssParameter>
   </sld:Stroke>
</SLD>
```

Nous utilisons ici les formalismes établis par la norme SLD, dont nous avons parlé un peu plus tôt. Ceci permet de définir une couleur pour le pourtour et l'intérieur de la surface à représenter sur la carte.

Résumé

Le GML Viewer avec lequel nous avons travaillé tire toutes les possibilités des techniques de développement en Ajax. A ce titre, il utilise une multitude de fichiers, qu'il aurait été fastidieux de détailler ici.

Retenons que le programme consiste en un fichier HTML principal, au sein duquel vient se greffer du code HTML généré à partir des fichiers de données XML. Ajoutons que dans ce code HTML généré, figurent des appels à des fonctions javascript qui agissent sur les différents variables du système. Le fichier HTML est, de plus, associé à plusieurs fichiers CSS, permettant de mettre en forme l'interface avec l'utilisateur.

6.2.4 Choix en faveur de GML Viewer

Dans le cadre de ce mémoire, nous avons choisi de travailler avec le programme GML Viewer issu de la communauté MapBuilder. Plusieurs raisons ont motivé ce choix, nous allons les détailler ici.

L'utilité d'un web-service

Le premier point qui nous a poussé vers ce choix est que le GML Viewer est un programme s'exécutant au sein d'un navigateur internet. Cela faisait d'ailleurs partie, dans la mesure du possible, du cahier des charges lors des discussions préliminaires relatives à ce mémoire.

Cette conception du programme sous forme de site internet est intéressante à plusieurs égards.

Premièrement, et il s'agit là d'un avantage très important, le programme peut s'exécuter sur plusieurs plates-formes. Nous ne sommes pas dépendants d'une architecture matérielle particulière. Le navigateur Firefox, dont nous nous sommes servis dans le cadre de ce mémoire, peut s'exécuter sur des plates-formes aussi diverses que Mac OS X, Linux, Windows, Solaris, ce qui représente plus de 95% des systèmes dans le monde[11].

De plus, le programme peut s'exécuter jusque maintenant sur Firefox, Mozilla, Internet Explorer (à partir de la version 6.0). Un travail est en ce moment effectué dans la communauté pour rendre le logiciel compatible avec d'autres navigateurs, bien qu'il ne s'agisse pas là d'une priorité.

Il est également important de remarquer que ceci permet une interopérabilité du logiciel. En effet, deux organisations pourraient développer le logiciel de

[11] `http://www.w3schools.com/browsers/browsers_os.asp`

concert, et ce même si elles utilisent des navigateurs internet différents, sur des plates-formes différentes.

De plus, le programme est accessible à partir du monde entier, à l'aide d'une simple adresse internet. Pas besoin d'installer un quelconque logiciel supplémentaire, le simple navigateur dont tout le monde dispose suffit. C'est un aspect important si on considère par exemple l'initiative EduGIS[12]. Il s'agit d'un portail GIS hollandais à vocation éducative, permettant de familiariser les enfants scolarisés dans le primaire avec les différents types de cartes. Différentes possibilités sont offertes via des cartes à thèmes, permettant, par exemple, de sensibiliser les plus petits aux problèmes environnementaux. Il est capital pour ce genre d'organisations de pouvoir mettre à disposition ces outils via un site internet. Les écoles n'ont pas besoin d'un investissement supplémentaire en logiciels, licences d'utilisation, ... et peuvent consulter ces informations à l'aide d'une connexion internet, disponible dans la plupart des écoles de nos jours.

Utilisation de standards ouverts

Ce programme se base sur l'utilisation de nombreux standards, la plupart mis en place par l'OGC. Ceci est avantageux car on veut ici favoriser l'utilisation de normes établies, et ce faisant on tend à diminuer la part de marché de techniques entièrement propriétaires.

De plus, ces standards sont ouverts. Cela signifie que l'organisme responsable publie une spécification d'implémentation, et laisse libre cours aux développeurs quant à la mise en oeuvre pratique de ce standard.

C'est un point crucial par rapport aux techniques propriétaires. Les développeurs peuvent adapter à leurs besoins les différentes normes disponibles, et ce gratuitement. En effet, ce logiciel est développé à base de langages et de librairies gratuites, et libres de droits (au sens de la licence LGPL).

Données textuelles

Tous les fichiers de données sont ici textuels. Il s'agit de fichiers XML, facilement lisibles par l'utilisateur. Ceci allège les différents problèmes liés à la compréhension du code. On peut développer le programme soi-même, en s'aidant des données déjà présentes, et en comprenant l'interaction des données avec le programme.

C'est tout-à-fait utile car ce type d'outils est appelé à être développé, adapté aux besoins de chacun. L'utilisation de fichiers de données textuelles favorise la compréhension par l'exemple.

[12] `http://www.edugis.nl`

Techniques éprouvées

Même si les techniques de développement Ajax sont une idée relativement nouvelle, les langages sur lesquels elles se basent sont, quant à eux, déjà connus et éprouvés. Ceci sous-entend donc une plus grande facilité de compréhension du programme et moins d'appréhensions de la part du développeur qui se retrouve dans un environnement familier. De plus, de très nombreuses ressources sont disponibles sur ces langages, à l'exception notable du GML, auquel peu de sites internet sont consacrés.

Cette utilisation de langages connus permet une plus grande flexibilité dans le recrutement des informaticiens appelés à développer les applications, ce qui représente un grand avantage en termes de coûts de développements.

6.3 La suite Oracle

Dans le cadre de ce mémoire, nous avons tenté d'afficher des données extraites d'Oracle, au sein d'un GML Viewer. Ceci nous a paru utile car, aussi étrange que cela puisse paraître, Oracle ne propose pas de telle solution logicielle, dite SOLAP.

Dans la suite d'outils Oracle, pourtant bien fournie, il n'est en effet pas possible de trouver un programme offrant une interaction entre des données extraites de la base et une carte géographique. Oracle offre bien un logiciel permettant d'afficher une carte correspondant à des données spatiales provenant de sa base, mais aucune interaction avec la carte n'est proposée.

Ce mémoire s'attelle donc à mettre au point un embryon d'une telle solution, et ce à l'aide d'outils gratuits et de normes ouvertes. Nous allons dans cette section présenter brièvement la société Oracle, et les outils qu'elle offre dans le domaine concerné par ce mémoire.

6.3.1 Présentation d'Oracle

La société Oracle fut fondée il y a trente ans, en 1977, par Larry Ellison, Bob Miner et Ed Oates. A cette époque, aucune société n'avait commercialisé la technologie des bases de données relationnelles, et les trois fondateurs comprirent qu'il s'agissait là d'un domaine à fort potentiel de croissance. Ils fondèrent alors la SDL (*Software Development Laboratories*), et commencèrent à commercialiser un système complet et fonctionnel en 1979. Il s'agissait en fait d'un système de management de bases de données relationnelles, fonctionnant en SQL qui, lui, avait été inventé quelques années plus tôt au sein d'une division de recherche de la société IBM. Lors de la commercialisation, la société en profita pour changer de nom, en faveur de RSI (*Relational Sysems Incorporated*).

Peu avant la sortie de la version 3 en 1983, la compagnie changea à nouveau de nom et devint *Oracle Systems*. Les années quatre-vingt virent la société se

développer significativement. Elle fut la première à proposer un RDBMS (*Relational Database Management System*) fonctionnant sur des mainframes, des mini-ordinateurs, et des PC. De nombreuses applications travaillant directement sur les bases de données furent développées et rendues disponibles.

Au cours des années nonante, Oracle diversifia son offre et mit l'accent sur l'innovation. Beaucoup de changements architecturaux dans la conception intrinsèque de la base de données virent le jour, une véritable stratégie de conquête d'internet fut mise en place, comprenant notamment le support d'XML.

Enfin, au cours des années 2000, Oracle continua à diversifier son offre, surtout dans le domaine de la *business intelligence*. En proposant par exemple des technologies de type grille de calcul dans la version 10*g* de sa base de données, sortie en 2003.

En 2005, Oracle était, de loin, le leader sur le marché des systèmes de bases de données relationnelles, avec une part de marché de 48.6%[13]. Durant l'année 2006, leurs revenus étaient de 14,380 milliards de dollars, avec une marge bénéficiaire de 4,736 milliards de dollars[14].

6.3.2 Oracle 10*g*

L'idée sous-tendante de ce mémoire est d'effectuer des requêtes sur une base de données Oracle, et d'afficher les résultats sur une carte géographique. Pour cela, nous nous sommes procuré la dernière version de la base de données oracle, la version 10*g*.[15]

La version 10g, sortie en 2003, combine, comme les versions précédentes, le système de gestion de la base de données proprement dit, et le serveur d'applications permettant de lier les suites logicielles à la base de données.

L'appellation *g* provient de l'introduction de la technologie des grilles de calcul (*Grid Computing*). Ces systèmes constituent ce que l'on appelle des grappes de serveurs (*clusters*) qui seront constitués de plusieurs ordinateurs sur lesquels on pourra répartir la charge de travail. Ceci est très utile à plusieurs points de vue :

- nous pouvons répartir la charge de travail sur plusieurs machines et ce faisant, augmenter considérablement notre force de calcul. Le temps de calcul s'en voit grandement amélioré,
- les grappes de serveurs peuvent être distantes de plusieurs milliers de kilomètres, facilitant la coopération entre organisations,
- nous pouvons tirer meilleur profit du hardware existant, en répartissant équitablement la charge sur plusieurs machines.

Pour exploiter les possibilités de cette technologie, Oracle propose, en option

[13] `http://www.gartner.com/press_releases/asset_152619_11.html`

[14] `http://www.oracle.com/corporate/investor_relations/10k-2006.pdf`

[15] Le 11 juillet 2007, Oracle a sorti une nouvelle version de sa suite, la version 11*g*. Les quelques opérations effectuées sur Oracle dans le cadre de ce mémoire ne sont aucunement concernées par ce changement de version.

à sa base de données, Oracle RAC (*Real Application Cluster*) qui permet de déployer la base de données sur une grappe de serveurs.

Oracle propose également, depuis sa version 9*i* une option OLAP. Elle met à la disposition des utilisateurs et des managers de bases de données une série d'outils afin d'offrir des possibilités OLAP à leurs systèmes. Notamment, Oracle met à disposition une API de programmation en java afin de développer des applications sur les *Analytic Workspace* qui sont un concept d'Oracle permettant de travailler avec des données multi-dimensionnelles.

Oracle enregistre les données dans son système sous une forme relationnelle. Il s'agit donc d'une solution ROLAP, terme que nous avons abordé dans le troisième chapitre. Afin de pouvoir interroger les données d'une manière multi-dimensionnelle dans des systèmes complexes, Oracle met à disposition de ses utilisateurs un programme, l'*Analytic Workspace Manager*.

Il permet de se créer un espace analytique, ce qui laisse à l'utilisateur le soin de se créer ses propres cubes et dimensions et ce d'une manière logique. Ensuite, les différents cubes et dimensions sont liés physiquement aux données, situées dans le système relationnel classique.

Oracle dispose d'un programme appelé *Warehouse Builder*, qui permet de se créer des entrepôts de données à des fins analytiques. Il est composé de deux éléments centraux : le "*Design Environment*" qui concerne l'architecture du cube proprement dit, ou ce que l'on appelle les "*metadata*" et le "*runtime environment*" qui concerne lui les données factuelles. Le *runtime environment* fait le lien avec les données physiques présentes dans le système Oracle.

6.3.3 Oracle Spatial

En ce qui concerne les logiciels touchant aux dimensions spatiales, Oracle propose, par défaut dans ses distributions, le programme *Oracle Spatial Locator*. Il permet de représenter sur une carte les zones concernées par des requêtes effectuées à partir d'une base de données spatiale Oracle.

De plus, il est possible de faire des requêtes telles que : "où est le plus proche voisin ?". Mais Oracle ne propose pas tel quel une solution SOLAP, c'est-à-dire, une solution mêlant données et représentation spatiale.

De plus, les données spatiales ne sont pas toujours forcément enregistrées dans une base de données spatiale, comme nous l'avons déjà dit. En effet, une simple information de lieu, comme une colonne "pays" dans une base de données relationnelle, constitue une donnée à caractère spatial, que nous pouvons exploiter afin de simplifier la visualisation des données. C'est l'approche que nous avons tenté dans le cadre de ce travail.

Il s'agissait de permettre à des données extraites d'Oracle, de s'afficher au sein d'une interface offrant une représentation cartographique, le tout dans le cadre d'un *web service*.

Nous sommes parvenus dans le cadre de ce mémoire, à afficher les données au

sein d'une page internet, et à lier ces données à la zone géographique correspondante sur une carte, s'affichant sur la même page internet.

Il s'agit évidemment d'un début, une solution complète consisterait à afficher les données, ou du moins une représentation de celles-ci, sur la carte elle-même, mais cela sortait des objectifs fixés dans le cadre de ce mémoire.

6.4 Conclusions

Dans ce chapitre, nous avons présenté le programme que nous avons choisi pour le développement de notre application. Après avoir introduit brièvement le logiciel, nous avons expliqué sur quelles techniques il se basait. Ensuite, et pour permettre une bonne compréhension du chapitre suivant consacré aux différentes implémentations, nous avons expliqué le fonctionnement interne du programme, à travers quelques fichiers principaux et illustrations de passages de code. Enfin, nous avons précisé les raisons qui avaient motivé ce choix.

Le prochain chapitre sera consacré à l'étude des modifications effectuées sur ce logiciel afin de parvenir à une solution de visualisation de requêtes SOLAP depuis Oracle.

Chapitre 7

Mise en oeuvre

7.1 Introduction

Dans ce chapitre, nous allons présenter les différents apports réalisés afin de représenter des données extraites d'Oracle dans le GML Viewer. Fort des connaissances acquises au cours des chapitres précédents, le lecteur devrait être en mesure de comprendre les différents morceaux de codes illustrés ici. Seuls les concepts et les idées sous-tendantes seront donc expliquées, afin d'éviter les redondances d'informations.

7.2 Présentation du travail effectué

Le but ce travail était donc d'afficher des données extraites d'Oracle dans un programme offrant une vue cartographique. Il s'agissait de relier les données à la zone géographique concernée, tout en maintenant un affichage des données relativement intuitif.

Une solution SOLAP complète aurait consisté à afficher les données, ou du moins une représentation de celles-ci, directement sur la carte, mais il s'agit d'un travail très long et fastidieux sortant du cadre d'un mémoire. En effet, les seuls outils connus exécutant cette tâche, que nous avons abordés au troisième chapitre, sont tous des logiciels propriétaires. Ils se distinguent bien souvent par leur lourdeur et par leur rigidité.

Il est donc particulièrement intéressant d'étudier la possibilité de combler cette lacune à l'aide de logiciels non-propriétaires. Les techniques permettant l'affichage de données directement sur la carte sont ici encore à développer, et feront l'objet d'une brève discussion dans le chapitre suivant.

Dans un premier temps, nous nous sommes concentrés sur une application reliant les données à la zone géographique concernée. Dans notre ensemble de données, nous travaillons exclusivement sur la Belgique, et les trois régions qu'elle comprend.

Ces données ont été obtenues via le site Eurostat, qui est un organisme européen récoltant les données que les différents pays de l'Union lui fournissent. Ce site propose un outil de consultation et de récupération de ces données, sous différents formats.

Nous avons introduit ces données dans une base de données Oracle $10g$, et nous les avons interrogées en simulant une requête OLAP possible. Ensuite, un fichier de données/résultats XML a été créé, et nous avons tenté de l'intégrer dans un outil offrant des possibilités de visualisations spatiales.

Nous avons choisi d'afficher ces données dans le programme GML Viewer, et ce, à droite de la carte. Ceci permettait d'avoir la carte et les données sur le même écran, d'où un gain d'ergonomie évident. Pour ce faire, il a fallu jouer sur la taille de la carte et sur la taille de l'espace réservé aux données proprement dites. Mais nous pensons être arrivés à un bon compromis.

En ce qui concerne les données, nous avons privilégié la représentation sous forme d'arbre, qui présente l'avantage d'être très intuitive. De plus, cette représentation est relativement facile à générer en HTML.

Il nous restait à créer un lien entre données et zone géographique correspondante. Nous avons choisi de fonctionner par clic de souris. Un clic sur les données provoquera l'apparition sur la carte de la zone de couverture géographique correspondante.

7.3 Fichier de données XML

Nous allons, dans cette section, décrire les formalismes du fichier de données XML avec lequel nous avons développé notre programme. Nous avons travaillé à l'envers ; nous avons d'abord conçu ce fichier et avons développé notre programme sur cette base. Ensuite, nous avons fait en sorte qu'Oracle exporte notre requête dans ce formalisme.

Voici un extrait de notre fichier de données.

```
<file>
  <country id="belgique">
    <region id="bruxelles">
      <data year="2004">999899</data>
      <data year="2005">1006749</data>
    </region>
[...]
  </country>
</file>
```

Notons que la balise `<file>` n'apporte rien au programme proprement dit, mais nous devions disposer d'une balise racine, que nous avons choisi de nommer

`file`. Nous voyons ici la description d'un pays contenant une région. Nous aurions pu avoir plusieurs pays contenant plusieurs régions.

7.4 Export en XML dans Oracle

7.4.1 Extraction de données en XML

Nous l'avons dit, dans le cadre de ce travail, nous avons extrait des données d'un système Oracle, et nous les avons ensuite récupérées dans un programme permettant de les lier à une carte géographique.

Nous avions donc besoin de pouvoir récupérer les résultats de requêtes effectuées sur le système Oracle dans un fichier externe, afin de pouvoir le lire dans notre logiciel.

Etant donné que nous voulions, depuis de départ, travailler avec du GML (et donc avec le GML Viewer), nous avons choisi d'exporter le résultat des requêtes dans un formalisme XML, langage que notre programme comprend et est capable d'interpréter.

Nous avons travaillé à l'envers ; nous avons d'abord développé le programme permettant la visualisation des données. Ce faisant, nous avons mis au point une syntaxe XML avec laquelle il serait confortable de travailler. Après s'être assurés que la syntaxe choisie fonctionnait, nous avons fait en sorte que le fichier produit par Oracle "colle" à nos desidarata.

Oracle propose plusieurs solutions pour faire de l'export en XML à partir des requêtes en SQL faites sur son système. Dressons un rapide portrait des différentes techniques disponibles.

DBMS_XMLGEN

La solution DBMS_XMLGEN produit un document XML à partir de n'importe quelle requête SQL en transformant les résultats en XML. Le résultat est retourné sous forme de CLOB (*Character Large Object*) qui peut être vu comme une large collection de caractères. Ces fichiers ont généralement une limite de taille assez large. Chaque ligne produite par la requête est insérée entre des balises `<row>`, que des fonctions permettent de renommer.

Cette technique nous a paru assez laborieuse et peu claire. elle ne nous a pas paru offrir des fonctionnalités spécifiques justifiant son emploi pour cette tâche.

XSQL Pages Publishing Framework

Il s'agit d'un ensemble de fonctions rendant possible la publication sur internet (en HTML ou en tout format XML) de résultats de requêtes SQL. Il fonctionne par la déclaration de templates dans ce que l'on appelle les pages XSQL.

Ces pages en XSQL contiennent la requête, au format SQL, et le résultat est produit sous forme de XML, dit diagramme XML. Ce fichier XML de sortie est ensuite transformé au format désiré à l'aide d'une feuille de style XSLT. Bien que cette méthode possède des avantages évident en termes de flexibilité, sa complexité relative le rendait moins attractif et les possibilités que nous souhaitions exploiter étaient présentes au sein d'autres méthodes avec lesquelles nous nous sentions plus confortables.

XSU (*XML SQL Utility*)

XML SQL Utility permet de générer de l'XML à partir de requêtes sur la base de données, mais aussi d'extraire des données d'un fichier XML afin de les insérer dans un table Oracle. Il offre également la possibilité de modifier ou d'effacer des données dans la table à partir d'un fichier XML.

En ce qui concerne l'export en XML, il fonctionne de manière semblable à la méthode DBMS_XMLGEN. Le résultat global de la requête est inséré dans un fichier XML où chaque ligne est insérée entre des balises `<row>`. Différentes fonctions permettent de changer le nom des balises, d'ajouter des attributs, ...

Fonctions SQL

Oracle propose des fonctions SQL qui lui sont spécifiques : les fonctions `SYS_XMLGEN()`, `XMLAGG()` et `XMLSEQUENCE()`. Elles permettent de générer un format de sortie en XML.

La différence fondamentale avec la méthode précédemment décrite provient du fait que DBMS_XMLGEN produit un fichier XML à partir du résultat de la requête alors que les fonctions SQL d'Oracle travaillent au niveau de la ligne de réponse. Chaque ligne est interprétée et le XML correspondant est produit. Nous n'avons pas utilisé cette technique, mais nous avons utilisé un système presque en tout point semblable, que nous décrivons ci-après.

Fonctions SQLX

SQLX est une partie émergente du standard SQL. Il propose une manière de travailler de concert avec les langages SQL et XML. Depuis la version `9i` de son système, Oracle propose le support des fonctions SQLX.

Différentes fonctions sont proposées, telles que :

- `XMLElement()` permet de définir une balise XML. On lui associe un nom, un éventuel attribut, et on décrit ce qui doit se trouver à l'intérieur de la balise. Il peut s'agir soit d'autres balises, soit de texte.
- `XMLConcat()` permet de concaténer tous les éléments passés en argument, afin de former un bloc XML.
- `XMLForest()` produit une forêt (suite de balises XML) à partir des éléments passés en argument.

– `XMLAgg()` est une fonction qui permet d'agréger des éléments entre eux. Elle peut être associée à une clause `ORDER BY` permettant de trier les résultats.

Dans le cadre de ce travail, nous avons utilisé cette solution. Elle permettait de travailler directement en ligne de commande, avec le système existant et fournissait toutes les fonctionnalités dont nous avions besoin pour notre fichier XML de sortie. De plus, il s'agit d'un standard émergent, et nous avons toujours tenté de travailler avec des normes standardisées dans le cadre de ce travail.

7.4.2 Génération du fichier de données XML

Nous avons pu acquérir, sur le site Eurostat, des données démographiques (nombre d'habitants) que nous avons introduites dans une table de notre système Oracle.

Nous avons ensuite effectué une requête sur cette table, en l'occurrence lister les données géographiques pour la Belgique (en ce compris ses trois régions) pour les années 2004 et 2005.

Voici cette requête :

```
select XMLELEMENT("file",XMLELEMENT("country",
       XMLATTRIBUTES(d.country as "id"),
       (select XMLAGG(XMLELEMENT("region",
               XMLATTRIBUTES(d.region as "id"),
               XMLAGG(XMLELEMENT("data",
               XMLATTRIBUTES(d.year as "year"), d.value))))
       from DATA d
       where d.country='belgique' and
       (d.year=2004 or d.year=2005)
       group by d.region)))
from DATA d
where d.country='belgique'
group by d.country
;
```

Cette requête est relativement aisée à comprendre. Retenons un point important : les fonctions SQLX travaillent au niveau de la ligne de résultat. Donc, chaque ligne de résultat est interprétée et insérée entre les balises XML désirées.

Or, nous avons vu dans la section 7.3.1 que nous souhaitions agréger les données relatives à un pays au sein d'une seule balise XML correspondant au pays concerné. Ceci afin de préserver la structure hiérarchique de nos données.

Nous utilisons donc pour cela la fonction `XMLAGG()`, associée à une clause `GROUP BY`. Ceci permet d'agréger toutes les données relatives à la Belgique, et de les regrouper au sein de notre balise `<country>`.

Nous procédons de la même manière au niveau inférieur (région). Nous voulons agréger au sein d'une seule balise `<region>` les données relatives à celle-ci. Etant donné que la clause `GROUP BY` se rapporte à une seule instruction `SELECT`, nous devons introduire une deuxième instruction `SELECT` pour effectuer ce groupement.

Le reste de la requête ne constitue que de la mise en forme proprement dite. Le code est relativement intuitif à ce sujet. Précisons que la fonction `XMLATTRIBUTES()` permet d'ajouter un attribut à la balise XML décrite dans la fonction `XMLELEMENT()` parente.

7.5 Préparation du GML Viewer

7.5.1 Création d'un modèle propre

Afin de permettre à nos données d'interagir avec les différentes modifications apportées au GML Viewer, nous avons créé un widget spécifique sur notre page. Ce widget est défini au sein d'un modèle à part entière.

Nous avons vu, en effet, que le GML Viewer était construit sur base de modèles et, au sein de ceux-ci, nous pouvons définir des widgets. Les widgets sont, dans le cas qui nous intéresse, liés à une feuille de style qui se charge de transformer les données au format XML en HTML. Or, le fichier de données XML source est propre à un modèle, et un modèle ne peut contenir qu'un seul fichier source XML. Nous avons donc créé un modèle à part entière, dérivé du modèle basique proposé par le programme. Ce modèle prend pour fichier source nos données extraites d'Oracle.

Afin de permettre une interaction avec la carte, qui fait partie d'un modèle différent, le logiciel propose une balise spécifique, `<targetModel>`. Cette balise, une fois définie, permet d'associer à tous les widgets du modèle, une variable `targetModel` faisant référence au modèle avec lequel nous souhaitons travailler. Ceci autorise l'accès aux fonctions définies pour le modèle cible, en l'occurrence le modèle `mainMap`.

Voici un extrait du fichier `config.xml` qui illustre le modèle ainsi créé.

```
<models>
  <OwsContext id="mainMap">
[...]
  </OwsContext>
  <Model id="dataRep">
    <defaultModelUrl>./arbre/data.xml</defaultModelUrl>
    <widgets>
      <Widget id="dataTree">
        <htmlTagId>data</htmlTagId>
        <stylesheet>./arbre/data.xsl</stylesheet>
        <targetModel>mainMap</targetModel>
```

```
        </Widget>
      </widgets>
    </Model>
</models>
```

Nous pouvons voir ici l'ajout de notre modèle et de notre widget au programme. Outre le modèle `<OwsContext>` dont nous avons par ailleurs parlé au chapitre précédent, nous avons ajouté un autre modèle, générique celui-ci, via la balise `<Model id="dataRep">`.

Cette balise permet de définir un modèle générique, ne comprenant pas de fonctions spécifiques associées. Seules les fonctions et variables globales de bases nécessaires propres à tous les modèles sont définies. Nous n'avions en effet pas besoin de fonctions supplémentaires, étant donné que nos données vont interagir avec la carte, qui fait partie d'un autre modèle.

Nous précisons ensuite l'adresse du fichier de données XML correspondant au modèle. Nous avons vu qu'à un modèle correspondait un et un seul fichier de données ; en l'occurrence, notre fichier de données et le fichier généré par Oracle lors de la requête effectuée et décrite plus haut. Ce fichier s'appelle `data.xml`.

Au sein de notre modèle, nous définissons notre widget. Il aura pour fonction d'afficher les données en HTML sur notre page, à l'aide d'une feuille de style XSLT. Cette feuille de style sera détaillée dans ce chapitre.

Autre point important, nous définissons, à l'aide de la balise `<targetModel>`, le modèle sur lequel nous souhaitons interagir. Nous avons dit précédemment que notre modèle était en fait un modèle de base. Or, pour modifer l'aspect de la carte à partir de notre modèle, nous allons avoir besoin de fonctions plus spécialisées que les fonctions définies pour tous les modèles.

Ceci est permis grâce à cette balise. Elle permet de définir une variable appartenant au modèle, celui que nous avons défini, appellée `targetModel`. A l'aide de cette variable, nous pouvons accéder, au sein de notre modèle, aux différentes fonctions définies pour le modèle cible.

Notre modèle et notre widget une fois définis dans le fichier de configuration XML, le programme, au chargement, va créer des instances de modèles et de widgets avec lesquelles nous pourrons interagir avec le programme.

7.5.2 Définition des couches en GML

Nous avions pour objectif de permettre une interaction entre les données extraites en XML et la carte au sein de notre programme. Nous voulions donc permettre, sur notre carte, l'affichage de zones géographiques que nous avons dû définir, ces zones s'affichant en surimpression sur la carte.

Pour ce qui est des zones géographiques, les couches sont écrites en GML. Un fichier dédié doit être codé pour décrire chaque couche. Ce fichier a une extension `.xml`, car les navigateurs internet ne peuvent charger que des pages

XML. Si nous avions spécifié une extension en `.gml`, la couche n'aurait pas pu être chargée. Cependant, ce n'est pas un problème car le code GML est avant tout du code XML avec des balises et une grammaire spécifique. Il s'agit donc aussi d'un fichier XML.

Afin de définir une couverture géographique, nous utilisons la propriété GML nommé `LinearRing`. Il s'agit d'un ensemble de segments contigus, dont les points sont renseignés par des coordonnées. Cet ensemble de segments commence et se termine en un point, formant un anneau (*ring*) décrivant la couverture.

Afin de représenter les zones concernées par nos données, à savoir la Belgique et ses trois régions, nous avons dû décrire les frontières de celles-ci à l'aide de coordonnées. Cependant, nous ne sommes pas parvenus à trouver des données comprenant les coordonnées GPS des frontières de la Belgique et de ses trois régions.

Par contre, nous avons pu trouver un site internet, Geoco[1], renseignant les coordonnées GPS de plus de 100.000 villes de douze pays européens, dont la Belgique. Afin de parvenir à une représentation réaliste de nos frontières, nous avons retenus les coordonnées de plus de 150 villes différentes, situées le long de la frontière extérieure ou le long de la frontière linguistique.

Quatre fichiers en GML ont ainsi été créés, correspondant au pays et à ses trois régions. Ensuite, nous avons inclus ces couches à notre programme en modifiant le fichier de contexte que nous avons détaillé dans le chapitre précédent.

Le code source apporte peu d'informations supplémentaires, il ne nous a pas semblé utile de le détailler ici.

Notons qu'en interne, dans le fichier de contexte, nous n'avons pas choisi le nom des couches par hasard. Ceci est un point important : le nom interne de la couche doit correspondre à l'identifiant de l'entité géographique concernée dans notre fichier XML de données.

Ceci permet d'éviter de devoir établir une quelconque correspondance entre le fichier de données et les différentes couches de notre programme. Cela ne représente pas un problème. En effet, lors de la mise en place d'une telle solution, les concepteurs peuvent également faire correspondre les noms des couches qu'ils vont définir aux formalismes utilisés dans la base de données.

7.6 Transformation du fichier source en HTML

Nous l'avons vu, le GML Viewer fonctionne à partir de fichiers de données/configurations XML, exception faite de la description des différentes couches de la carte, quant à elles écrites en GML. Ces fichiers XML sont interprétés par une feuille de style XSLT et affichés entre les balises appropriées au sein de la page HTML principale, dont nous prenons le soin de concevoir l'affichage.

[1] `http://www.geoco.org`

Afin d'afficher les données XML au sein du programme, et de rendre possible l'interaction avec celui-ci, nous avons procédé de la même manière. Nous avons écrit une feuille de style XSLT, capable de transformer notre fichier XML de données, en un fichier HTML dont la plupart des balises comportent des attributs nous permettant de faire interagir le code HTML généré avec le programme.

Notre feuille de style s'emploie donc à transformer le fichier de données que nous avons obtenu grâce à la base de données oracle en HTML. Elle est particulièrement importante, car elle crée l'interaction proprement dite entre le programme GML Viewer et nos données.

Il a donc été conçu un système de balises possédant presque toutes un attribut. Ces attributs sont de différents types et correspondent chaque fois à une fonction bien précise.

A l'aide des fonctions DOM du langage javascript, nous avons pu nous "promener" dans le document HTML ainsi créé, et permettre alors l'interaction souhaitée.

Nous allons à présent illustrer quelques morceaux de code, particulièrement révélateurs. En parallèle, nous montrerons quelques passages du code HTML ainsi généré, et nous pourrons alors comprendre plus aisément les différents choix effectués.

Tout d'abord, illustrons les variables permettant d'appeler les fonctions définies dans le programme, et ce qu'elles soient déjà existantes ou créées par nos soins.

```
<xsl:param name="targetModel"/>
<xsl:param name="target">config.objects.<xsl:value-of
        select="$targetModel"/></xsl:param>
```

Nous avons vu dans la section précédente la définition d'une variable `targetModel` qui nous permettait d'avoir accès aux fonctions définies pour l'objet carte de la page internet. Ces fonctions doivent être appelées depuis la page principale, et nous devons, dans notre feuille de style, avoir une variable contenant cette valeur.

Le GML Viewer permet ceci. Un certain nombre de variables sont pré-définies pour chaque widget et nous les spécifions dans le fichier de configuration, ce que nous avons fait avec `targetModel`. A la création d'une instance du widget, les différences variables définies dans le fichier de configuration sont assignées dans la classe du widget proprement dite, mais aussi dans la feuille de style correspondante. Cette feuille de style est soit renseignée par l'utilisateur, soit définie par défaut, et se trouve alors à l'endroit `./lib/widget/nom_du_widget.xsl`.

A l'aide de cette variable, nous pouvons maintenant accéder aux fonctions définies pour la carte, via le HTML que nous voulons générer.

Illustrons maintenant le début de notre feuille de style.

```
<xsl:template match="/file">
  <div><h1>[Données]</h1><br/>
  <xsl:apply-templates/>
  </div>
</xsl:template>
```

Voici le début du HTML que nous générons à partir de notre fichier de données XML. Nous produisons un titre : “Données”. Les différents balises HTML qui sont ici employées ont des propriétés de style qui leurs sont associées. Ces propriétés sont définies dans un fichier CSS séparé, que nous avons écrit. Il ne nous a pas paru utile de le détailler ici.

Via l'instruction `<xsl:apply-templates/>`, nous spécifions qu'à cet endroit, nous souhaitons exécuter les transformations décrites pour les noeuds enfants du noeud courant. Nous pouvons observer que tout le code HTML que nous produisons est compris au sein d'une balise `<div>`. En effet, le programme n'insère qu'une seule balise racine à l'intérieur de la balise prévue dans la page HTML principale. Nous devons donc “entourer” (*wrapper*) le code que nous souhaitons produire d'une balise, et la balise `<div>` est idéale pour cette fonction.

```
<xsl:template match="country">
  <div class="country">
  <xsl:attribute name="id">
    <xsl:value-of select="@id"/>
  </xsl:attribute>

    <span>
      <xsl:attribute name="onclick">
        <xsl:value-of select="$target"/>.switchLayerDisplay
        ('<xsl:value-of select="@id"/>', true);
      </xsl:attribute>
      <xsl:value-of select="@id"/>
    </span>

    <div class="region">
      <xsl:apply-templates/>
    </div>
  </div>
</xsl:template>
```

Commentons ce morceau de code. Nous avons vu, dans notre fichier de données XML produit, que nous avions plusieurs niveaux de granularité[2] disponibles. La feuille de style XSLT que nous avons produite traite chaque niveau de granularité de la même manière. Nous illustrons ici la manière dont sont traitées

[2] voir chapitre 2, consacré aux entrepôts de données

les balises `<country>`, mais le procédé est similaire pour les balises des niveaux de granularité inférieurs (`<region>`, `<data>`), ou supérieurs (par exemple `<continent>`, `<zone>`, ... si nous les avions définis).

Pour rappel, ce code est destiné à produire une vision des données sous forme d'arbre, en HTML, à l'aide de feuilles de style CSS et de fonctions javascript.

Chaque niveau de granularité en XML est donc représenté par une balise `<div>`, à laquelle on associe un attribut `class`. Ce dernier fait référence aux propriétés de style définies dans le fichier de style externe. Outre les différentes modifications classiques (`font`, `font-weight`, `font-color`, ...), nous avons associé, à chaque attribut `class`, une propriété : `margin-left`. Ceci constitue un décalage par rapport à la marge gauche de la page, ou de la balise `<div>` courante, et simule un niveau inférieur dans l'arbre représentant les données.

C'est ainsi que, par exemple, la balise `<div class="region">`, et surtout le texte qu'elle contient, seront décalés par rapport à la balise `<div class="country">` car, dans le fichier CSS distant, nous avons :

```
.region{
          margin-left:  40px;
          cursor:  pointer;
          display:  none;
}
```

La balise `<div class="region">` étant incluse dans la balise `<div class="country">`, elle sera décalée de quarante pixels à gauche par rapport à cette dernière. Les balises incluses dans la région seront, elles aussi, décalées de quarante pixels et ainsi de suite, créant par conséquent la structure d'arbre qui, à notre sens, reflète de la manière la plus intuitive la relation entre les différentes données.

En effet, il nous a paru plus sensé de procéder de cette manière. Nous avions d'abord envisagé un affichage classique sous forme de tableau, mais cet affichage sous forme d'arbre est bien plus indiqué dans ce cas. Il procure un niveau d'abstraction supplémentaire dans la visualisation des données. Nous n'avons en effet pas besoin de préciser par exemple que les entités "wallonie", "bruxelles", et "flandre" appartiennent à "belgique" via une colonne supplémentaire, la seule représentation sous forme d'arbre suffit à le comprendre.

Ceci est le but vers lequel nous voulons tendre. Les solutions SOLAP tirent leur atouts de ces niveaux d'abstraction d'informations rendant plus claire la visualisation des données proprement dites.

Dans cet extrait de code CSS, l'attribut `display` permet de ne pas afficher les éléments non-racine de l'arbre. Cette propriété sera modifiée grâce aux événements effectués sur les éléments de l'arbre. Un clic de souris sur le parent, provoquera la modification de la propriété `display` de ses enfants.

Retournons au code de la page 77, outre l'attribut de classe, nous pouvons voir

que les balises `<div>` possèdent aussi un deuxième attribut, nommé `id`. Il nous est nécessaire pour les fonctions javascript que nous allons décrire dans la suite. En effet, à l'aide des fonctions DOM présentes dans javascript, nous sommes capables d'identifier un noeud (balise, texte) d'un document HTML en fonction de la valeur de son attribut `id`.

Au sein de la balise, nous allons produire le texte correspondant afin qu'il s'affiche sur la page internet. Ce texte sera inséré dans des balises `<span>`, qui ont pour but d'associer des attributs, souvent des propriétés de style, à une partie de texte, indépendamment de la balise au sein de laquelle elle se trouve.

Ici, nous désirons simplement produire un évènement javascript. Nous apportons, via l'attribut `onclick`, la possibilité d'appeler une fonction si on clique sur le texte considéré. Afin de rendre la chose encore plus intuitive, nous avons redéfini dans notre feuille de style CSS le curseur à afficher au survol de ce texte. Il s'agit du curseur `pointer` indiquant que le texte est cliquable.

Au moment d'afficher le texte, la stratégie diffère quelque peu selon le cas considéré. Si nous affichons les données proprement dites, donc les feuilles dans notre arbre de représentation, il suffit d'afficher au sein des balises `<span>` les feuilles du fichier XML, via l'instruction `<xsl:apply-templates/>`. En effet, si nous regardons le fichier XML de données sur lequel nous travaillons, les données sont exprimées sous forme de texte au sein des balises `<data>` que nous sommes en train de traiter, et elles n'ont pas de fils dans le document.

Si, au contraire, nous nous situons plus haut dans la hiérarchie et donc dans l'arbre, alors nous sommes forcément en train de traiter des informations de lieu (pays, région, ...). Dans ce cas, le texte à afficher est, comme dans notre exemple, la valeur de l'attribut `id` de la balise `<country>` de notre fichier XML de données.

Enfin, au sein de notre balise principale, pour rappel `<div id="country">`, nous introduisons déjà la balise du niveau précédent, `<div class="region">`.

Ceci a été introduit dans le but d'améliorer quelque peu les performances. En effet, le souci principal est le parcours du fichier XML, dans le cas où nous travaillons avec de grosses quantités de données. Cette balise que nous ajoutons, va contenir tous les fils de l'élément que nous sommes en train de traiter. Ces fils seront donc tous inclus dans la même balise, possédant un attribut de classe. Si nous cliquons sur l'élément courant, il nous suffit donc de provoquer l'affichage de la balise rajoutée, plutôt que de parcourir tous les fils afin de modifier leur propriété d'affichage.

A titre d'exemple, voici un extrait du code HTML ainsi produit, et interprété par notre navigateur internet :

```
<div class="country" id="belgium">
  <span onclick="config.objects.mainMap.
   switchLayerDisplay('belgium', true);">
    belgium
  </span>
```

```
  <div class="region">
[...]
     <div id="wallonie">
       <span onclick="config.objects.mainMap.
        switchLayerDisplay('wallonie', false);">wallonie</span>
       <div class="data">
         <div><span>2004: 2000</span></div>
         <div><span>2005: 2000</span></div>
       </div>
     </div>
[...]
```

7.7 Interaction avec le GML Viewer

Dans la section précédente, nous avons vu que des événements javascript `onclick` ont été insérés dans le HTML généré. Ces événements appellent une fonction, `switchLayerDisplay`, chargée à la fois de gérer l'affichage/masquage des couches sur la carte, mais aussi de gérer l'affichage de notre arbre de représentation de données.

Nous avons choisi de regrouper ces deux tâches en une seule fonction afin de minimiser les parcours de fichier. En effet, nous devons dans les deux cas parcourir le fichier HTML et accéder aux mêmes informations. En regroupant les tâches, nous ne faisons plus qu'un seul parcours. Nous allons illustrer ici cette fonction.

L'idée est simple, nous devons faire appel à cette fonction lorsque nous voulons basculer l'affichage des couches et déplier/replier l'arbre. Ces deux actions vont de pair. En effet, l'affichage d'une couche sur la carte correspond au déploiement des fils du noeud correspondant dans l'arbre de même que l'action de cliquer une seconde fois sur un noeud de l'arbre provoque la suppression de l'affichage de ses fils (au sens large) et de la (des) couche(s) concernée(s).

Tout d'abord, remarquons que nous avons ajouté cette fonction dans la classe de notre modèle cible[3], et non de notre propre modèle.

La raison en est simple : cette fonction a pour but d'afficher/masquer des couches sur la carte et cette dernière appartient au modèle cible. De plus, les interactions souhaitées avec nos données (le déploiement de l'arbre) se feront non pas sur notre fichier de données XML, mais sur le fichier HTML de la page principale de notre programme.

Il est important de garder ceci à l'esprit. Nous travaillons ici sur une fonction à laquelle on accède à partir du code HTML généré. Donc, les données sont accé-

[3]Pour rappel, nous avons travaillé à partir de notre modèle, interagissant avec un modèle cible, défini dans le fichier de configuration par la balise `<targetModel>`.

dées via les différentes fonctions DOM du javascript qui parcourent le document HTML.

```
this.switchLayerDisplay=function(layerName, parent){
  var tag = document.getElementById(layerName);
  var children=tag.childNodes;
```

Voici le début de notre fonction. Elle nécessite deux paramètres : le nom de la couche que nous désirons afficher/cacher et un booléen, nommé `parent` dont nous allons parler très prochainement. Elle illustre également un point important de notre manière de travailler : notre recours aux fonctions DOM accessibles dans javascript.

Dans ce cas précis, nous utilisons la fonction `getElementById`, qui permet d'identifier un noeud du document HTML d'après son attribut `id`. Nous l'avons dit plus haut, c'est la raison pour laquelle nous avons associé des attributs `id` à certaines de nos balises `<div>`.

Si nous nous reportons à l'extrait de code HTML généré pour notre fichier XSLT, nous pouvons voir quel noeud sera assigné dans la variable `tag` ; il s'agit, par exemple, du noeud `<div id="wallonie">`.

Ensuite, nous pouvons utiliser, depuis ce noeud, plusieurs fonctions DOM de javascript. En l'occurrence, nous avons maintenant en mémoire le noeud racine d'une région dans notre document. Nous allons, depuis la variable `children`, accéder à tous ses fils. La variable `children` est donc un vecteur contenant tous les fils du noeud `tag` dans l'ordre dans lequel ils sont rencontrés dans le document. Voyons la suite :

```
if(children[1].style.display=="block"){
  if(layerName!="total")
    this.setHidden(layerName, true);
  children[1].style.display="none";
  if(parent)
    this.clearChildren(children[1]);
  }
  else{
    if(layerName!="total")
      this.setHidden(layerName,false);
    children[1].style.display="block";
  }
}
```

Afin de bien comprendre, plaçons nous dans un contexte. Cette fonction est appelée, par exemple, avec les paramètres `'belgique'` et `'true'`. Au premier clic, correspondant à un affichage d'informations, elle doit donc faire deux choses :

– afficher la couche correspondant à la Belgique,
– afficher les fils directs dans l'arbre.

Ceci est fait dans le corps de l'instruction `else`. Nous provoquons l'affichage de la couche passée en paramètre, si elle ne s'appelle pas "total". En effet, chaque niveau de granularité (sauf le dernier, constitué des données) comporte une entité "total". Dans notre exemple, il s'agit du total pour la Belgique. Si nous avions eu des données par province, un total aurait été spécifié dans chacune des régions comprenant les totaux de toutes les provinces de cette region, ... La pseudo-couche "total" ne correspond donc pas à une zone à afficher ici.

Ensuite, nous voulons provoquer l'affichage des fils de cet élément dans l'arbre. Nous avons dit dans le détail de notre conception que nous avions regroupé tous les fils d'un élément au sein d'une balise `<div>` afin d'en faciliter l'affichage. Cette balise est **toujours** le deuxième fils du noeud courant dans le document[4], accessible via `children[1]`. Pour l'affichage proprement dit, nous jouons simplement sur la propriété `display` de la feuille de style CSS associée : `"block"` affiche l'information alors que `"none"` ne l'affiche pas.

Lorsque que nous cliquons pour la deuxième fois sur un élément de notre arbre, ou, plus généralement, à chaque clic pair, nous souhaitons masquer l'affichage de la couche, mais aussi l'affichage de tous les fils (au sens large).

En effet, il nous a paru plus intuitif de procéder ainsi. Si cinq niveaux de l'arbre sont dépliés, mais que l'on désire replier le deuxième, alors tous les niveaux inférieurs le seront également. Ceci est fait grâce à la fonction `clearChildren`. Cette fonction est appelée seulement si le booléen `parent` est vrai. En effet, si l'élément que nous traitons ne possède que des fils qui sont des feuilles, nous n'avons pas besoin de descendre dans la hiérarchie afin de modifier les propriétés des éléments présents.

La fonction `clearChildren` est donc une fonction récursive qui parcourt le sous-arbre à partir d'un noeud et supprime, d'une part, l'affichage des éléments dans l'arbre et, d'autre part, l'affichage des couches GML correspondantes.

Cette solution pose évidemment des problèmes en cas de gros fichiers de données, mais nous travaillons depuis le début avec des fichiers textes, et nous avons, dans le cadre de ce travail, voulu explorer cette piste. Nous discuterons des quelques limitations que cela engendre dans le chapitre suivant.

Voici une capture d'écran du résultat obtenu, visionnée avec le navigateur Firefox (figure 7.1).

[4]Il peut s'avérer utile de se reporter au code HTML généré par notre feuille de style à la page 80.

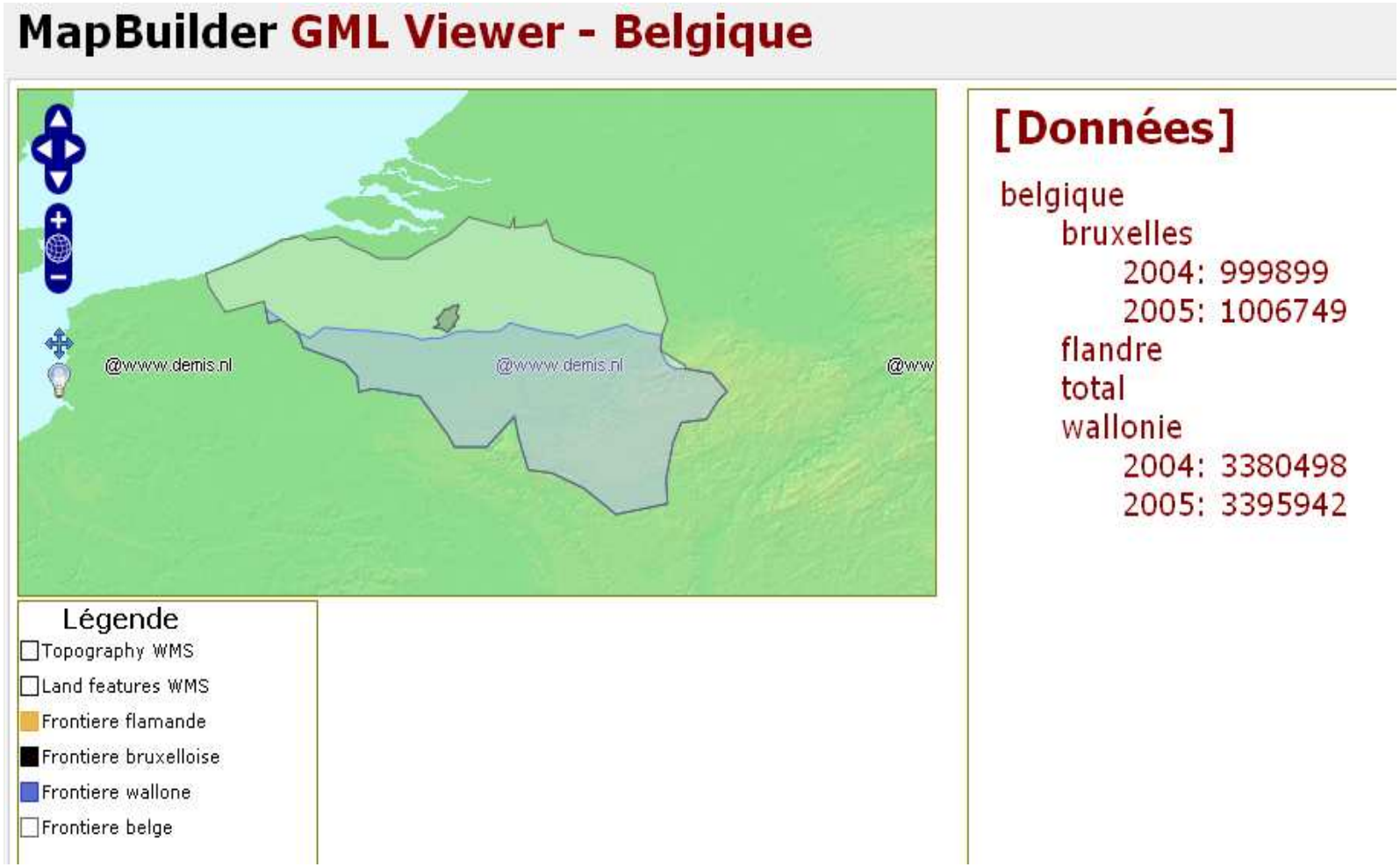

FIG. 7.1 – Capture d'écran du résultat dans GML Viewer

7.8 Limites

Gardons à l'esprit que nous avons choisi de travailler avec des fichiers de données textuels. Nous avons, à plusieurs reprises, précisé que leur principal défaut consistait en leur lenteur par rapport aux fichiers binaires.

Nous sommes donc confrontés à des limitations de taille de fichier en travaillant comme nous l'avons fait, ce dont nous étions conscients depuis le début.

Afin de ne pas avoir à définir des fichiers GML trop conséquents, nous avons ici travaillé avec la Belgique et ses trois régions uniquement. Ceci ne peut constituer un gros fichier de données.

Nous avons donc voulu tester notre programme avec des fichiers plus conséquents. Nous nous sommes pour cela aidés une nouvelle fois des schémas XML. En effet, nous avons écrit un fichier `.xsd` (fichier XML Schema) validant notre fichier de données généré par Oracle. Armés de ce schéma, nous avons pu générer automatiquement des fichiers XML lui correspondant.

Evidemment, seules les balises étaient correctes, les attributs ne correspondaient plus à des couches présentes sur notre carte. Ceci ne pose aucun problème, car nous parcourons le fichier autant pour l'affichage de données au sein de la page internet que pour l'affichage des couches sur la carte. En fait, l'affichage/masquage des couches n'est constitué que d'un appel de fonction s'exécutant en un temps constant.

Nous avons donc créé une fonction de test chargée de déployer entièrement l'arbre de données sur la page principale du programme. Ensuite, nous avons cliqué sur la racine de l'arbre afin de replier toute la structure. Nous avions préalablement pris le soin de définir des variables permettant de calculer le temps d'exécution de notre fonction.

Nos fichiers XML comprenaient de 10 à 10.000 régions au sein du pays et chacune de ces régions contient les données de dix années. Notre fonction s'exécute assez rapidement, même avec le plus gros fichier. Il a suffit de 3,303 secondes pour tout déplier, et 4,004 secondes pour tout replier[5].

Par contre, le navigateur utilisé, Firefox, peinait à afficher les données. Il nous est apparu qu'en fait, ce qui freinait le plus le navigateur était le processeur XSLT interne, lors de sa transcription du XML vers l'HTML.

La réactivité reste très correcte avec 1.000 régions, soit à peu près 10.000 lignes dans le fichier XML. Par contre, si nous multiplions par dix la taille du fichier, soit 10.000 régions, notre navigateur pouvait mettre jusqu'à quinze secondes pour répondre à l'événement généré[6].

Ceci ne nous étonne pas outre mesure. Nous devons garder à l'esprit que nous manipulons des fichiers textuels tout au long de notre processus de transformation. Nous pouvons donc établir que des solutions de ce type ne peuvent

[5] Temps de réponse le plus long recensé

[6] Test réalisé sur un processeur PPC G4 @1.42GHz, 1Gb RAM.

en aucun cas intéresser des compagnies ayant de gros volumes de données à analyser.

Cependant, de part sa simplicité, sa gratuité, et sa portabilité, ces systèmes restent intéressants à bien des égards. Il convient cependant de rester lucide quant aux limitations inhérentes à la méthode utilisée.

7.9 Conclusions

Dans ce chapitre, nous avons décrit la solution proposée dans le cadre de ce mémoire afin de pouvoir visionner des résultats de requêtes depuis Oracle dans un logiciel offrant un support cartographique.

Dans le chapitre suivant, nous allons conclure notre travail, et énoncer les limites de celui-ci. De plus, nous discuterons des apports indispensables si nous voulons voir se propager ce type de techniques.

Chapitre 8

Conclusions et développements futurs

8.1 Conclusions

Les entrepôts de données sont devenus aujourd'hui un incontournable dans le domaine de la *Business Intelligence*. Couplés avec les possibilité des techniques OLAP, ils permettent une étude analytique poussée des données, procurant une efficace aide à la décision dans tous les domaines.

Une autre catégorie d'outils, dérivée des premiers, est également en plein essor. Il s'agit des outils SOLAP, qui allient la puissance analytique des outils OLAP, avec les capacités de visualisation des cartes géographiques.

Afin de développer un embryon de solution SOLAP, but final de ce mémoire, nous avons dû nous familiariser avec les différents éléments qui composent généralement ces systèmes.

Nous avons donc présenté les entrepôts de données et illustré, à l'aide d'exemples, les différentes possibilités qu'ils offrent. Ensuite, nous avons décrit les systèmes dits SOLAP, et avons présenté pour cela les systèmes GIS et OLAP, dont ils sont issus.

Dans les chapitres suivants, nous avons étudié les langages nécessaires dans le cadre de la réalisation pratique de ce mémoire. Nous avons en effet eu à travailler avec les langages XML, GML, XSLT, et plus généralement, nous avons dû nous familiariser avec les techniques de développement dites "Ajax".

Nous avons ensuite présenté le logiciel sur lequel nous nous sommes basés pour proposer une solution de visualisation des données se rapprochant des techniques SOLAP. Enfin, nous avons décrit les différentes étapes qui ont abouti à la réalisation de notre programme.

En conclusion, il convient de garder à l'esprit que cette solution ne peut s'adapter à de gros volumes de données. Nous avons vu que, travaillant avec des fichiers textuels, nous étions limités en termes de quantité de données. Cependant, ces

méthodes présentent un certain nombre d'avantages non négligeables.

Tout au long de ce travail, nous avons pu nous passer de tous logiciels, langages, librairies propriétaires. De plus, nous ne nous sommes pas rendus dépendants d'une plate-forme de développement particulière.

Il s'agit donc d'une solution entièrement gratuite, ne nécessitant aucun investissement, et relativement facile à mettre au point de part les technologies utilisées. Afin d'arriver à une solution SOLAP complète, il convient de continuer le développement de ce programme, afin d'offrir une meilleure visualisation des données à des fins d'analyses.

8.2 Développements futurs

Le développement de tels programmes paraît plus que jamais une piste intéressante au vu des nombreuses applications possibles. Pour ce faire, plusieurs aspects sont à développer.

Premièrement, les données pourraient être représentées plus efficacement. Ici, nous nous sommes attelés à afficher une représentation de la zone géographique concernée, ce qui constitue un premier pas. Il pourrait être avantageux, en complément, d'offrir des outils analytiques représentant ces mêmes données. Par exemple, il serait utile d'afficher divers graphiques (camemberts, diagrammes, ...) sur cette même page internet.

Il serait également intéressant de développer une personnalisation de l'interface, avec diverses représentations des données, divers agencements, une personnalisation poussée au niveau des couleurs, ... En effet, il s'agit là d'un aspect à prévoir lors de la conception de systèmes SOLAP, si nous nous reportons aux premiers articles les décrivant.

Enfin, le GML Viewer permet pour le moment d'afficher des couches GML sur une carte au sein d'un navigateur. Or, le GML est conçu pour décrire des entités géographiques, et ses possibilités au niveau graphique sont très limitées. Il nous est apparu contre nature de tenter de représenter d'une quelconque manière des données en GML (à l'aide de cercles, de lignes, ...). De plus, le programme n'interprète pour le moment qu'une infime partie du langage GML, pour laquelle une conversion est prévue.

Il serait donc utile de développer cet aspect du programme. A savoir, permettre l'affichage de diverses représentations des données directement sur la carte. La littérature sur le sujet est loin d'être abondante, et les différentes techniques efficaces que nous avons pu recenser sont toutes propriétaires.

A ce titre, le langage SVG pourrait constituer une piste de travail. Ce langage, écrit lui aussi en GML, est encore une spécification du W3C. Il peut être visualisé nativement par la plupart des navigateurs. Il permet de décrire, à l'aide de formalismes XML, diverses sortes d'objets géométriques. Il est également possible d'afficher du texte, de définir certaines propriétés de style (couleur), ... En fait, le GML Viewer utilisé se sert de ce langage pour

transformer certaines balises GML, mais il ne permet pas de charger directement un fichier SVG.

Nous restons convaincus que ce type de logiciels a un avenir prometteur. Pour les raisons précitées, mais aussi parce qu'il s'agit de techniques pouvant s'adapter à toutes les situations. Nous pouvons donc arriver à un programme de stature professionnelle, à moindre coût. Ceci pourrait intéresser un ensemble de compagnies ou organisations aux besoins relativement modestes et qui pourraient tirer un réel avantage en termes d'aide à la décision. Ces mêmes compagnies n'auraient peut-être jamais considéré un investissement dans de coûteuses techniques propriétaires, et peuvent ici trouver une solution pouvant s'adapter fidèlement à leurs besoins.

Table des figures

2.1 Représentation cubique des données 8
2.2 Architecture classique des entrepôts de données 9
2.3 Opération *slice* 11
2.4 Opération *dice* 12
2.5 Opération *pivot* 12
2.6 Opération *roll-up* 12
2.7 Opération *drill-down* 13

3.1 Couverture des succursales de la Bank of America dans la ville de New York. 17
3.2 Architecture d'un système GIS. 18

4.1 Aperçu de la transformation XSLT dans Firefox 38

6.1 Capture d'écran de GML Viewer 49
6.2 Fichiers principaux de GML Viewer 52

7.1 Capture d'écran du résultat dans GML Viewer 83

Bibliographie

[1] E. Turban and J. Aronson. *Decision support systems and intelligent systems*. Prentice-Hall, 2001.

[2] D. J. Power. *Decision support systems : concepts and ressources for managers*. Quorum Books, 2002.

[3] T. B. Pedersen and C. S. Jensen. Multidimensional database technology. *IEEE Computer society*, pages 40 – 46, décembre 2001.

[4] W. H. Inmon and R. D. Hackathorn. *Using the Data Warehouse*. Wiley, 1994.

[5] R. Kimball and M. Ross. *The Data Warehouse Toolkit : the Complete Guide to Dimensional Modeling*. John Wiley & Sons, seconde edition, 2002.

[6] A. Shukla, J. F. Naughton, P. M. Deshpande, and K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 522 – 531. Morgan Kaufmann, 1996.

[7] J. Kiviniemi, A. Wolski, A. Pesonen, and J. Arminen. Lazy aggregates for real-time OLAP. In *Data Warehousing and Knowledge Discovery*, pages 165–172, 1999.

[8] N. Pasquier. OLAP and data warehousing. Technical report, Université de Nice, 2006.

[9] M. Worboys and M. Duckham. *GIS, A Computing Perspective*. CRC Press, 2004.

[10] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP to user-analysts : An IT mandate. Technical report, Hyperion Solutions Corporation, 1993.

[11] Y. Bedard, S. Rivest, and M.-J. Proulx. Spatial on-line analytical processing (SOLAP) : Concepts, architectures and solutions from a geomatics engineering perspective. In *Data Warehouses and OLAP : Concepts, Architectures and solutions*, chapter 13, pages 298 – 319. Robert Wrembel & Christian Koncilia (ed(s)), 2007.

[12] S. Rivest, Y. Bedard, and P. Marchand. Toward better support for spatial decision making : Defining the characteristics of spatial on-line analytical processing. *Geomatica*, 55(4) :539 – 555, 2001.

[13] Y. Bedard, M. Proulx, S. Rivest, and T. Badard. Data warehouses and OLAP : Concepts, architectures and solutions. In *Geographic Hypermedia : Concepts and Systems*, pages 167 – 185. E.Stefanakis, M.P. Peterson, C. Armenakis, V. Deli (ed(s)), 2006.

[14] T. Bray, J. Paoli, C. M. Sperberg McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0. W3C recommandation, 2006.

[15] D. Murray. GML — user perspectives. In *Map Asia 2004*, 2004.

[16] P. H. Dana. The mapinfo interchange file format specification. Technical report, MapInfo Corporation, 1999.

[17] S. Cox, A. Cuthbert, R. Lake, and R. Martell. Geography markup language (GML) 2.0. Technical report, Open Geospatial Consortium, 2001.

[18] C. Vangenot. Datawarehouse. Technical report, Ecole polytechnique fédérale de Lausanne, 2003.

[19] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1) :65 – 74, 1997.

[20] P. A. Longley, M. F. Goodchild, D. J. Maguire, D. W. Rhind, and D. J. Maguire. *Geographic Information Systems and Science*. Wiley, 2001.

[21] S. Chaudhuri, U. Dayal, and V. Ganti. Database technology for decision support systems. *Computer*, 34(12) :48 – 55, 2001.

[22] J. Trujillo, M. Palomar, J. Gomez, and I.-Y. Song. Designing data warehouses with OO conceptual models. *Computer*, 34(12) :66 – 75, 2001.

[23] R. Mc Hugh, F. Bilodeau, S. Rivest, and Y. Bédard. Analyse du potentiel d'une application SOLAP pour une gestion efficace de l'erosion des berges en gaspesie illes-de-la-madeleine. In *Geomatique 2006*, Montréal, Canada, 25-26 Octobre 2006.

[24] S. Cox, P. Daisey, R. Lake, C. Portele, and A. Whiteside. Geography markup language (GML) 3.1. Committee draft, Open Geospatial Consortium, 2004.

[25] D. Cabasson. *Ajax, une auto-complétion pas à pas*, 2006.

[26] J. De La Beaujardiere and A. Doyle. Web map service implementation specification. Technical report, Open Geospatial Consortium, 2002.

[27] D. S. Burggraf. Geography markup language. *Data Science*, 5 :178–204, Octobre 2006.

[28] F. Depasse. Conception sous eclipse d'un visualisateur de données spatio-temporelles, 2005.

Annexe A

Liste des acronymes rencontrés

AJAX	*Asynchronous Javascript And XML*
CLOB	*Character Large Object*
DOM	*Document Object Model*
DSS	*Decision Support System*
DTD	*Document Type Definition*
EPSG	*European Petroleum Survey Group*
GIS	*Geographic Information System*
GML	*Generalized Markup Language*
	Geography Markup Language
GPL	*General Public License*
HOLAP	*Hybrid On-Line Analytical Processing*
HTML	*HyperText Markup Language*
ISO	*International Organization for Standardization*
LGPL	*Lesser General Public License*
MOLAP	*Multidimensional On-Line Analytical Processing*
OGC	*Open Geospatial Consortium*
	Open GIS Consortium
OLAP	*On-Line Analytical Processing*
OLTP	*On-Line Transaction Processing*
OWS	*OGC Web Service*
RDBMS	*Relational DataBase Management System*
RDF/S	*Resource Description Framework Schema*
ROLAP	*Relational On-Line Analytical Processing*
RTOLAP	*Real-Time On-Line Analytical Processing*
SAX	*Simple API for XML*
SDSS	*Spatial Decision Support System*
SGML	*Standard Generalized Markup Language*
SLD	*Styled Layer Descriptor*
SOLAP	*Spatial On-Line Analytical Processing*
SOX	*Schema for Object-oriented XML*

SQL	*Structured Query Language*
StAX	*Streaming API for XML*
SVG	*Scalable Vector Graphics*
URL	*Uniform Resource Location*
W3C	*World Wide Web Consortium*
WMS	*Web Map Service*
WOLAP	*Web-based On-Line Analytical Processing*
XDR	*XML Data Reduced*
XML	*eXtensible Markup Language*
XSLT	*eXtensible Stylesheet Language Transformation*
XSL-FO	*eXtensible Stylesheet Language - Formatting Objects*

Annexe B

Exemples du chapitre 4

B.1 Les DTD's

Code source complet du schéma de validation DTD de l'exemple XML du chapitre éponyme.

```
<!ELEMENT annuaire (contact+)>
<!ELEMENT contact (nom, prenom, adresse, tel)>
<!ATTLIST contact id ID #REQUIRED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT adresse (pays, province, ville, zip, rue, num)>
<!ELEMENT pays (#PCDATA)>
<!ELEMENT province (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT rue (#PCDATA)>
<!ELEMENT num (#PCDATA)>
<!ELEMENT tel (#PCDATA)>
```

B.2 XML Schema

Code source complet du schéma de validation en XML Schema du même exemple XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="nom" type="xsd:string"/>
<xsd:element name="prenom" type="xsd:string"/>
<xsd:element name="pays" type="xsd:string"/>
<xsd:element name="province" type="xsd:string"/>
<xsd:element name="ville" type="xsd:string"/>
<xsd:element name="rue" type="xsd:string"/>
<xsd:element name="num" type="xsd:int"/>

<xsd:element name="tel">
<xsd:simpleType>
     <xsd:restriction base="xsd:string">
          <xsd:pattern value="[0-9]{3}-[0-9]{3}-[0-9]{4}"/>
     </xsd:restriction>
</xsd:simpleType>
</xsd:element>

<xsd:element name="zip">
<xsd:simpleType>
     <xsd:restriction base="xsd:string">
          <xsd:pattern value="[A-Z][0-9][A-Z][0-9][A-Z][0-9]"/>
     </xsd:restriction>
</xsd:simpleType>
</xsd:element>

<xsd:element name="adresse">
<xsd:complexType>
     <xsd:sequence>
          <xsd:element ref="pays"/>
          <xsd:element ref="province"/>
          <xsd:element ref="ville"/>
          <xsd:element ref="zip"/>
          <xsd:element ref="rue"/>
          <xsd:element ref="num"/>
     </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="contact">
```

```
<xsd:complexType>
     <xsd:sequence>
          <xsd:element ref="nom"/>
          <xsd:element ref="prenom"/>
          <xsd:element ref="adresse"/>
          <xsd:element ref="tel"/>
     </xsd:sequence>
     <xsd:attribute name="id" type="xsd:string" use="required"/>
</xsd:complexType>
</xsd:element>

<xsd:element name="annuaire">
<xsd:complexType>
     <xsd:sequence>
          <xsd:element ref="contact" maxOccurs="unbounded"/>
     </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

B.3 Feuille de style XSLT

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/
   1999/XSL/Transform">

<xsl:template match="/annuaire">
   <html><head><title>Exemple de tranformation</title>
   </head>
   <body>
   <h1>Mon annuaire</h1>
   <table border="1">
   <xsl:apply-templates/>
   </table>
   </body>
   </html>
</xsl:template>

<xsl:template match="contact">
   <tr>
   <td><xsl:value-of select="nom"/></td>
   <td><xsl:value-of select="prenom"/></td>
   <td><xsl:value-of select="adresse/num"/>
   <xsl:value-of select="adresse/rue"/><br/>
   <xsl:value-of select="adresse/zip"/> 
   <xsl:value-of select="adresse/ville"/></td>
   <td><xsl:value-of select="tel"/></td>
   </tr>
   </xsl:template>
</xsl:stylesheet>
```

Annexe C

Mise en oeuvre

C.1 Feuille de style XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
   xmlns:wmc="http://www.opengis.net/context"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">

<xsl:output method="xml" encoding="utf-8"/>

<xsl:param name="lang">en</xsl:param>
<xsl:param name="modelId"/>
<xsl:param name="widgetId"/>

<xsl:param name="context">config.objects.<xsl:value-of
     select="$modelId"/>
</xsl:param>
<xsl:param name="targetModel"/>
<xsl:param name="target">config.objects.<xsl:value-of
     select="$targetModel"/>
</xsl:param>


<xsl:template match="/file">
   <div><h1>[Données]</h1><br/>
   <xsl:apply-templates/>
   </div>
</xsl:template>

<xsl:template match="country">
   <div class="country">
   <xsl:attribute name="id">
```

```
      <xsl:value-of select="@id"/>
    </xsl:attribute>
    <span>
    <xsl:attribute name="onclick">
      <xsl:value-of select="$target"/>.switchLayerDisplay
        ('<xsl:value-of select="@id"/>', true);
    </xsl:attribute>
      <xsl:value-of select="@id"/>
    </span>
    <div class="region">
    <xsl:apply-templates/>
    </div>
    </div>
</xsl:template>

<xsl:template match="region">
    <div>
    <xsl:attribute name="id">
      <xsl:value-of select="@id"/>
    </xsl:attribute>
    <span>
    <xsl:attribute name="onclick">
      <xsl:value-of select="$target"/>.switchLayerDisplay
        ('<xsl:value-of select="@id"/>', false);
    </xsl:attribute>
    <xsl:value-of select="@id"/></span>
    <div class="data">
    <xsl:apply-templates/>
    </div>
    </div>
</xsl:template>

<xsl:template match="data">
    <div>
    <span><xsl:value-of select="@year"/>: 
      <xsl:apply-templates/></span>
    </div>
</xsl:template>
</xsl:stylesheet>
```

C.2 Fonctions javascript

```
this.switchLayerDisplay=function(layerName, parent){
  var tag = document.getElementById(layerName);
  var children=tag.childNodes;
  // Repliage arbre/Masquage couches
  if(children[1].style.display=="block"){
    if(layerName!="total") // rien a masquer
      this.setHidden(layerName, true); // masque la couche
    children[1].style.display="none"; // replie les fils directs
    if(parent)
      this.clearChildren(children[1]);
  }
  else{
    if(layerName!="total")
      this.setHidden(layerName,false);
    children[1].style.display="block";
  }
}


this.clearChildren=function(element){
/* annule l'affichage de toute la descendance récursivement.  */
  if(element.getAttribute("class")!="data"){ // fonction d'arret
    var children=element.childNodes;
    var nbChildren=children.length;
    for (var i=0; i<nbChildren; i++){
      // obtient le nom de la couche
      var layer=children[i].getAttribute("id");
      if(layer!="total")
        this.setHidden(layer,true); // masque la couche
      // suppression affichage du groupe d'enfants
      children[i].childNodes[1].style.display="none";
      this.clearChildren(children[i].childNodes[1]);
    }
  }
}
```

C.3 Fichier GML de description des frontières.

Ce fichier représente la description, en GML, des frontières wallonnes. Les autres fichiers GML étant semblables, nous avons choisi de ne pas les transcrire ici.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  http://www.opengis.net/wfs
  http://192.168.0.100:8080/geoserver-1.4.0-RC3
  /schemas/wfs/1.0.0/WFS-basic.xsd">

  <gml:boundedBy>
    <gml:Box srsName="http://www.opengis.net/gml/srs/
    epsg.xml#4326">
      <gml:coordinates xmlns:gml="http://www.opengis.net/gml"
      decimal="." cs="," ts=" ">0.0,48.0 7.0,52.0
      </gml:coordinates>
    </gml:Box>
  </gml:boundedBy>
  <gml:featureMember>
    <gml:MultiPolygon srsName="http://www.opengis.net/gml/srs/
    epsg.xml#4326">
      <gml:polygonMember>
        <gml:Polygon>
          <gml:outerBoundaryIs>
            <gml:LinearRing>
              <gml:coordinates xmlns:gml=
              "http://www.opengis.net/gml"
              decimal="." cs="," ts=" ">
  6.33,50.33 6.27,50.28 6.18,50.15 6.0,50.13 5.94,50.0
  5.89,49.76 <!--Arlon-->5.90,49.60 5.53,49.53 5.3,49.7
  5.07,49.8 4.9,49.82 4.83,50.03 <!--Givet-->4.8,50.17 4.73,50.1
  4.60,49.97 4.37,49.97 4.32,50.05 <!--Erquelinnes-->4.12,50.3
  3.78,50.35 <!--Roisin-->3.68,50.33 3.67,50.43 3.63,50.47
  3.58,50.52 3.28,50.62 3.25,50.7 <!--Mouscron-->3.22,50.73
  3.33,50.68 3.43,50.75 3.68,50.73 3.73,50.73 3.83,50.72
  4.03,50.7 4.13,50.67 4.2,50.68 4.27,50.68 4.38,50.72 4.48,50.73
  4.53,50.72 4.62,50.72 4.7,50.73 4.77,50.78 4.88,50.75 5.03,50.72
  5.22,50.7 <!--Waremme-->5.25,50.68 5.4,50.72 5.6,50.77 5.72,50.72
  5.86,50.7 <!--Aubel-->5.85,50.6 5.90,50.51 5.96,50.48 6.03,50.48
  6.22,50.45 <!--Point EST-->6.33,50.33
              </gml:coordinates>
            </gml:LinearRing>
          </gml:outerBoundaryIs>
```

```
          </gml:Polygon>
        </gml:polygonMember>
        <gml:polygonMember>
          <gml:Polygon>
            <gml:outerBoundaryIs>
              <gml:LinearRing>
                <gml:coordinates xmlns:gml=
                "http://www.opengis.net/gml"
                decimal="." cs="," ts=" ">
                  3.12,50.8 3.01,50.87 3.03,50.78
                </gml:coordinates>
              </gml:LinearRing>
            </gml:outerBoundaryIs>
          </gml:Polygon>
        </gml:polygonMember>
      </gml:MultiPolygon>
    </gml:featureMember>
</wfs:FeatureCollection>
```

C.4 Fichier de données généré

```
<file>
  <country id="belgique">
    <region id="bruxelles">
      <data year="2004">999899</data>
      <data year="2005">1006749</data>
    </region>
    <region id="flandre">
      <data year="2004">6016024</data>
      <data year="2005">6043161</data>
    </region>
    <region id="total">
      <data year="2004">10396421</data>
      <data year="2005">10445852</data>
    </region>
    <region id="wallonie">
      <data year="2004">3380498</data>
      <data year="2005">3395942</data>
    </region>
  </country>
</file>
```

C.5 Feuille de style CSS

```
h1{
     font:  20pt Verdana,sans-serif;
     color:  darkred;
     font-weight:  bold;
     margin-left:15px;
}

span{
     color:darkred;
     font-weight:  bold;
     margin-left:20px;
     font:  14pt Verdana,sans-serif;
}

p{
     color:darkred;
     font-weight:  bold;
     margin-left:20px;
     font:  14pt Verdana,sans-serif;
}

.country{
     cursor:  pointer;
     cursor:  hand;
     display:  block;
}

.region{
     margin:  0px;
     margin-left:  40px;
     cursor:  pointer;
     display:  none;
}

.data{
     padding:  0px;
     margin:  0px;
     margin-left:  40px;
     display:none;
}
```

C.6 Fichier config.xml

Ce fichier a été modifié pour s'adapter à nos besoins. Il est retranscrit ici dans son intégralité.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<MapbuilderConfig version="0.2.1" id="simpleTemplate"
   xmlns="http://mapbuilder.sourceforge.net/mapbuilder"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://mapbuilder.sourceforge.net/mapbuilder
   lib/schemas/config.xsd">

<models>
   <OwsContext id="mainMap">
      <defaultModelUrl>Belgium/belgiumContext.xml</defaultModelUrl>
      <widgets>
         <MapPaneOL id="mainMapWidget">
            <htmlTagId>mainMapPane</htmlTagId>
            <mapContainerId>mainMapContainer</mapContainerId>
         </MapPaneOL>
         <Legend id="contextLegend">
            <htmlTagId>legend</htmlTagId>
            <stylesheet>./Legend2.xsl</stylesheet>
         </Legend>
      </widgets>
      <tools>
         <AoiMouseHandler id="mainAoi"/>
         <DragPanHandler id="mainDragPan">
            <enabled>false</enabled>
         </DragPanHandler>
      </tools>
   </OwsContext

   <Model id="dataRep">
      <defaultModelUrl>./arbre/data.xml</defaultModelUrl>
      <widgets>
         <Widget id="dataTree">
            <htmlTagId>data</htmlTagId>
            <stylesheet>./arbre/data.xsl</stylesheet>
            <targetModel>mainMap</targetModel>
         </Widget>
      </widgets>
   </Model>
</models>
<widgets> </widgets>
<skinDir>lib/skin/default</skinDir>
```

```
<widgetTextUrl>widgetText.xml</widgetTextUrl>
</MapbuilderConfig>
```