UNIVERSITE LIBRE DE BRUXELLES
FACULTE DE SCIENCE APPLIQUEES
ECOLE POLYTECHNIQUE

ANNEE ACADEMIQUE
2001-2002

Sujet de mémoire :

Interrogation de bases de données spatio-temporelles : Conception d'un outil de visualisation des réponses aux requêtes

Directeur de Mémoire
Professeur Esteban Zimányi

Travail de fin d'étude présenté par
CHAHHOU Mohamed pour
l'obtention du grade d'ingénieur
civil informaticien

Remerciements

Je tiens tout d'abord à remercier mon promoteur Mr Esteban Zimányi de m'avoir proposer ce travail, ce n'est pas tous les jours qu'on a l'occasion de travail sur un projet d'une telle envergure. Je le remercie aussi pour son aide, notamment en ce qui concerne ses propositions pour orienter notre travail, ainsi que pour ses nombreuses critiques et commentaires sur mon mémoire, me permettant ainsi de l'améliorer constamment.
Je remercie aussi Benali Choukri, mon partenaire sur ce projet, pour son soutien ainsi que pour les idées qu'il a apportés pour ce travail.
Une attention toute particulière à mes parents vivants à l'étranger ainsi qu'à tous mes collègues qui m'ont soutenu sur ce travail et chez qui j'ai souvent trouvé des encouragements et la confiance en mon travail qui me manquait lors des moments difficiles.

Chapitre 1 : Introduction

Chapitre 1 : Introduction

Les bases de données ont pour but de stocker un ensemble de données qui donneront les informations nécessaires aux différents utilisateurs selon les besoins de chacun. Ces données représentent des phénomènes du monde réel qui intéressent leurs utilisateurs. Si le monde réel est supposé être unique, sa représentation dépend du but à atteindre. De ce fait, différentes applications qui traitent d'un phénomène du monde réel auront des représentations différentes du même phénomène. Ces différences peuvent être : quelle information est gardée, comment est-elle décrite, comment est-elle organisée (en terme de structure de données), comment est-elle codée, comment est-elle présentée, quelles contraintes, processus, et règles lui appliquer, etc.
De nos jours, l'interopérabilité vient remplir le fossé qui existait entre les différents lots de données donnant ainsi la chance de trouver des informations complémentaires sur des faits similaires, ou reliés entre eux dans différentes sources développes indépendamment. Malheureusement, la sémantique de l'interopérabilité reste difficile à définir puisque les connaissances qui sont reliées entre elles sont la plupart du temps décrites dans des termes différents, et utilisent des structures de données différentes. Réconcilier cette hétérogénéité pour construire une base de données complètement intégrée est un problème très complexe qui n'a toujours pas été résolu. Une première étape est l'identification des faits reliés entre eux et la mise au point d'un mécanisme qui pourra d'une certaine manière matérialiser les relations entre différentes représentations d'un même fait. Les DBMS (DataBase Management System) supportant la multi-représentation sont une première étape nécessaire à l'interopérabilité .
Un des exemples les plus évidents où la multi-représentation est primordiale mais n'est pas encore supportée est la cartographie.
La production de cartes est une application importante des bases de données spatiales. La carte permet de visualiser la localisation des données et aussi de visualiser les résultats de requêtes d'analyse spatiale. Un des facteurs importants dont il faut tenir compte lors de la réalisation des cartes est l'échelle, qui influe sur la représentation des données de la carte, par ex : un lac sera représenté par une surface sur une carte au 1/20'000 et par un point sur une carte au 1/100'000 et ne sera plus représenté aux échelles supérieures. Lorsqu'un organisme produit une carte à une échelle donnée et qu'il désire par la suite produire une seconde carte du même territoire à une échelle plus petite, deux possibilités lui sont offertes. Il peut aller recueillir de nouvelles données sur le terrain, engendrant de ce fait des dépenses considérables, ou bien utiliser le même jeu de données initial, à plus grande échelle, pour l'élaboration de ces cartes. Cependant, la carte désirée ayant une échelle plus petite, les entités représentées sur la première carte deviennent plus denses sur la seconde, rendant la lisibilité beaucoup plus difficile. Ainsi, pour palier à ce problème, différentes méthodes ont été mises au point, destinées à simplifier les cartes de plus petite échelle. Ces méthodes peuvent consister en différents filtrages ou modifications des données tel que la suppression des éléments les plus petits, le déplacement d'objets pouvant se chevaucher visuellement, l'agrégation de plusieurs objets semblables, etc. Ce processus est communément appelé généralisation cartographique. Cette méthode présente cependant l'inconvénient de ne pas pouvoir être totalement automatisable et requiert souvent une intervention humaine afin de prendre certaines décisions lors du processus. De plus, la généralisation automatique nécessite des temps de traitement considérables afin de faire fonctionner les algorithmes sur les jeux de données. Ce processus de généralisation, dans un contexte de cartographie numérique, peut

être effectué chaque fois que le besoin s'en fait sentir au sein de l'organisme produisant les cartes. Le résultat de cette généralisation peut alors être conservé et réutilisé par la suite si une carte à une même échelle doit être produite. On peut alors obtenir différents jeux de données représentant les mêmes entités spatiales d'un même territoire. Ces bases de données sont redondantes parce qu'elles décrivent le même espace géographique mais sont gérées indépendamment ce qui entraîne des problèmes d'incohérence des données et ne permet pas la propagation des mises à jour.
Au-delà de la cartographie, la multi-représentation des données géographiques sont nécessaires pour servir des communautés d'utilisateurs multidisciplinaires de façon à ce qu'une même parcelle de terrain puissent supporter des analyses, plannings, et activités forestières par des administrations de villes, environnementalistes, sociologistes, botanistes, zoologiste, etc.
Actuellement, si le concept d'échelle est parfaitement compris et approprié lorsqu'on parle de cartes ou de bases de données cartographiques, il ne l'est plus lorsqu'on parle de la représentation géographique d'un objet dans une base de données (l'échelle est définie comme le rapport entre la taille de l'objet dans la carte et sa taille dans le monde réel). Lorsqu'on parle de bases de données, il serait plus approprié d'utiliser le terme de résolution, définie comme étant la taille minimum d'un objet à représenter. La résolution de l'information dans la base de données est la résolution utilisée lors de l'acquisition de données. Si plusieurs résolutions ont été utilisées pour le même objet, on parlera alors de multi-résolution. Le changement de résolution a un impact sur la forme géométrique des objets (les formes deviennent plus simples à des résolutions moins précises), ainsi que sur l'existence des objets (à cause des changements dans les règles d'agrégations).

Le but principal de ce sujet est la visionneuse de requêtes « Query Viewer ». Cette dernière devra afficher les résultats géographiques et thématiques correspondants aux requêtes en utilisant deux fenêtres synchronisées, la première affiche la carte géographique et la seconde les résultats alphanumériques (ou thématiques) de l'objet.
Le mémoire sera partagé en deux parties. Dans la première partie nous commencerons par décrire de manière un peu plus approfondie, mais simpliste, les concepts de la mutli-représentation et de la multi-résolution spatiale qui sont les points clés du projet MURMUR. Nous procéderons ensuite à une brève discussion sur le contexte de ce travail, à savoir le projet MURMUR dont nous rappellerons les points essentiels. Puis nous développerons de manière détaillée les 3 langages GML, SVG, XSLT qui permettront la transformation de données d'entrées en cartes affichables dans la visionneuse.
La seconde partie sera consacrée à l'architecture da la « Visionneuse de requêtes » et détaillera les fonctions principales de l'interface graphique servant de support à l'affichage des cartes, ainsi que les étapes par lesquelles nous sommes passés pour effectuer la transformation. Nous proposerons aussi un modèle qui pourra être utilisé pour l'affichage des données thématiques.

Chapitre 2 : La multi-représentation et la multi-résolution

Chapitre 2 : La multi-représentation et la multi-résolution

2.1. Introduction

L'élaboration d'une représentation se fait au moyen d'un langage dont la nature détermine le type de représentation. Ainsi un même phénomène du monde réel peut avoir plusieurs représentations selon le langage utilisé. Lorsqu'on travaille avec des bases de données, le type de représentation dépend du modèle de données (par ex : un tuple pour une base de données relationnelle, ou un objet pour les bases de données orientées objets)

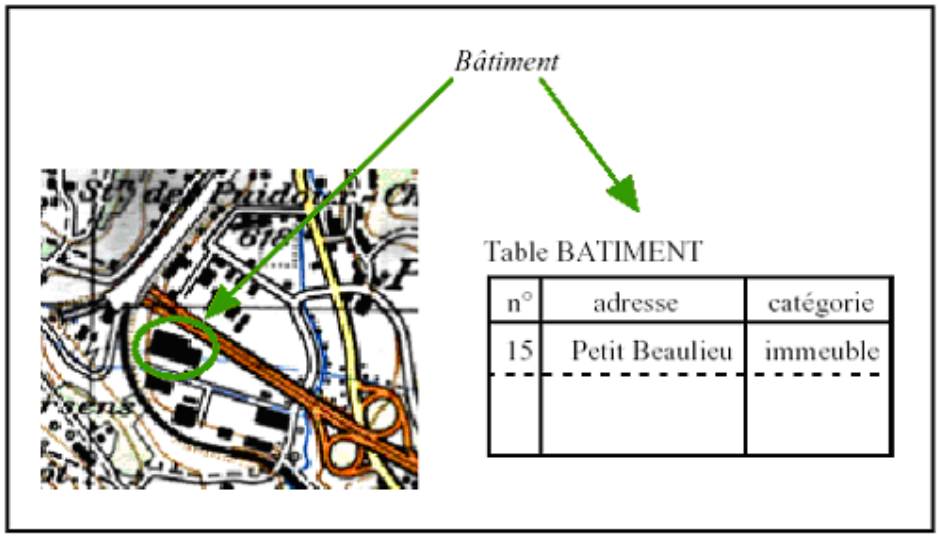


figure 2.1 Deux représentations d'un bâtiment avec des langages différents [1]

2.2. Le point de vue de l'utilisateur

Lors de l'étape d'analyse des besoins des utilisateurs qui consiste à définir les types d'objets, de liens, de propriétés et de méthodes qui seront stockés dans la base de donnes, on identifie le sous-ensemble des phénomènes du monde réel à décrire dans la base de donnes, c'est ce qu'on appelle la délimitation de l'univers du discours.
Cette délimitation de l'univers du discours est spécifique à un groupe d'utilisateurs, qui ne travaillent pas nécessairement sur le même sous-ensemble de phénomènes du monde réel et par conséquent un univers de discours différent.
L'univers de représentation de plusieurs utilisateurs peut donc être disjoint ou avoir certains phénomènes en commun.
Dans la Figure 2.2, les univers de discours de deux utilisateurs sont présentés.
Les deux utilisateurs décrivent les routes, les ponts et les lacs alors que seul l'utilisateur 1 décrit les bâtiments et seul l'utilisateur 2 décrit les avalanches.

[1] Toutes les images sont extraites de la Thèse de doctorat N° 2430 de Christelle VANGENOT
Multi-représentations dans les bases de données géographiques

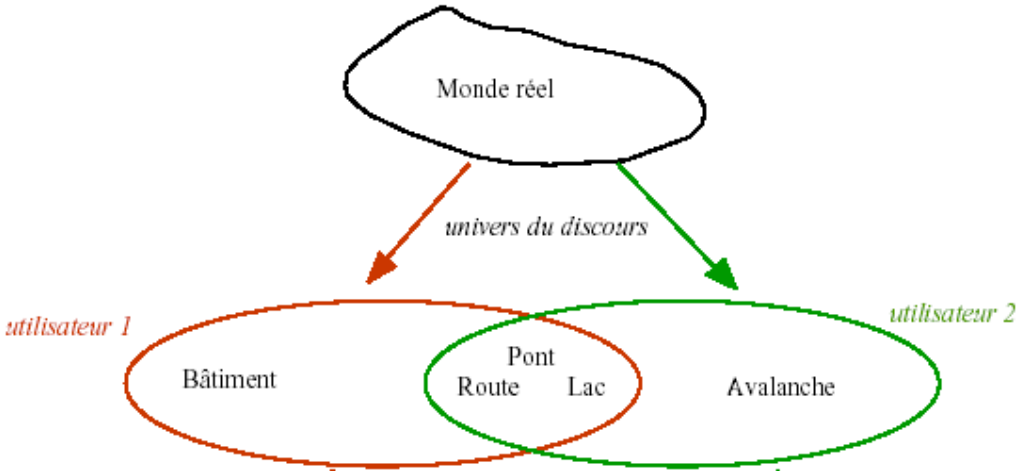


Figure 2.2 univers de discours de deux utilisateurs différents

Le même phénomène peut aussi être décrit différemment selon le groupe d'utilisateurs. Un même phénomène peut ainsi être représenté comme un objet, une association ou un attribut. Par exemple, un pont peut être représenté comme une association entre deux objets de la classe Route, ou comme un objet de la classe Pont.
C'est l'examen des traitements sur les données à représenter qui permet de déterminer la représentation adéquate. Par exemple, on représentera le pont comme un objet si certains traitements portent sur les ponts et comme une association entre routes s'il n'est interrogé qu'à propos des routes. La Figure 3.3 présente deux modélisations différentes d'un pont. A gauche le pont est considéré comme un objet, à droite, il est représenté comme une association entre deux routes.

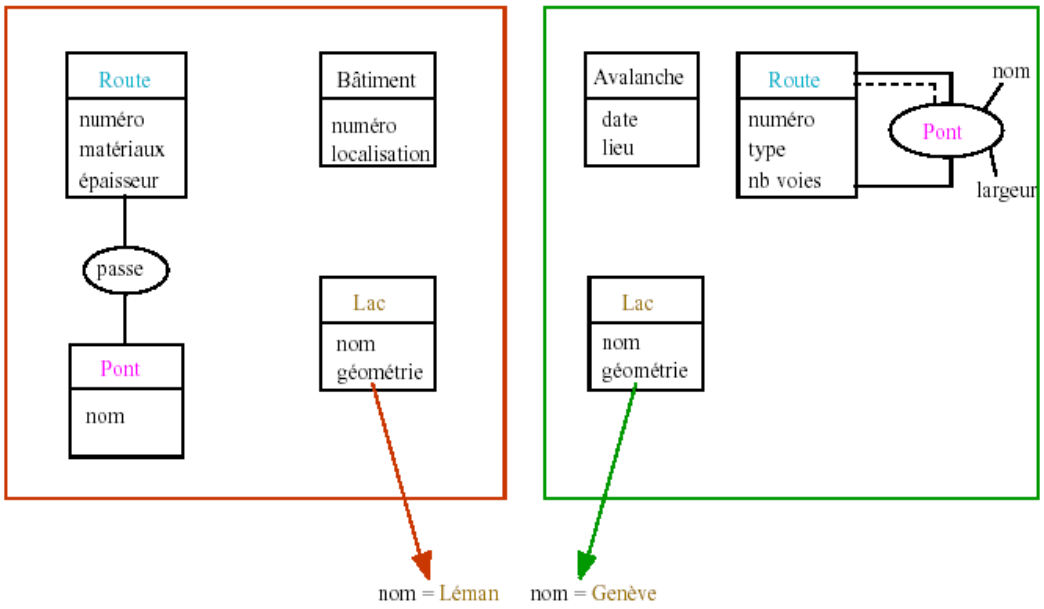


Figure 2.3 deux modélisations différentes d'un pont

Le type de la représentation est une description sous la forme d'un ensemble de couples (nom d'attribut , Domaine) de certaines propriétés du phénomène du monde réel représenté. Différents groupes d'utilisateurs ont besoin de décrire des propriétés différentes pour le même phénomène: le constructeur de la route a besoin de connaître le numéro de la route, les matériaux utilisés pour la construire, son épaisseur … alors que le cartographe ne s'intéresse qu'à son numéro, son type (route nationale ou départementale) et son nombre de voies. La Figure 3.3 présente deux représentations de la même route: les représentations spécifiques au constructeur et au cartographe. Pour chaque représentation, une valeur par attribut est stockée. Dans les bases de données usuelles (mono représentation), pour une propriété donnée il n'existe qu'une seule valeur qui est valable quelque soit l'objectif de la représentation. Or si on considère la propriété *nom* d'un lac, par exemple, différents utilisateurs conserveront une valeur différente: le même lac est effectivement nommé *Lac de Genève* ou *Lac Léman*. Dans les bases de données multi-représentation, certaines propriétés peuvent donc avoir une valeur par représentation.

2.3. association monde réel - base de données

Le processus de représentation associe en principe à un phénomène du monde réel, un élément de la base de données.

Une telle représentation:

- s'exprime à l'aide d'un concept du modèle de données utilisé,
- possède un type, défini par l'ensemble des propriétés qui le composent,
- possède un ensemble de valeurs pour les propriétés décrites dans le type.

Lorsque le phénomène à représenter est spatial, sa représentation comporte en plus des trois parties décrites ci-dessus (aussi appelées parties thématiques), une partie permettant de décrire ses caractéristiques spatiales:

- son type spatial (par exemple point, ligne ou surface)
- sa valeur (géométrie)

2.4. La multi-représentation est un problème à plusieurs dimensions

L'approche conceptuelle de modélisation que nous suivons nous dit que le monde réel sera représenté dans la base de données par un ensemble d'objets, de liens entre objets et de leurs propriétés statiques et dynamiques (attributs et méthodes). Cet ensemble structuré constituera la base de données, sa description en terme de types constituant le schéma de la base.
La vision multidimensionnelle permet de séparer facilement les aspects de perception/représentation (comment les données sont vues par utilisateurs) des aspects d'organisation schématique des descriptions (comment les schémas décrivant les données sont définis).
L'élaboration du schéma avec un modèle classique (par exemple UML) fixe une représentation dans ce que l'on peut appeler la dimension structurelle. Cette représentation est spécifique d'un certain nombre de paramètres parmi lesquels nous avons mis en évidence et retenu pour cette étude le point de vue et la résolution.
L'espace de représentation peut être vu comme un espace multidimensionnel, où chaque dimension est relative à la prise en compte d'un paramètre de représentation.

La dimension structurelle, dite aussi thématique, constitue en quelque sorte la dimension de base de cet espace, car il ne peut y avoir représentation sans définition explicite d'une structure de données.
L'approche retenue sera composée de trois dimensions de multi-représentation:

- Le **point de vue**: les coordonnées sur cet axe représentent les différents points de vue existants. Ces coordonnées constituent un ensemble discret et en principe non ordonné de points.
- La **résolution**: le terme générique de résolution recouvre la résolution spatiale et la résolution sémantique. Les coordonnées sur cet axe représentent les résolutions avec lesquelles les représentations ont été élaborées. Les coordonnées sur cet axe peuvent constituer un ensemble ordonné, par exemple du moins précis au plus précis, ou plus généralement un ensemble partiellement ordonné (les résolutions sémantiques n'étant pas nécessairement comparables).
- La **classification**: les coordonnées sur cet axe constituent un ensemble discret de points qui représentent les différentes instances dans la population de la base de données. Cette dimension permet de rendre compte de la population pertinente pour chaque combinaison <point de vue, résolution>. Cette population est située sur l'axe vertical qui s'élève à partir du point correspondant à l'intersection qui matérialise la combinaison sélectionnée. La dimension rend compte également des choix de classification (et donc d'instanciation) faits lors de la conception du schéma de la base. Cette possibilité de classification multiple participe aussi à la multi-représentation. A un même objet (ou association) du monde réel peuvent ainsi correspondre plusieurs instances dans la base de données. Chaque instance correspond à une coordonnée sur l'axe classification. Une instance d'un type décrivant plusieurs représentations donnera plusieurs points dans notre espace tridimensionnel, tous situés sur un même plan orthogonal à l'axe classification. Si un objet est représenté par un ensemble d'instances décrivant une unique représentation, la ligne reliant les points correspondants sera une ligne zigzaguant dans l'espace en trois dimensions (le changement de classification peut s'accompagner d'un changement de résolution et/ou de point de vue).

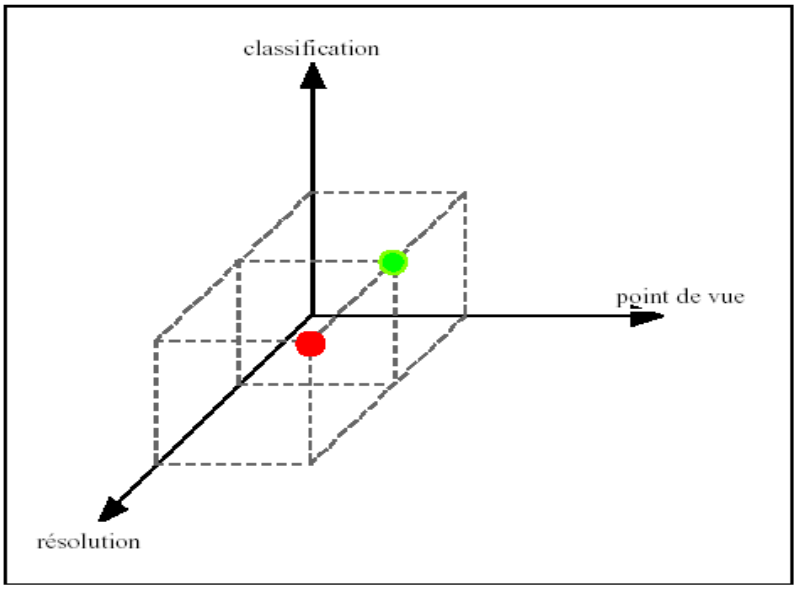


Figure 2.4 : Espace tridimensionnel de représentation

Un point dans notre espace représente une instance en tant que membre de la population d'un type donné, dans un point de vue et une résolution donnée (Figure 2.4). Les deux points dans la Figure 2.4 sont deux représentations du même phénomène décrit au sein d'une seule instance d'un type, selon le même point de vue et à différentes résolutions.
Chaque instance mémorisée dans la base de données doit être caractérisée relativement aux axes de notre espace multidimensionnel, i.e. elle connaît la valeur de ses coordonnées dans cet espace.
D'autres dimensions (comme l'espace et le temps) peuvent venir s'ajouter à notre description sans pour autant altérer les concepts de la multi-représentation proposés. Ainsi ajouter une dimension supplémentaire de représentation (par exemple, une dimension de granularité temporelle) revient simplement à ajouter un axe supplémentaire à l'espace de représentation, pour peu que cet ajout soit fait en respectant l'orthogonalité entre les dimensions. Le modèle MADS offre en plus les concepts nécessaires pour la modélisation des aspects temporels et spatiaux.
La description des types décrivant les points, autrement dit la définition du schéma utilisé, peut être organisée selon plusieurs alternatives. Il est par exemple possible de:

- décrire dans un unique schéma toutes les représentations aux différents points de vue et résolutions. On constitue ainsi une seule base de données multi-représentation couvrant tout l'espace tridimensionnel utilisé.
- créer plusieurs schémas multipoint de vue mais mono-résolution (ou multi-résolution mais pour un point de vue unique). Ceci revient à associer un schéma spécifique à chaque plan orthogonal à l'axe résolution (axe point de vue, respectivement). Il y aura autant de schémas que de coordonnées sur l'axe.
- créer plusieurs schémas mono-point de vue et mono-résolution. Ceci revient à associer un schéma spécifique à chaque axe vertical. Il pourra y avoir autant de schémas que de combinaisons possibles de coordonnées sur les deux axes. Cette approche permet, par exemple, de couvrir par un schéma toutes les données nécessaires à la production d'une carte géographique définie par un seul point de vue et une seule résolution.
- définir, en complémentarité avec l'une des approches précédentes, un schéma intrinsèque s'ajoutant aux autres schémas. Ce schéma intrinsèque décrit les données qui sont indépendantes des résolutions et des points de vue, autrement dit les données pertinentes pour tout utilisateur. La définition d'un schéma spécifique pour ces données "universelles" permet d'éviter la répétition des descriptions dans tous les schémas définis par ailleurs. Dans cette approche, le "vrai" schéma pour une utilisation donnée est constitué de l'union du schéma intrinsèque avec le schéma spécifique de l'utilisation visée.
- mettre en place toute solution hybride en fonction des besoins de l'application.

2.5. Multi-résolution

2.5.1. Un objet = une instance multi-résolution

Pour passer d'une base de données à simple résolution à une base de données multi-résolutions, une solution est de permettre à une instance d'objet d'avoir plusieurs géométries. Chaque géométrie est caractérisée par sa résolution. Les différentes géométries, autres que les points, sont acquises principalement à partir de processus de collections de données, ou de manière interactive par des processus de généralisation cartographique, et doivent être introduites de manière explicite dans les bases de données. Cette approche suit le principe de

représentation : un objet dans le monde réel est traduit en une instance dans la base de données.
Cependant la multi-résolution ne se réduit pas aux géométries multiples. En effet, d'un niveau de résolution à un autre, les objets eux-mêmes changent: une résolution plus détaillée entraîne la description d'un plus grand nombre d'objets et inversement à une résolution moins détaillée certains objets sont agrégés pour former de nouveaux objets d'un type différent. Les relations entre les objets peuvent aussi changer, y compris les associations topologiques. Les attributs thématiques des objets, et même leurs valeurs peuvent changer.
Une base de données multi-résolution doit garder trace de tous les liens nécessaires pour retrouver les représentations qui intéressent chaque utilisateur à une certaine résolution. Les liens d'agrégations sont par exemple nécessaires pour supporter le zoom intelligent.
Un cas particulier dans cette catégorie est celui des bases de données fédérées. Dans ce cas, les utilisateurs accèdent à la base de données à travers un unique schéma intégré qui décrit des instances virtuelles multi-résolution. Mais les instances réelles sont distribuées à travers un ensemble sous-jacent de bases de données mono-résolution participant à la fédération.
Dans ce contexte, la difficulté réside dans le processus d'intégration sémantique, qui consiste à reconnaître les représentations des différentes bases de données qui décrivent les mêmes phénomènes du monde réel ou des phénomènes qui doivent être reliés entre eux, et dans la construction de la table de correspondance entre les objets virtuels intégrés et les représentations existantes. Le modèle définissant le schéma intégré doit permettre la définition de géométries dérivées. Si cette approche intégrant au sein d'une même instance les représentations aux différentes résolutions est aussi utilisée pour les dimensions de point de vue et de classification, le résultat sera une instance comprenant toutes les représentations possibles. Du fait de la complexité des changements auxquels la représentation d'un phénomène du monde réel est sujet lors du passage d'une résolution à une autre, conserver toutes les facettes dans une instance unique peut devenir lourd. Par exemple, l'affichage d'une carte à une échelle donnée nécessitera l'examen de toutes les instances de la base de données pour voir si elles possèdent une géométrie correspondant à la résolution requise et si elles sont localisées dans l'espace couvert par la carte.

2.5.2. Un objet = plusieurs instances à résolution unique

Pour réduire la complexité, il est possible de diviser la représentation d'un phénomène du monde réel en plusieurs représentations interconnectées, chacune étant matérialisée dans la base de données par une instance d'objet. Cette division peut être opérée différemment selon qu'elle doit servir les besoins des utilisateurs ou du système. En effet, les concepteurs de la base de données choisissent comment l'information est présentée aux utilisateurs (dans l'idéal, comment les utilisateurs voudraient la voir présentée). Cependant, la façon dont l'information est effectivement stockée peut être sensiblement différente, parce que le critère essentiel ici est la performance du système ou l'autonomie du site, et non pas le confort des utilisateurs. Nous nous intéressons ici uniquement à la perspective de l'utilisateur.
La division peut s'opérer sur une seule dimension: la résolution, le point de vue ou la classification. Diviser, comme nous l'avons dit, signifie qu'il existe plusieurs instances pour le même phénomène du monde réel. Si la division est opérée selon la résolution (le cas discuté dans cette section), les différentes instances posséderont une géométrie différente, chacune correspondant à une résolution donnée. L'existence d'instances multiples soulève trois questions:

- comment les instances sont classifiées: dans une seule classe d'un seul schéma de base de données, ou dans différentes classes du même schéma, ou dans différentes classes appartenant à des schémas différents.
- comment les instances sont reliées: implicitement, à travers leur mécanisme d'identification, ou explicitement par des liens (par exemple, liens d'association ou de généralisation).
- quelles propriétés sont associées à chaque instance: toutes les propriétés de façon explicite ou seulement les propriétés spécifiques à la résolution de l'instance, les autres propriétés étant héritées des autres instances.

Si toutes les instances sont classifiées à l'intérieur d'une classe unique, disons la classe Bâtiment, les utilisateurs devront avoir recours à un mécanisme d'identification plus complexe (typiquement, l'identificateur "normal" plus un code correspondant à la résolution) pour désigner l'instance qui les intéresse: par exemple, les valeurs pour l'identificateur de bâtiment-id pourront être <bâtiment#**.**résolution-code>, comme par exemple, 372**.**r1, 372**.**r2, etc. Si chaque instance appartient à une classe différente, l'identification se fera par le nom de la classe plus l'identificateur normal. Ce qui signifie que c'est le nom de la classe qui inclut la spécification de la résolution (par exemple, Bâtiment-r1, Bâtiment-r2,...). Les propositions actuelles définissant des instances multiples utilisent toutes la deuxième solution. Plus précisément, elles recommandent de réunir au sein d'un même schéma les types d'objet qui relèvent de la même résolution. Les phénomènes multi-résolution sont donc décrits par un ensemble de schémas mono-résolution. Ces schémas peuvent éventuellement se rapporter à une unique base de données physique. Ce peut être implémenté par la définition de vues au-dessus d'un schéma unique global multi-résolution. Ces schémas peuvent aussi correspondre à des bases de données différentes, une pour chaque résolution.
Pour ce qui est des liens entre instances, la liaison implicite par les identificateurs est possible mais elle n'est pas recommandée. Elle oblige en effet les utilisateurs à gérer les instances multiples, ne permet pas ou peu d'assurer la cohérence, et offrira probablement des performances limitées. Toutes les propositions décrivant des instances multiples définissent explicitement les liens entre instances de même identificateur. Si les types d'objet liés appartiennent au même schéma, les liens inter-représentation sont décrits par un type spécifique d'association, si les types d'objet liés appartiennent à des schémas différents des liens inter-schémas doivent être définis. Au sein d'un même schéma, la sémantique de tels liens est que les instances reliées "représentent le même phénomène du monde réel à des résolutions différentes". La sémantique de ces liens se rapproche donc de la sémantique du lien is-a. Cependant, la description avec le lien is-a d'instances représentant le même phénomène du monde réel à des résolutions spatiale et sémantique différentes ne respecte pas la sémantique d'inclusion qui caractérise le lien is-a dans les systèmes de bases de données actuels. En effet, la description de données à une résolution sémantique moins détaillée résulte en la description d'une classe moins spécifique (et donc une sur-classe) alors que la description de données à une résolution spatiale moins détaillée induit la disparition de certains objets et donc résulte en un ensemble d'objets moins important (et donc une sous-classe). Par exemple, dans le cas d'une base de données décrivant des routes, le passage à une résolution moins détaillée peut entraîner la disparition des petites routes (qui se retrouvent au-delà du seuil de la résolution spatiale) et la fusion des routes parallèles (par exemple, les voies d'une autoroute). En conséquence, deux types d'objet décrivant le même phénomène à des résolutions différentes auront plus certainement des populations en intersection que des populations incluses l'une dans l'autre. Un lien autre que le lien is-a est donc nécessaire. D'autres liens permettant de décrire des relations non plus entre deux instances représentant exactement le même phénomène du monde réel mais entre un nombre quelconque d'instances

d'objets différents (un-à-plusieurs ou plusieurs-à-plusieurs) représentant globalement un même phénomène doivent aussi être décrits. Ces liens sont par exemple les liens d'agrégation. La description de tous ces liens est importante parce qu'elle favorise la manipulation des données spatiales à différentes résolutions (zoom intelligent, navigation parmi les représentations…). Elle permet d'assurer la cohérence des différentes représentations des mêmes phénomènes et par la suite éventuellement de véhiculer des mises à jour.
Quant aux propriétés, l'association à chaque instance de l'ensemble complet des propriétés pertinentes pour cette instance assure la complétude des représentations, et garantit flexibilité et simplicité de la manipulation. Toutefois, cette solution nécessite la définition de contraintes d'intégrité pour s'assurer que les propriétés indépendantes de la résolution, et donc décrites dans plusieurs instances, possèdent et conservent la même valeur dans toutes ces instances. La vérification de telles contraintes d'intégrité étant coûteuse en terme de temps (ce qui diminue la performance du système), les systèmes de gestion de bases de données modernes associent un mécanisme d'héritage au lien is-a. Malheureusement, comme nous venons de le voir, les liens is-a ne sont pas toujours appropriés pour décrire les classifications multi-résolution. Des recherches ultérieures sont donc nécessaires pour étendre l'héritage aux types d'objet ayant des populations en intersection.

Chapitre 3 : Le projet MURMUR

Chapitre 3 : Le projet MURMUR

3.1. Les objectifs du projet MurMur[2]

L'objectif principal du projet MurMur et d'étendre les fonctionnalités actuelles fournies par les logicielles commerciaux de gestion de données (DBMS ou SIG) afin de supporter des schémas de représentation plus flexible, tels que plusieurs représentations coexistantes d'un même phénomène du monde réel, et les représentations temporelles gardant ainsi une trace de l'évolution du phénomène (par ex : on peut associer à une rivière plusieurs géométries pour représenter son évolution le long des années).
L'intérêt principal de MurMur est de fournir un support pour la multiple représentation des données géographiques. L'une des motivations est l'importance des données géographiques : 80 % de tous les problèmes de commerce et de politique utilisent les données géographiques comme élément clé dans le processus de prise de décision (source : NCGIA 1999 National Geodata Forum, Washington, DC, June 1999). Une autre motivation est que la diversité des profils d'utilisateurs est plus importante dans les applications géographiques (environnementalistes, sociologistes, botanistes, zoologiste, …) que dans les applications utilisant des bases de données conventionnelles où une même base de données est utilisée par une communauté d'utilisateurs appartenant à la même organisation.
Deux applications ont été sélectionnées pour définir les besoins en terme de concept de modélisation et pour tester et valider les résultats escomptés à différents niveaux du projet. Les deux applications traitent de problèmes très différents et fournissent ainsi des connaissances complémentaires. De plus, ces applications sont représentatives d'un grand nombre d'applications géographiques nécessitant le support de la multiple représentation.
Le premier cas d'étude est une application cartographique et concerne principalement le problème des bases de données multi-échelles. Pour cette application, plusieurs bases de données à différentes échelles ont été crées de manière indépendante.
Le second cas d'étude est une application de gestion du risque qui gère aussi bien les risques naturels (avalanche, inondations, …) que les risques industriels (pollution, …). Cette application nécessite aussi la manipulation de données à multiple résolution, mais ici l'aspect temporel est d'une importance vitale pour l'analyse du risque et sa prévention.
L'objectif scientifique de ce projet sera une définition formelle d'un modèle de données intégrant la spatio-temporalité, ainsi qu'un langage de définition et de manipulation associé pour permettre la gestion d'objets du monde réel à des résolutions spatiales et sémantiques différentes et à des temps différents. Le modèle enrichi et les langages devront permettre de :

- Associer plusieurs représentations à un objet.
- Définir des relations temporelles et topologiques qui peuvent différer en fonction de la résolution spatiale et de l'évolution temporelle.
- Associer des objets qui se correspondent à différentes résolutions et à différents moments, et finalement,
- Décrire et effectuer des requêtes sur des objets et leurs attributs à différents niveaux de résolution spatiale et sémantique.

[2] Stefano Spaccapietra,Christelle Vangenot, Christine Parent, Esteban Zimanyi : MurMur : A Research Agenda on Multiple Representations, p. 4-8

Les besoins des deux applications de tests sera fondamentalement la détermination du pouvoir d'expression du modèle de données et des langages de définitions et de manipulation associés. Les tests réalisés avec les utilisateurs sur ces applications et leurs conséquences assureront que le modèle et les langages sont simples d'utilisation et qu'ils répondent aux besoins de la multiple représentation.
Partant d'une perspective de développement, MurMur entends développer, tester, et valider une couche logicielle (middleware) au-dessus du SIG de Star Informatic. Cette couche servira d'interface pour les utilisateurs d'une part et pour le SIG d'autre part. Le design et l'implémentation devront maximiser les possibilités de portabilité à d'autres SIG et DBMS.
Les fonctionnalités fournies par la couche MurMur devront permettre aux utilisateurs de :

- Définir et d'éditer un schéma d'une base de données spatio-temporelle en utilisant le modèle de données enrichi.
- Introduire des données dans la base de données et lui permettre d'évoluer en utilisant des opérations d'insertion, de suppression, de mises à jours, tout en préservant les contraintes d'intégrités sur les multiples représentations associées entre elles.
- Effectuer des requêtes sur la base de données en utilisant aussi bien des critères thématiques que spatio-temporelles.

3.2. L'analyse d'application

L'analyse d'application a pour but de définire de façon précise les besoins des applications cibles en terme de modélisation spatio-temporelle incluant les besoins en multiple représentation. Ces nécessites seront évaluées pour savoir ce qui est faisable sans surcharger le système ou l'utilisateur, et seront intégrées dans un ensemble consistant. Et finalement, elles seront formalisés de façon à ce qu'une solution puisse être élaborée. Les deux cas d'étude fourniront des données significatives au projet MurMur et éviter ainsi que les analyses initiales soient déformées par les particularités d'une application spécifique.

3.2.1. Le cas d'étude de la cartographie

Le premier cas d'étude est basé sur 3 bases de données crées de manière indépendante par IGN (l'agence nationale des cartes qui est partenaire du projet), chacune d'elles stocke les données d'une carte à une certaine résolution.
Il n'y a pas de procédure automatisée de propagation de mises à jours d'une base de données vers une autre. L'IGN a beaucoup investit dans le développement d'algorithmes de généralisation cartographique, certains traitants d'objets isolés, d'autres d'objets groupés.

3.2.2. Le cas d'étude de la gestion du risque

Cette partie se concentre sur les besoins d'application d'une administration régionale française dont les responsabilités sont : le planning urbain, les routes et les infrastructures. L'administration coopère, pour le développement de ses applications à données géographique, avec le centre de recherche Cemagref.
Cemagref étudie actuellement une application où le problème de la multiple représentation est particulièrement pertinent. Cette application a pour objectif le développement d'un schéma pour la gestion du risque dans une vallée habitée prés d'un environnement montagneux. Elle doit être capable de gérer aussi bien les risques naturels qu'industriels. En effet, plusieurs usines sont situées dans la vallée, ce qui peut engendrer des catastrophes environnementales

pour les gens habitants aux alentours si jamais les usines subissaient des dommages dus à des catastrophes naturelles.
L'application d'évaluation des risques aura besoin de données provenant de différentes bases de données, pour avoir des informations complémentaires sur l'occupation des terres, les menaces des usines, les caractéristiques géographiques, etc.

3.3. La modélisation des données

Cette phase définira les données modélisant le composant de la solution de Murmure du problème. Le noyau de cette solution est un modèle de données spatio-temporelles qui réponds aux besoins qui ont été éventuellement sélectionnés. Le modèle de données doit être à la fois puissant et simple. Il doit faire avancer l'état de l'art et être encore possible de le mettre en correspondance avec un autre modèle opérationnel existant.
La définition d'un modèle spatio-temporel de base appelé MADS est disponible et servira comme base à ce travail. Le modèle existant a déjà été confronté à des applications utilisant des données géographiques. Le résultat positif nous laisse croire qu'il puisse répondre aux exigences de modélisation spatio-temporelles des applications d'essai de MurMur. Le modèle de données actuel obéit au principe d'orthogonalité, c'est-à-dire, il fournit la fonctionnalité pour les descriptions de structure de données, de l'espace et d temps qui sont indépendants les uns des autres. Cela donne une flexibilité maximale aux designers d'application, qui peuvent concevoir leur définition de données comme ils estiment approprier et sans être ennuyé par des soucis d'implémentation.

3.3.1. Spécification du modèle de données

Le modèle devra supporter des structures de données conceptuelles puissantes, capables de décrire la complexité des phénomènes du monde réel. A cette fin, il inclura des objets avec une structure d'attribut complexe, des relations génériques, des liaisons d'agrégation et des liaisons multi-instantiations.
Pour la description de l'espace, le modèle permettra aux caractéristiques spatiales d'être associé à des objets, à des liaisons ou à des propriétés, selon les besoins de l'application. Plus particulièrement, il fournira des facilités pour la description explicite de rapports spatiaux et des contraintes associées. Il devra supporter à la fois la vue discrète (ou vecteur) et la vue continue (ou raster) de l'espace.
Pour la description du temps, le modèle doit principalement traiter avec un support de temps valide, bien que la transaction de temps puisse être considérée. Les caractéristiques du temps peuvent être associées à des objets, des relations, ou à des propriétés, selon les besoins de l'application.
Ils couvriront la sémantique de cycle de vie de l'objet, ainsi que la sémantique de l'histoire de l'objet. Ainsi, il doit être possible d'effectuer une requête pour s'assurer de l'existence d'un objet à un moment donné, de son statut en ce qui concerne le cycle de vie et de l'histoire des propriétés choisies. Finalement, par la combinaison des spécifications de l'espace et du temps, le modèle devra supporter la description et la gestion d'objets se mouvant ou changeants de forme (par exemple, une inondation, une avalanche, une équipe de secours).

3.3.2. Acquisition des données

Cette étape va définire les procédures d'acquisition des données qui établiront une correspondance entre les données pertinentes extraites des deux applications test et le modèle de donnée de MurMur. Les données de test sont actuellement stockées dans des SIG

supportant leurs propres règles et concepts de modélisation de données. Un sous-ensemble pertinent devra être extrait de ces données et introduit dans MurMur pour démontrer les fonctionnalités de MurMur. On n'a pas l'intention de développer un outil de « reverse engineering » qui fera le travail automatiquement, même si un tel outil est très désirable, son design et son implémentation sera une surcharge majeure qui fera éloigner MurMur ses objectifs principaux. On développera plutôt des procédures ad hoc qui seront exécutées de manière interactive, selon les besoins en tenant compte des besoins que l'on veut analyser.

3.4. Manipulation des données

En supposant qu'une base de données puisse être installée et que les données sont acquises selon le nouveau modèle de donnée de MurMur, cette phase a pour objectif de rendre accessible les données aux applications. Une interface pour les formulations de requêtes sera définie en se basant sur un langage formel qui sera défini en premier lieu. Comme pour le design du modèle de données de MurMur, l'élaboration du langage de requête associé proviendra de l'analyse des exigences des applications, formulé comme un ensemble de requêtes et services typiques dont on aura besoin dans les deux cas d'étude.

3.4.1. Le langage de requête

Une analyse des exigences des applications en terme de manipulation de données sera faite avec la participation des utilisateurs finaux, et avec une forte interdépendance avec les analyses d'applications. Cela fournira des informations pour définire un langage de requête spatio-temporel approprié qui couvrira les caractéristiques de la multiple représentation, et pour spécifier une interface qui le supportera.
Le résultat le plus souhaitable en terme de convivialité pour l'utilisateur sera probablement une interface visuelle offrant une formulation de requêtes multi-paradigme, même si ce résultat reste difficile à atteindre. Beaucoup de recherches sont encore nécessaires pour identifier une interface visuelle qui aura un total pouvoir d'expression, qui sera suffisamment flexible pour s'ajuster aux différentes approches de formulation de requêtes (ex : navigation en utilisant la voix ou des gestes) et rester suffisamment simple pour être comprise et utilisée. Ce projet se limitera à offrir une interface simple mais puissante, combinant la manipulation directe avec des spécifications textuelle ou à menus déroulant, qui sera capable de prendre ne compte toutes les exigences à travers des techniques sophistiquées mais faciles à utiliser. Typiquement, un tel éditeur visuel de requêtes affichera d'abord le diagramme du schéma de la base de données que l'utilisateur veut interroger. Apres, l'utilisateur sélectionnera les éléments des données (objets, relations, attributs) qui l'intéressent pour la requête, et de façon similaire, spécifier la structure des résultats qu'il recherche. Cela utilisera principalement des opérations tel que le click et le glisser/déposer. La spécification de la sélection des prédicats sera la partie la plus difficile à réaliser puisque les prédicats seront assez complexes avec parfois des formulations non intuitives, et incluront des composants spatio-temporels pour lesquels il n'y a pas d 'approche de formulation visuelle bien établie. Le langage d'interrogation lui-même sera formellement défini comme un jeu d'opérateurs algébriques, dont les propriétés et les règles seront précisément exposées. Tandis que le noyau du langage sera formé d'opérateurs bien connus, façonnés aux structures de données du MurMur et aux particularités spatio-temporelles, des opérateurs additionnels et des règles seront définis pour tenir compte des particularités de la représentation multiple.

3.4.2. Le zoom intelligent

Une opération de zoom fait passer une zone d'affichage d'une certaine surface ou objet à une résolution donnée vers une zone d'affichage de la même surface ou objet à une autre résolution. Une opération de zoom physique est purement une opération géométrique. Un zoom intelligent est un zoom qui affiche, à l'échelle désirée, des objets ou des champs qui sont pertinents à cette échelle. Par ex : zoomer pour agrandir un objet qui est affiché comme un bâtiment unique peut révéler que ce bâtiment « fictive » représente au fait deux bâtiments séparés, qui ont été fusionnés à une échelle moins précise parce qu'ils étaient trop proches l'un de l'autre pour pouvoir les représenter séparément.

3.4.3. Le voyage dans le temps

Quand la dimension temporelle est prise en compte, différentes représentations d'une même zone ou objet correspondent à différents moments dans le temps.
Le voyage dans le temps permettra de visualiser l'information à propos d'une région ou d'un objet à un certain moment dans le temps. Cependant, cette fonctionnalité est connue pour être difficile à réaliser, en particulier parce qu'une information sur un objet ne sera pas connue à un certain moment dans le temps qui intéresse l'utilisateur. Des fonctions d'interpolation seront donc nécessaires.

3.5. L'implémentation

Un prototype devra implémenter le modèle de données, le langage de requêtes, et d'autres services qui ont été développés durant les phases précédentes du projet. L'objectif est d'implémenter un prototype sous forme d'un package logiciel qui sera intégré dans le SIG de Star Informatic. Cependant, l'implémentation sera conçue pour faciliter dans l'avenir le développement d'une interface entre MurMur et d'autres SIG et DBMS.
La traduction des structures de MurMur vers les structures Star Informatic sera effectuée en deux étapes. La première étape simplifiera le schéma de MurMur par un jeu d'opérations de restructuration de schéma prédéterminées. Le but de cette étape est d'obtenir un schéma de MurMur qui utilise seulement la modélisation simple, comme celle trouvée dans les logiciels existants.
La deuxième étape réécrit le schéma de MurMur simplifié en utilisant les constructions spécifiques du système cible. L'avantage de cette décomposition du processus de traduction est que les modules effectuant la première étape peuvent être réutilisés par d'autres systèmes. Cela améliore la portabilité et la réutilisation du code et augmente les capacités d'évolution du produit. Un souci semblable pour la portabilité sera mis en avant pour le développement du schéma et de l'éditeur de requête.

Chapitre 4 : Les langages GML, SVG et XSLT

Chapitre 4 : Les langages GML, SVG, XSLT

4.1. Le choix des langages

L'éditeur de requêtes fait partie intégrante de la visionneuse de requête. Après avoir formuler une requête, celle-ci sera envoyée à une base de données contenant les informations qu'on désire extraire. La réponse à la requête est renvoyée dans un format structuré, en utilisant le langage GML qui est un langage de transfert et de stockage d'information géographique incluant les propriétés spatiales et non spatiales des entités géographiques.
Cette réponse ne peut être affichée directement dans la visionneuse de requête sans un traitement préalable, en effet GML contient juste des informations sur les entités géographiques et non pas la façon dont elles devront être affichées.
Pour résoudre ce problème, nous allons devoir utiliser un autre langage en parallèle avec GML pour nous affranchir de ces contraintes. Le choix du langage à utiliser s'est porté sur SVG. Ce choix a été dicté par ses nombreuses propriétés avantageuses par rapport aux autres formats d'affichage. En effet GML et SVG étant tous les deux des applications de XML, le passage de l'un à l'autre sera plus facile. Les autres avantages seront détaillés dans la section se rapportant à SVG.
La transformation de GML vers SVG sera effectuée par le langage XSLT. Ce dernier est un langage écrit en XML, sa structure étant définie en XML. Ce langage permet de transformer un flux XML en un autre flux XML

4.2. GML 2.0

4.2.1. Introduction

GML (Geography Markup Language) est une application de XML pour le transfert et le stockage d'information géographique, ce qui inclut les propriétés spatiales et non spatiales des entités géographiques. Cette spécification définit les mécanismes, la syntaxe et les conventions des schémas XML pour

- Fournir une architecture ouverte, indépendante de vendeurs, pour la définition des schémas et des objets d'applications géospatiales ;
- Permettre la définition de profils qui supportent des sous-ensembles des possibilités descriptives de l'architecture GML ;
- Supporter la description de schémas d'applications géospatiales dans des domaines et des communautés spécifiques ;
- Permettre la création et la maintenance des schémas et des données d'applications géographiques connectées ;
- Supporter le stockage et le transfert des schémas et des données des applications ;
- Augmenter la capacité des organismes à partager les schémas des applications géographiques et l'information.

Les développeurs peuvent décider de stocker l'information géographique en GML et les schémas des applications géographiques ou choisir de convertir à la demande à partir d'un autre format de stockage et de n'utiliser GML que pour les schémas et le transfert des données.

4.2.1.1. Les objectifs

Le développement de GML obéit à certains objectifs, dont certains sont communs avec ceux de XML :

- Fournir un moyen de codage de l'information spatiale pour le transfert et le stockage des données plus particulièrement dans un contexte Internet.
- Être suffisamment extensible pour permettre une grande variété de tâches, de l'interprétation à l'analyse spatiale.
- Établir les fondations d'un Système d'Information Géographique (SIG) sur Internet d'une manière modulaire et incrémentielle.
- Permettre un codage efficace des géométries géospatiales (par exemple la compression des données).
- Fournir un codage de l'information spatiale et des relations spatiales faciles à comprendre, y compris les codages définis dans le modèle des entités simples de l'OGC.[3]
- Être capable de séparer les contenus spatiaux et non spatiaux de la présentation (graphique ou autre) des données.
- Permettre une intégration facile des données spatiales et non spatiales, particulièrement lorsque les données non spatiales sont codées en XML.
- Être capable de relier facilement les éléments spatiaux (géométriques) aux non-spatiaux ou à d'autres éléments spatiaux.

[3] Open GIS Consortium

- Fournir un ensemble de modèles d'objets géographiques courants pour permettre l'interopérabilité d'applications développées indépendamment.

GML est conçu pour permettre l'interopérabilité et la réalise par la mise à disposition de balises géométriques de base (tous les systèmes qui supportent GML utilisent les mêmes balises géométriques), un modèle de données commun (entités/propriétés) et un mécanisme pour créer et partager des schémas d'application. La plupart des communautés d'information vont chercher à améliorer leur interopérabilité par la publication de leurs schémas d'applications.

4.2.2.2. Concept d'entité

Ce chapitre présente une introduction aux concepts clés requis pour comprendre comment GML modélise le monde. Ils sont basés sur les spécifications abstraites d'OpenGIS (« OpenGIS Abstract Specifications » sous le lien http://www.opengis.org/techno/specs.htm) qui définissent une entité géographique ('geographic feature') comme « *une abstraction d'un phénomène du monde réel ; c'est une entité géographique si elle est associée à une position sur la Terre* ». Ainsi, une représentation numérique du monde réel peut être vue comme un ensemble d'entités géographiques (qu'on abrègera en « entité » dans le document). L'état d'une entité est défini comme un ensemble de propriétés, chacune de celles-ci étant définie par un triplet {nom, type, valeur}. Le nombre de propriétés (ainsi que le nom et type) qu'une entité peut avoir est déterminé par le type de l'entité. Les entités géographiques sont celles dont les propriétés peuvent avoir une valeur géométrique. Une collection d'entités peut être considérée comme une entité ; par conséquence, une collection d'entités a un type et peut avoir ses propres propriétés en plus des entités qu'elle contient.
Cette spécification n'est concernée que par ce que l'OGC appelle les entités simples ('simple feature'). Ce sont les entités dont les propriétés géométriques sont limitées à la « géométrie simple » pour laquelle les coordonnées sont définies en deux dimensions et le tracé d'une courbe entre deux coordonnées est interpolé de façon linéaire. Bien que cette version de GML permette de spécifier des coordonnées en 3 dimensions, elle ne fournit pas de support direct à des constructions géométriques en 3D.
Comment GML doit-il être utilisé pour représenter un phénomène du monde réel ? Supposons que quelqu'un souhaite construire une représentation numérique de la ville de Cambridge en Angleterre. Celle-ci peut être représentée comme une collection d'entités dans laquelle les entités représentent des rivières, des routes, des collèges ; cette classification des phénomènes du monde réel indique le type d'entités qui doit être défini. Le choix de cette classification est lié à l'usage éventuel de la représentation numérique. L'entité 'River' (rivière) peut avoir une propriété 'name' (nom) dont la valeur est de type 'string' (chaîne). Il est courant d'avoir des propriétés typées ; ainsi, dans l'exemple précédent, on dit que le type d'entité 'River' a une propriété appelée 'name' dont la valeur doit être du type chaîne. De la même manière, l'entité 'Road' (route) peut avoir une propriété chaîne appelée 'classification' et une *propriété entière* appelée 'number' (nombre). Les propriétés de type simple (entier, chaîne, réel, booléen) sont appelées des *propriétés simples*.
Les entités nécessaires pour modéliser Cambridge peuvent avoir des propriétés géométriques comme des propriétés simples. Comme les autres propriétés, les propriétés géométriques doivent être nommées. Ainsi, le type de l'entité 'River' peut avoir une propriété géométrique appelée 'centerLineOf' et le type de l'entité 'Road' peut avoir une propriété géométrique appelée 'linearGeometry'. Il est possible d'être plus précis sur le type de géométrie qui peut être utilisée comme valeur de propriété. Ainsi, dans les exemples précédents, la propriété géométrique peut être précisée comme étant une propriété 'linestring' (ligne brisée). Il est

courant d'avoir plusieurs propriétés simples définies dans le type d'une entité, de même celui-ci peut aussi avoir plusieurs propriétés géométriques.

4.2.2. Les modèles de la géométrie et des entités

Le modèle des entités simples ('Simple Features') représente une simplification du modèle plus général décrit dans les spécifications abstraites d'OpenGIS. Il y a deux simplifications majeures :

- Les entités sont supposées n'avoir que des propriétés simples (booléens, entiers, réels, chaînes) ou des propriétés géométriques ; et
- Les géométries sont supposées être définies dans un SRS à deux dimensions et utiliser l'interpolation linéaire entre deux coordonnées.

Un certain nombre de conséquences découlent de ces simplifications. Par exemple, les entités simples peuvent seulement supporter les données vectorielles ; et les entités simples ne sont pas suffisantes pour expliciter un modèle topologique. Cette version de GML aborde la première de ces limitations en permettant aux entités d'avoir des propriétés non géométriques complexes ou globales. Ces propriétés complexes peuvent elles-mêmes être composées d'autres propriétés simples ou complexes. Les exemples courants sont les dates, les heures et les adresses. Il est prévu que les futures versions de GML abordent la seconde limitation et disposent de modèles géométriques plus élaborés.

Le modèle objet de la géométrie pour les entités simples (Figure 4.1) a une classe (abstraite) de base 'Geometry' et associe chaque objet géométrique avec un SRS qui décrit l'espace de coordonnées dans lequel l'objet est défini. GML reprend cette hiérarchie de classes en supprimant certains types intermédiaires comme 'Curve' (Courbe), 'Surface', 'MultiSurface' et 'MultiCurve' (MultiCourbe).

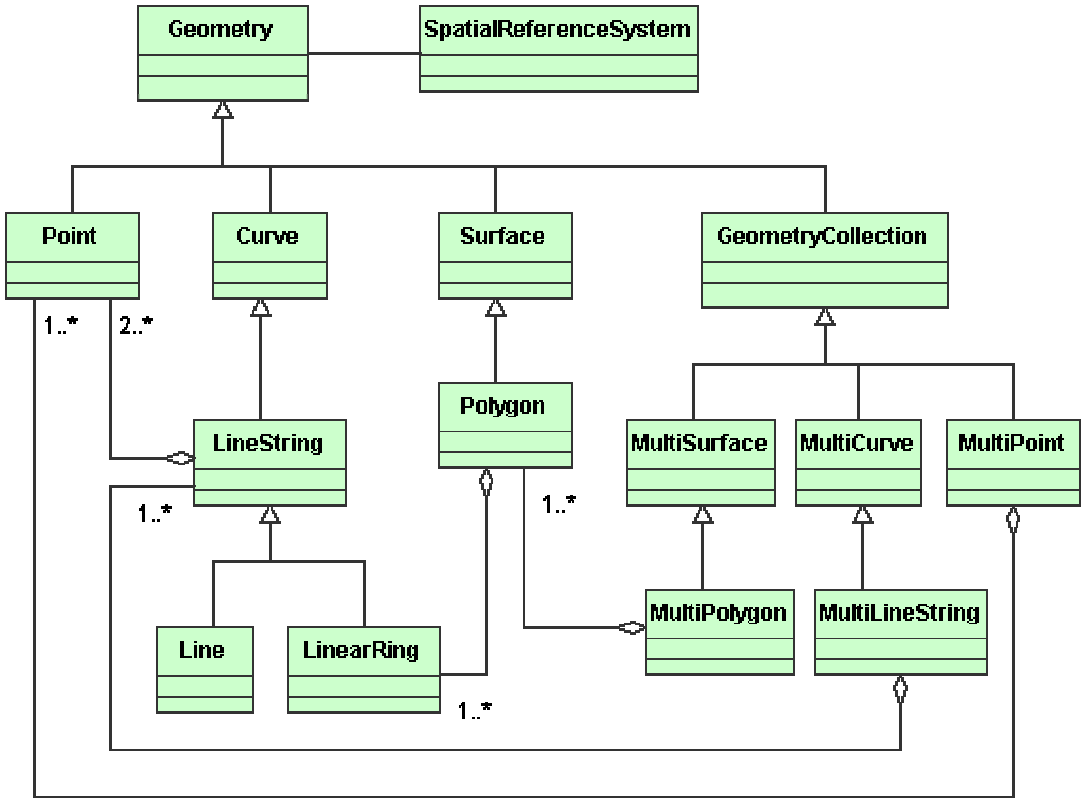


Figure 4.1 : Le modèle de la géométrie pour les entités simples[4]

4.2.3. Schémas pour les données géospatiales

En termes généraux, un schéma définit les caractéristiques d'une classe d'objets ; avec XML un schéma décrit aussi les balises des données. GML a aussi été développé pour être compatible avec la recommandation des espaces de noms XML. Les espaces de noms sont utilisés pour distinguer entre elles les définitions des entités géographiques et des propriétés définies dans des domaines spécifiques aux applications ainsi que des concepts de base définis par les modules GML.

Les types des entités géospatiales peuvent être examinés séparément de leur schéma associé. En d'autres termes, un type (ou une classe) 'Road' existe indépendamment de la définition de son schéma qu'il soit exprimé sous forme d'une DTD ou d'un schéma XML. En GML 2.0, les types géospatiaux sont des noms d'éléments mais ces noms supposent l'existence de types au niveau du modèle indépendamment du codage du schéma XML. Considérons l'exemple suivant : un type 'Road' est introduit dans un schéma d'application par la déclaration d'un élément global <Road> du type 'RoadType' (e. g. <element name="Road" type="RoadType" />. Nous remarquons l'interaction de deux niveaux : conceptuel et mise en œuvre. Déclarer que 'width' (largeur) est une propriété d'une route, est une affirmation au niveau du modèle sans rapport avec le fait que la largeur soit un nombre décimal avec deux décimales, ou un entier ; il s'agit de caractéristiques des processus et des données au niveau mise en œuvre.

[4] Tous les exemples et figures proviennent du site : http://opengis.net/gml/01-029/GML2.html

4.2.4. Schémas d'application

Trois documents des schémas XML de base sont fournis par GML : « feature.xsd » qui définit le modèle général des propriétés des entités, « geometry.xsd » qui contient les composants géométriques détaillés et « xlinks.xsd » qui fournit les attributs Xlink utilisés pour mettre en application les liens. Par eux-mêmes, ces trois documents ne fournissent pas un schéma approprié pour contraindre les instances de données ; ils fournissent plutôt les types et les structures de base qui peuvent être utilisés par un schéma d'application. Un schéma d'application déclare les types des entités réelles et les types des propriétés intéressants pour un domaine particulier en utilisant les composants GML de manière standard. Globalement, ce système implique la définition de types spécifiques à l'application qui sont dérivés des types des schémas GML standards ou qui incluent directement les éléments et les types des schémas GML standards.

Les schémas GML de base fournissent un méta-schéma ou un ensemble de classes de base à partir desquelles un schéma d'application peut être construit. Les schémas d'application écrits par l'utilisateur peuvent déclarer des éléments et/ou définir des types pour nommer et distinguer les unes des autres des entités et des collections d'entités
Le schéma de la géométrie GML contient les définitions de type pour les éléments abstraits de la géométrie et les éléments concrets (multi) point, ligne et polygone, ainsi que les définitions de type complexe pour les types géométriques sous-jacents.
Comme pour le schéma de la géométrie, le schéma des entités définit les éléments et types abstraits et concrets.

4.2.5. Codage GML

4.2.5.1. Codage des entités sans géométrie

Même s'il n'est pas prévu que beaucoup d'entités soient codées en GML sans propriétés géométriques, ce chapitre commence par un exemple basé sur une entité non spatiale. Nous l'appellerons l'exemple 'Dean' (Doyen). C'est une entité qui est définie comme ayant une propriété Chaîne appelée 'familyName' (Nom de famille) et une propriété entière appelée 'age' (age). De plus, une entité 'Dean' peut avoir zéro ou plusieurs propriétés chaîne appelées 'nickName' (surnom). Une instance de l'entité 'Dean' peut alors être codée comme ceci en XML :

```
<Dean>
  <familyName>Smith</familyName>
  <age>42</age>
  <nickName>Smithy</nickName>
  <nickName>Bonehead</nickName>
</Dean>
```

Sans connaître GML, le schéma XML correspondant peut être défini comme ceci :

```
<element name="Dean" type="ex:DeanType" />
<complexType name="DeanType">
  <sequence>
    <element name="familyName" type="string"/>
```

```
    <element name="age" type="integer"/>
    <element name="nickName" type="string" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

Néanmoins, il est nécessaire d'utiliser le schéma des entités GML pour identifier et distinguer les entités(et leurs types) et les propriétés. Dans l'exemple, 'Dean' est un type d'entité et 'age' est une propriété. Voilà comment cela est indiqué en GML :

```
<element name="Dean" type="ex:DeanType"
  substitutionGroup="gml:_Feature" />

<complexType name="DeanType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="familyName" type="string"/>
        <element name="age" type="integer"/>
        <element name="nickName" type="string" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Notons que non seulement l'instance de document est valide par rapport aux deux définitions de schéma XML, mais le modèle du contenu de 'DeanType' est le même dans les deux définitions de schéma XML. L'utilisation du schéma des entités de GML permet l'utilisation des attributs prédéfinis. Par exemple, les entités peuvent être identifiées par l'attribut 'fid' et les entités peuvent être décrites en utilisant la propriété prédéfinie 'gml:description'. Ces possibilités sont héritées de 'gml:AbstractFeatureType' (pour être plus précis 'Dean' étend 'gml:AbstractFeatureType').

```
<Dean fid="D1123">
  <gml:description>A nice old chap</gml:description>
  <familyName>Smith</familyName>
  <age>42</age>
  <nickName>Smithy</nickName>
  <nickName>Bonehead</nickName>
</Dean>
```

4.2.5.2. Codage de la géométrie

Ce chapitre décrit comment GML code les types géométriques de base et il introduit le schéma de la géométrie GML (« GML Geometry schema » - 'geometry.xsd') sur lequel s'appuie ce codage.

L'élément coordonnées (<coordinates>)

Les coordonnées de toutes les instances de classe géométrique sont codées soit par une séquence d'éléments <coord> qui contient les composants du tuple, soit par une chaîne contenue dans un élément <coordinates>. L'avantage d'utiliser des éléments <coord> est qu'un analyseur de validation XML peut effectuer des vérifications de type et faire respecter des contraintes sur le nombre de tuples qui apparaissent dans une instance particulière de la géométrie. Les deux approches peuvent exprimer des coordonnées en une, deux ou trois dimensions. L'extrait de schéma XML correspondant est le suivant :

```
<element name="coord" type="gml:CoordType" />

<complexType name="CoordType">
  <sequence>
    <element name="X" type="decimal"/>
    <element name="Y" type="decimal" minOccurs="0"/>
    <element name="Z" type="decimal" minOccurs="0"/>
  </sequence>
</complexType>
```

Un niveau supplémentaire de contraintes restreint le nombre de tuple pour un type de données. Par exemple, un élément <Point> contient exactement un tuple de coordonnées :

```
<Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <Coord><x>5.0</x><y>40.0</y></Coord>
</Point>
```

L'élément coordonnées (<coordinates>) est défini par l'extrait de schéma XML suivant :

```
<element name="coordinates" type="gml:CoordinatesType"/>
<complexType name="CoordinatesType">
  <simpleContent>
    <extension base="string">
     <attribute name="decimal" type="string" use="default" value="."/>
     <attribute name="cs" type="string" use="default" value=","/>
     <attribute name="ts" type="string" use="default" value="&#x20;"/>
    </extension>
  </simpleContent>
</complexType>
```

Ceci permettra à l'exemple Point ci-dessus d'être codé comme ceci :

```
<Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coordinates>5.0,40.0</coordinates>
</Point>
```

4.2.5.3. Eléments géométriques de base

Les coordonnées des éléments géométriques sont définies dans un Système de Référence Spatial (SRS), et tous les éléments géométriques doivent préciser ce SRS. GML 2.0 n'aborde pas en détail la définition des systèmes de référence spatiaux. Une proposition de spécification basée sur XML est actuellement proposée pour manipuler les systèmes de référence des coordonnées et les transformations de coordonnées. L'attribut 'srsName' des types géométriques peut être utilisé pour tester l'équivalence des SRS entre les différentes géométries. L'attribut 'srsName' (puisque c'est une référence URI) peut être parcouru pour accéder à la définition du SRS.

L'élément Point (<Point>)

L'élément Point est utilisé pour coder les instances de la classe géométrique Point. Chaque <Point> contient soit un unique élément <coord>, soit un élément <coordinates> contenant exactement un tuple de coordonnées; l'attribut 'srsName' est optionnel puisqu'un élément Point peut être contenu dans d'autres éléments qui spécifient un système de référence. Des considérations similaires s'appliquent aux autres éléments géométriques. L'élément Point, comme les autres types géométriques, a aussi un attribut optionnel 'gid' qui sert d'identificateur. Voici un exemple :

```
<Point gid="P1"
      srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>56.1</X><Y>0.45</Y></coord>
</Point>
```

L'élément Cadre (<Box>)

L'élément 'Box' est utilisé pour coder les emprises. Chaque <Box> contient soit une séquence de deux éléments <coord>, soit un élément <coordinates> contenant exactement deux tuples de coordonnées ; le premier est construit avec les valeurs minimum mesurées sur tous les axes, le second avec les valeurs maximum mesurées sur tous les axes. Une valeur doit être donnée à l'attribut 'srsName' puisqu'un cadre ne peut pas être inclus dans une autre classe géométrique. Une instance de cadre ressemble à ceci :

```
<Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>0.0</X><Y>0.0</Y></coord>
  <coord><X>100.0</X><Y>100.0</Y></coord>
</Box>
```

L'élément Ligne (<LineString>)

Une ligne est en fait une ligne brisée, c'est-à-dire un chemin linéaire d'un seul tenant défini par une liste de coordonnées qui sont supposées connectées par des segments de ligne droite.

Un chemin fermé est indiqué en ayant les mêmes premières et dernières coordonnées. Au moins deux coordonnées sont requises. Voici un exemple d'instance de ligne :

```
<LineString srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>0.0</X><Y>0.0</Y></coord>
    <coord><X>20.0</X><Y>35.0</Y></coord>
  <coord><X>100.0</X><Y>100.0</Y></coord>
</LineString>
```

L'élément Anneau linéaire (<LinearRing>)

Un anneau linéaire est un chemin linéaire fermé défini par une liste de coordonnées qui sont supposées connectées par des segments de ligne droite. La dernière coordonnée doit coïncider avec la première et au moins quatre coordonnées sont requises (les trois pour définir un anneau et la quatrième identique à la première). Comme un 'LinearRing' est utilisé dans la construction des Polygones (qui définissent leur propre SRS), l'attribut 'srsName' n'est pas nécessaire.

L'élément Polygone (<Polygon>)

Un polygone est une surface connexe. Deux points quelconques dans le polygone peuvent être reliés par un chemin. La limite du polygone est un ensemble de 'LinearRing' ; Nous distinguons la limite externe (extérieure) et la limite interne (intérieure). Les 'LinearRing' de la limite interne ne peuvent en croiser aucun autre et ne peuvent être contenus les uns dans les autres. Il doit y avoir au plus une limite extérieure et zéro ou plusieurs limites intérieures. L'ordre des 'LinearRing', soit dans le sens des aiguilles d'une montre, soit l'inverse, n'est pas important. Dans l'exemple suivant, le polygone a deux limites intérieures et utilise les chaînes de coordonnées :

4.2.5.4. Codage des entités avec une géométrie

GML 2.0 fournit un ensemble prédéfini de propriétés géométriques qui peuvent être utilisées pour relier des géométries de type particulier aux entités. Considérons le cas dans lequel la définition de l'entité 'DeanType' a une propriété Point appelée 'location' (localisation) qui est un nom descriptif prédéfini qui peut se substituer au nom formel 'pointProperty'.

```
<Dean>
  <familyName>Smith</familyName>
  <age>42</age>
  <nickName>Smithy</nickName>
  <nickName>Bonehead</nickName>
  <gml:location>
    <gml:Point>
      <gml:coord><gml:X>1.0</gml:X><gml:Y>1.0</gml:Y></gml:coord>
    </gml:Point>
  <gml:location>
</Dean>
```

Cette instance est basée sur l'extrait de schéma d'application suivant :

```
<element name="Dean" type="ex:DeanType"
       substitutionGroup="gml:_Feature"/>

<complexType name="DeanType">
 <complexContent>
  <extension base="gml:AbstractFeatureType">
   <sequence>
    <element name="familyName" type="string"/>
    <element name="age" type="integer"/>
    <element name="nickName" type="string" minOccurs="0"
     maxOccurs="unbounded"/>
    <element ref="gml:location"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

Autrement, on peut définir des propriétés géométriques spécifiques à un schéma d'application. Par exemple, on peut vouloir appeler 'deanLocation' la propriété précisant la localisation de l'instance 'Dean' :

```
<element name="Dean" type="ex:DeanType"
       substitutionGroup="gml:_Feature"/>

<complexType name="DeanType">
 <complexContent>
  <extension base="gml:AbstractFeatureType">
   <sequence>
    <element name="familyName" type="string"/>
    <element name="age" type="integer"/>
    <element name="nickName" type="string" minOccurs="0"
     maxOccurs="unbounded"/>
    <element name="deanLocation" type="gml:PointPropertyType"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

Dans cet exemple, 'gml:PointPropertyType' est utilisé comme un type de propriété prédéfinie pour une entité de la même manière que le sont les chaînes et les entiers. La déclaration locale de l'élément <deanLocation> définit un alias pour <gml:pointProperty> comme sous-élément de 'Dean'.
L'utilisation exclusive des déclarations globales d'éléments forme un style de rédaction différent qui met en commun tous les éléments dans le même espace de symboles ; ce style nous autorise aussi à affecter des éléments à un groupe de substitution tel que les éléments désignés peuvent se substituer à un élément de tête particulier, qui doit être déclaré comme

élément global. La propriété ‘deanLocation’ doit être déclarée globalement et référencée dans une définition de type comme ceci :

```
<element name="Dean" type="ex:DeanType"
       substitutionGroup="gml:_Feature"/>
<element name="deanLocation" type="gml:PointPropertyType"
 substitutionGroup="gml:pointProperty"/>

<complexType name="DeanType">
 <complexContent>
  <extension base="gml:AbstractFeatureType">
   <sequence>
    <element name="familyName" type="string"/>
    <element name="age" type="integer"/>
    <element name="nickName" type="string" minOccurs="0"
          maxOccurs="unbounded"/>
    <element ref="ex:deanLocation" />
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

4.2.5.5. Codage des collections d’entités

GML 2.0 permet de construire des collections d’entités. Un élément d’un schéma d’application qui joue le rôle de collection d’entités doit être dérivé du type ‘gml:AbstractFeatureCollectionType’ et déclaré pouvant se substituer à l’élément (abstrait) <gml:_FeatureCollection>. Une collection d’entités peut utiliser la propriété ‘featureMember’ pour montrer l’appartenance des autres entités et/ou collections d’entités.

Considérons l’exemple ‘Cambridge’ dans lequel une collection d’entités ‘CityModel’ contient les composants ‘Road’ (Route) et ‘River’ (Rivière). Les entités sont contenues dans ‘CityModel’ en utilisant la propriété générique ‘gml:featureMember’ qui est une instance du type ‘gml:FeatureAssociationType’ :

```
<CityModel fid="Cm1456">
 <dateCreated>Feb 2000</dateCreated>
 <gml:featureMember>
  <River fid="Rv567">....</River>
 </gml:featureMember>

 <gml:featureMember>
  <River fid="Rv568">....</River>
 </gml:featureMember>

 <gml:featureMember>
  <Road fid="Rd812">....</Road>
 </gml:featureMember>
```

```
</CityModel>
```

avec l'extrait du schéma d'application suivant :

```
<element name="CityModel" type="ex:CityModelType"
        substitutionGroup="gml:_FeatureCollection"/>
<element name="River" type="ex:RiverType"
        substitutionGroup="gml:_Feature"/>
<element name="Road" type="ex:RoadType"
        substitutionGroup="gml:_Feature"/>

<complexType name="CityModelType">
  <complexContent>
    <extension base="gml:AbstractFeatureCollectionType">
      <sequence>
        <element name="dateCreated" type="month"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="RiverType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>....</sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="RoadType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>.....</sequence>
    </extension>
  </complexContent>
</complexType>
```

GML 2.0 fournit un mécanisme pour l'identification des entités. Toutes les entités GML ont un attribut optionnel 'fid' de type ID hérité de 'gml:AbstractFeatureType' ; cela signifie que les entités du même document GML ne doivent pas avoir la même valeur de 'fid'. L'attribut 'fid' et les éléments liens construits en utilisant les attributs 'XLink' permettent de référencer sans ambiguïté des éléments spécifiques dans un document GML.

Dans une collection d'entités, un élément <featureMember> peut soit contenir une entité, soit pointer sur une entité stockée à distance (y compris ailleurs dans le même document). Un élément lien peut être construit en intégrant un ensemble spécifique d'attributs 'XLink'. Le langage de liens XML (« XML Linking Language » - XLink) est actuellement une proposition de recommandation du WorldWideWeb Consortium. Xlink permet d'insérer des éléments dans les documents pour créer des liens sophistiqués entre les ressources ; de tels liens peuvent être utilisés pour référencer des propriétés distantes.

Un élément lien peut être utilisé pour mettre en application les pointeurs et cette fonctionnalité a été intégrée dans divers éléments de GML 2.0, y compris 'gml:AssociationAttributeGroup' dans ces spécifications :

- gml:FeatureAssociationType,
- les types collection de géométries et
- tous les types de propriétés géométriques prédéfinis.

Par exemple, nous pouvons modifier l'extrait de 'CityModel' vu ci-dessus pour ajouter des composants 'River' distants en ne faisant aucun changement au schéma de l'application.

```
<CityModel fid="Cm1456">
  <dateCreated>Feb 2000</dateCreated>

  <gml:featureMember xlink:type="simple"
    xlink:href="http://www.myfavoritesite.com/rivers.xml#Rv567"/>

  <gml:featureMember xlink:type="simple"
    xlink:href="http://www.myfavoritesite.com/rivers.xml#Rv568"/>

  <gml:featureMember>
    <Road fid="Rd812">....</Road>
  </gml:featureMember>
</CityModel>
```

Notez que la propriété <featureMember> peut à la fois pointer sur une instance distante et contenir une entité. Il n'est pas possible dans les schémas XML d'empêcher cette pratique en n'utilisant qu'une validation par la grammaire. Les spécifications de GML 2.0 considèrent qu'un 'featureMember' dans cet état est indéfini.

La syntaxe de base pour un lien simple est :

```
<propertyName xlink:type="simple"
xlink:title="Description of target instance"
xlink:href="http://www.myfavoritesite.com/locations.xml#identifier" />
```

L'attribut 'xlink:title' est optionnel. L'attribut 'xlink:href' doit pointer sur un objet dont le type correspond à celui de la valeur de la propriété. Dans ce cas, c'est à l'application de la valider, un analyseur XML ne mettra aucune contrainte sur l'élément lié à l'attribut 'href'.

Pour plus de clarté, un nouvel attribut défini dans l'espace de noms 'gml' est introduit pour compléter les attributs de base des Xlinks : 'remoteSchema'. L'attribut 'remoteSchema' est une référence URI qui utilise la syntaxe des Xpointeurs pour identifier la partie de schéma GML qui contraint la ressource pointée par l'attribut de localisation ; cet attribut supplémentaire est inclus dans 'gml:AssociationAttributeGroup' et est ainsi disponible aux propriétés 'featureMember'. Ainsi il peut être écrit de la manière ci-dessous (supposons que 'RiverType' est la valeur d'un identificateur) :

```
<gml:featureMember xlink:type="simple"
  gml:remoteSchema="http://www.myfavoritesite.com/types.xsd#RiverType"
  xlink:href="http://www.myfavoritesite.com/rivers#Rv567"/>
```

Les attributs 'Xlink' peuvent seulement être placés dans les éléments de propriétés et il n'y a aucune contrainte sur les valeurs de l'attribut 'xlink:title'. Les XLink simples permettent aussi l'utilisation de l'attribut 'xlink:role'. Toutefois, la plupart du temps, c'est une répétition de la propriété (par exemple, le nom de rôle 'featureMember') qui utilise le lien.

Les spécifications 'Xlink' exigent que l'attribut 'xlink:href' pointe sur la ressource du lien en fournissant un « Uniform Resource Identifier » (URI).

4.2.6. Les schémas d'application GML

4.2.6.1. Introduction

Les schémas GML de base (Géométrie, Entités, XLiens) peuvent être vus comme les composants de la structure de base d'une application pour le développement de schémas ou d'ensemble de schémas se rapportant à un domaine particulier , une juridiction particulière ou une communauté d'information. De plus, de tels schémas d'applications peuvent être développés de manière plus horizontale pour plusieurs communautés d'information.

Il existe quelques obligations de conformité que tout schéma d'application doit satisfaire. En particulier, un schéma d'application GML conforme doit tenir compte des obligations générales suivantes :

1. un schéma d'application ne doit pas changer le nom, la définition ou les types de données des éléments obligatoires de GML,
2. les définitions de type abstrait peuvent être librement étendues ou restreintes,
3. un schéma d'application doit être disponible pour quiconque reçoit de l'information structurée selon ce schéma,
4. les schémas pertinents doivent être dans un espace de noms qui ne doit pas être http://www.opengis.net/gml (c'est-à-dire l'espace de noms 'gml').

Les schémas GML fournissent les constructions de base pour le traitement des données géospatiales de manière modulaire. Une architecture d'application plus spécialisée contenant des schémas d'applications sera créée pour un thème ou un domaine particulier mais pourra aussi être totalement horizontale.

4.2.6.2. Règles pour la construction de schémas d'application

Il doit être tenu compte des règles suivantes lors de la construction des schémas d'application XML.

Définition de nouveaux types d'entités

Les développeurs de schémas d'application peuvent créer leurs propres types d'entités ou de collections d'entités mais ils doivent faire en sorte que les types d'entités concrètes et les types de collection d'entités soient des sous-types (soit directement, soit indirectement) des types abstraits GML correspondants : 'gml:AbstractFeatureType' ou 'gml:AbstractFeatureCollectionType'.

```
<complexType name="MyFeature1Type">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <!-- additional child elements inserted here -->
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="MyFeature2Type">
  <complexContent>
    <extension base="foo:MyFeature1Type">
      <sequence>
        <!-- additional child elements inserted here -->
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Définition de nouveaux types géométriques

Les auteurs peuvent créer leurs propres types géométriques si la construction désirée n'est pas dans GML. Pour cela, les auteurs doivent faire en sorte que les types de géométries concrètes et les types de collection de géométrie soient des sous-types (soit directement, soit indirectement) des types abstraits GML correspondants : 'gml:AbstractGeometryType' ou 'gml:AbstractGeometryCollectionType'.

```
<complexType name="MyGeometry1Type">
  <complexContent>
    <extension base="gml:AbstractGeometryType">
      <sequence>
        <!-- additional child elements inserted here -->
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Tout sous-type de géométrie défini par l'utilisateur héritera des éléments et des attributs des types de géométrie GML de base, sans restriction, mais peut étendre ces types de base selon les besoins de l'application, en fournissant par exemple un plus grand degré d'interopérabilité avec les systèmes d'héritage et les ensembles de données.

Définition de nouvelles propriétés géométriques

Les auteurs peuvent créer leurs propres propriétés géométriques qui contiennent les types géométriques qu'ils ont définis; ils doivent faire en sorte que ces propriétés soient des sous-types (soit directement, soit indirectement) de 'gml:GeometryPropertyType' et qu'elles ne changent pas la cardinalité de l'instance cible, qui doit être une vraie construction géométrique.

```
<complexType name="MyGeometry1PropertyType">
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="foo:MyGeometry1Type" />
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>
```

Déclaration d'un espace de noms

Les auteurs doivent déclarer un espace de noms pour leurs schémas. Tous les éléments déclarés dans le schéma, accompagné de leurs définitions de type, seront dans cet espace de noms. La validation ne réussira pas si le document instance n'est pas dans l'espace de noms du schéma. Notez qu'il n'est pas obligatoire que les URI pointent sur une adresse réelle, comme un schéma de document ; les espace de noms sont juste un moyen pour avoir des noms d'éléments distincts et éviter ainsi les 'collisions' d'espaces de noms.

Pour utiliser les espaces de noms, les éléments sont décrits par des noms complets ('qualified name' = QName) qui comportent deux parties : le *préfixe* qui correspond à une référence URI et indique l'espace de noms auquel l'élément appartient ; la *partie locale* conforme aux règles de désignation 'NCName' de recommandation des espaces de noms du W3C.

```
NCName ::=  (Letter | '_') (NCNameChar)*
NCNameChar ::=  Letter | Digit | '.' | '-' | '_' | CombiningChar | Extender
```

Dans chaque exemple, les espaces de noms pour tous les éléments sont indiqués explicitement pour montrer comment le vocabulaire de différents espaces de noms se mélange dans un document instance. L'utilisation des noms complets est précisée par la valeur 'qualified' de l'attribut 'elementFormDefault' de <schema> :

```
<schema  targetNamespace="http://www.bar.net/foo"
        xmlns="http://www.w3.org/1999/XMLSchema"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns:foo="http://www.bar.net/foo"
        elementFormDefault="qualified"
```

```
      version="0.1">

<!-- import constructs from the GML Feature and Geometry schemas -->
<import namespace="http://www.opengis.net/gml" schemaLocation="feature.xsd"/>
```

Importation de schémas

Un document instance conforme peut utiliser des concepts de plusieurs espaces de noms de la manière indiquée sur la Figure 4.2. Le schéma-A dans l'espace de noms 'foo' connecte la structure GML via le schéma des entités (qui apporte aussi la géométrie). Le schéma des entités est dans l'espace de noms 'gml' avec le schéma de la géométrie, et utilise donc le mécanisme d'inclusion ('include'). Par contre, comme le schéma-A est un espace de noms différent, il doit utiliser le mécanisme d'importation ('import') pour utiliser les concepts de base de GML.

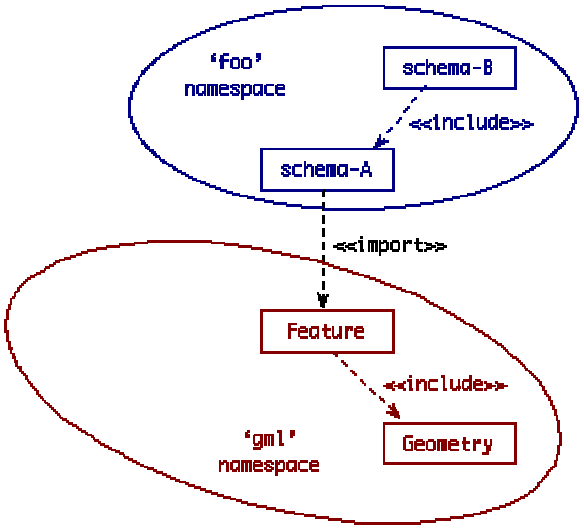


Figure 4.2 : Utilisation de schémas de plusieurs espaces de noms

Les auteurs ne doivent pas oublier que le mécanisme d'importation ('import') est le seul moyen par lequel un schéma d'application peut accéder aux concepts GML et les utiliser. Puisque les schémas d'application conformes ne peuvent pas utiliser l'espace de noms GML de base, les autres mécanismes comme 'include' et 'redefine' ne peuvent pas être employés.

Utilisation de groupes de substitution

Les éléments globaux qui définissent des groupes de substitution doivent être utilisés pour les classes (par exemple entité, géométrie, etc.) et les propriétés qui sont définies dans les schémas d'application GML et peuvent se substituer aux éléments GML correspondants là où ils sont attendus. Il faut déclarer un élément globalement et spécifier un groupe de substitution adéquat si l'élément a besoin de se substituer à un autre (éventuellement abstrait) ; les groupes de substitution permettent la 'promotion' des types (c'est-à-dire traiter un type spécifique comme un super-type plus général). La déclaration globale suivante

garantit que si ‘foo:CircleType’ est un type géométrique défini, alors un élément <Circle> peut apparaître partout où l’élément (abstrait) ‘gml:_Geometry’ est attendu :

```
<schema . . .>
  <element name="Circle" type="foo:CircleType"
         substitutionGroup="gml:_Geometry" />
  . . .
</schema>
```

Des éléments identiques déclarés dans plus d’une définition de type complexe dans un schéma doivent référencer un élément global. Si l’élément <Circle> est déclaré globalement dans l’espace de noms ‘foo’ (comme vu ci-dessus), il est référencé dans une définition de type comme ceci :

```
<complexType name="MyHubPropertyType">
  <complexContent>
    <restriction base="gml:GeometryPropertyType">
      <sequence minOccurs="0">
        <element ref="foo:Circle" />
      </sequence>
      <attributeGroup ref="gml:AssociationAttributeGroup"/>
    </restriction>
  </complexContent>
</complexType>
```

4.3. SVG

4.3.1. L'explication de la dénomination : SVG

SVG correspond aux initiales de « Scalable Vector Graphics », une grammaire XML pour des graphiques stylisables.

Adaptable [Scalable]

Adaptable veut dire « qu'on peut augmenter ou diminuer uniformément ». En termes graphiques, adaptable signifie « qui ne se limite pas à une seule dimension fixe en pixels ». Pour le Web, adaptable signifie qu'une technologie donnée peut s'étendre à un grand nombre de fichiers, un grand nombre d'utilisateurs, une grande variété d'applications. SVG, qui est une technologie concernant les graphiques pour le Web, est adaptable dans les deux sens du mot.
Les graphiques SVG s'adaptent aux différentes résolutions d'affichage.
Les graphiques SVG sont adaptables parce qu'un même contenu SVG peut être un graphique autonome, être référencé ou être inclus dans d'autres graphiques SVG, ce qui permet ainsi la construction d'une illustration complexe par tranche, éventuellement par plusieurs personnes.

Vectoriel [Vector]

Les graphiques vectoriels contiennent des objets géométriques tels que des droites ou des courbes. Ceci permet une plus grande souplesse en comparaison avec des formats seulement pixélisés (comme PNG et JPEG) qui doivent conserver l'information de chaque pixel du graphique

Graphiques

La plupart des grammaires XML existantes représentent soit une information textuelle, soit des données brutes telles que des informations financières. Ces grammaires n'offrent en général que des possibilités graphiques rudimentaires, souvent moindres que l'élément HTML 'img'. SVG comble une lacune dans cet univers en offrant une description riche et structurée des graphiques vectoriels et des graphiques mixtes vectoriels/pixélisés .

Stylisable

Les avantages des feuilles de style, en termes de contrôle de la présentation, de souplesse, de chargement plus rapide et de facilité de maintenance, sont maintenant bien admis, plus sûrement pour du texte. SVG étend ce contrôle au domaine des graphiques.
On qualifie souvent la combinaison des scripts, du DOM et de CSS de « HTML dynamique » et on utilise couramment une telle combinaison pour des effets d'animation, d'interactivité et de présentation. SVG permet une même manipulation, à partir de scripts, de l'arbre du document et de la feuille de style.

4.3.2. Les principaux concepts SVG

4.3.2.1. Les objets graphiques

SVG propose un élément général 'path', qui peut être utilisé pour créer une grande variété d'objets graphiques et aussi pour offrir des formes de base courantes, comme les rectangles et les ellipses. Celles-ci, pratiques pour un codage manuel, peuvent être employées tout comme l'élément 'path', plus général. SVG offre un contrôle fin sur le système de coordonnées dans lequel les objets graphiques sont définis et sur les transformations qui seront appliquées lors du rendu.

4.3.2.2. Les types des éléments graphiques

SVG reconnaît 3 types fondamentaux d'éléments graphiques pouvant être rendus sur un canevas :

- Les formes, qui représentent une certaine combinaison de droites et de courbes ;
- Le texte, qui représente une certaine combinaison de glyphes de caractères ;
- Les images pixélisées, qui représentent un réseau de valeurs spécifiant la couleur de peinture et l'opacité (souvent nommée couche alpha) comme une série de points sur un quadrillage rectangulaire.

4.3.2.3. L'animation

On peut produire une animation via la manipulation du document à partir de scripts, mais les scripts sont difficiles à écrire et leur transposition entre les outils d'édition est plus compliquée. En réponse encore une fois à la communauté des concepteurs, SVG incorpore des éléments d'animation déclaratifs qui ont été conçus en collaboration par les Groupes de Travail SVG et SYMM. Ceci permet une expression en SVG des effets animés courants dans les graphiques Web existants

4.3.3. La structure du document

4.3.3.1. La définition d'un fragment de document SVG : l'élément ‘svg’[5]

Un fragment de document SVG consiste en un nombre quelconque d'éléments SVG contenu dans un élément ‘svg’.
Un fragment de document SVG peut consister en un fragment vide (i.e., l'élément ‘svg’ n'a pas de contenu), ou encore en un fragment de document SVG très simple, avec un seul élément graphique SVG, tel un élément ‘rect’, et aller jusqu'à une collection complexe et profondément imbriquée d'éléments conteneurs et d'éléments graphiques.

[5] Les exemples et figures proviennent du site : http://www.w3.org/TR/2001/REC-SVG-20010904/

Un fragment de document SVG peut exister de lui-même comme fichier auto-contenu ou ressource, auquel cas le fragment de document SVG est un document SVG, ou peut être incorporé en ligne comme fragment dans un document XML parent.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="4cm"
     xmlns="http://www.w3.org/2000/svg">

    <rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
    <rect x="0.5cm" y="2cm" width="1cm" height="1.5cm"/>
    <rect x="3cm" y="0.5cm" width="1.5cm" height="2cm"/>
    <rect x="3.5cm" y="3cm" width="1cm" height="0.5cm"/>
 </svg>
```

Dans tous les cas, on doit fournir une déclaration d'espace de nommage pour SVG, ainsi tous les éléments SVG sont identifiés comme appartenant à l'espace de nommage SVG. La fourniture de la déclaration de l'espace de nommage peut être effectuée des manières suivantes. Un attribut xmlns, sans préfixe d'espace de nommage, pourrait être spécifié pour un élément ‘svg’, ceci signifie que SVG est l'espace de nommage par défaut pour tous les éléments dans le rayon d'action de l'élément avec l'attribut xmlns :

```
<svg xmlns="http://www.w3.org/2000/svg"...>
  <rect .../>
</svg>
```

Si on spécifie un préfixe d'espace de nommage sur l'attribut xmlns (par exemple, xmlns :svg ="http://www.w3.org/2000/svg"), alors l'espace de nommage correspondant n'est pas celui par défaut, c'est pourquoi on doit assigner un préfixe d'espace de nommage explicite aux éléments :

```
<svg:svg xmlns:svg="http://www.w3.org/2000/svg"...>
  <svg:rect .../>
</svg:svg>
```

4.3.3.2. Le regroupement : l'élément ‘g’

L'élément ‘g’ est un élément conteneur pour le regroupement d'éléments graphiques reliés.
Le regroupement des constructions, quand il est utilisé conjointement avec les éléments ‘desc’ et ‘title’, fournit une information au sujet de la structure et de la sémantique du document. Les documents de structure riche peuvent être rendus visuellement, en parole ou en braille, favorisant ainsi l'accessibilité.
Un groupe d'éléments, tout comme des objets individuels, peuvent recevoir une identité au moyen de l'attribut id. Les groupes nommés sont nécessaires pour plusieurs finalités telles que l'animation et les objets ré-utilisables.
Un exemple :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
```

```
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="5cm" height="5cm"
    xmlns="http://www.w3.org/2000/svg">
  <desc>Deux groupes, chacun avec deux rectangles
  </desc>
  <g id="group1" fill="red" >
    <rect x="1cm" y="1cm" width="1cm" height="1cm" />
    <rect x="3cm" y="1cm" width="1cm" height="1cm" />
  </g>
  <g id="group2" fill="blue" >
    <rect x="1cm" y="3cm" width="1cm" height="1cm" />
    <rect x="3cm" y="3cm" width="1cm" height="1cm" />
  </g>

  <!-- Montre le contour du canevas en utilisant un élément 'rect' -->
  <rect x=".01cm" y=".01cm" width="4.98cm" height="4.98cm"
       fill="none" stroke="blue" stroke-width=".02cm" />
</svg>
```

Un élément 'g' peut contenir d'autres éléments 'g' imbriqués dans celui-ci, à une profondeur arbitraire. Ainsi, il est possible d'avoir ceci :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="3in"
    xmlns="http://www.w3.org/2000/svg">
  <desc>Les groupes peuvent s'imbriquer
  </desc>
  <g>
    <g>
      <g>
      </g>
    </g>
   </g>
</svg>
```

Tout élément, qui n'est pas contenu dans un élément 'g', est traité (au moins conceptuellement) comme s'il était dans son propre groupe.

4.3.3.3. Les références et l'élément ‘defs’

SVG fait un usage étendu des références d'URI vers d'autres objets. Par exemple, pour remplir un rectangle avec un dégradé linéaire, vous définissez d'abord un élément ‘linearGradient’ et vous lui donnez une ID, comme dans :

```
<linearGradient id="MonDegrade">...</linearGradient>
```

Ensuite vous appelez le dégradé linéaire au travers de la valeur de la propriété 'fill' pour le rectangle, comme dans :

<rect style="fill:url(#MonDegrade)"/>

SVG reconnaît deux types de référence d'URI : les références d'URI locales, celles-ci, ne contenant pas de valeur <URI-absolu> ou <URI-relatif>, ne contiennent ainsi qu'un identifiant de fragment (i.e., une valeur #<ID-élément> ou #xpointer(id<ID-élément>)) ; les références d'URI non locales, celles-ci contiennent une valeur <URI-absolu> ou <URI-relatif>.

L'élément 'defs' est un conteneur pour des éléments référencés. Pour des raisons de compréhension et d'accessibilité, on recommande, autant que possible, que les éléments référencés soient définis dans un élément 'defs'.
Le modèle de contenu pour 'defs' est le même que pour l'élément 'g' ; ainsi, tout élément qui peut être l'enfant d'un élément 'g' peut aussi être l'enfant d'un élément 'defs' et vice versa.
Par exemple :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="8cm" height="3cm"
    xmlns="http://www.w3.org/2000/svg">
 <desc>Des références d'URI locales dans l'ancêtre de l'élément 'defs'.</desc>
 <defs>
  <degradeLineaire id="Gradient01">
   <stop offset="20%" stop-color="#39F" />
   <stop offset="90%" stop-color="#F3F" />
  </degradeLineaire>
 </defs>
 <rect x="1cm" y="1cm" width="6cm" height="1cm"
      fill="url(#Gradient01)"  />

 <!-- Montre le contour du canevas en utilisant un élément 'rect' -->
 <rect x=".01cm" y=".01cm" width="7.98cm" height="2.98cm"
      fill="none" stroke="blue" stroke-width=".02cm" />
</svg>
```

4.3.3.4. L'élément 'symbol'

L'élément 'symbol' est utilisé pour définir des objets gabarits graphiques qui peuvent être instanciés par un élément 'use'.
L'emploi d'éléments 'symbol' pour des graphiques, ré-utilisés plusieurs fois dans le même document, ajoute de la structure et du sens. Les documents de structure riche peuvent se rendre visuellement, en parole ou en braille et, ainsi, favoriser l'accessibilité.
Les distinctions clés entre les éléments 'symbol' et 'g' sont :

- Un élément 'symbol' lui-même n'est pas rendu. Seules sont rendues les instances d'un élément 'symbol' (i.e., la référence vers un objet 'symbol' par un élément 'use') ;

- Un élément ‘symbol’ a des attributs viewBox et preserveAspectRatio qui lui permettent de s'adapter pour tenir dans une zone de visualisation rectangulaire définie par l'élément ‘use’ appelant.

L'élément ‘symbol’ n'est jamais rendu directement ; son seul usage consiste à être appelé par un élément ‘use’. La propriété ‘display’ ne s'applique pas à l'élément ‘symbol’ ; ainsi, l'élément ‘symbol’ n'est jamais rendu directement, même si la propriété ‘display’ est paramétrée à une autre valeur que none et il peut toujours être appelé, même si la valeur de la propriété ‘display’ de l'élément ‘symbol’, ou de l'un de ses ancêtres, est none.

4.3.3.5. L'élément ‘use’

Tout ‘svg’, ‘symbol’, ‘g’, éléments graphiques ou autre ‘use’ est un objet gabarit potentiel qui peut être ré-utilisé, ou « instancé », dans le document SVG via un élément ‘use’. L'élément ‘use’ appelle un autre élément et indique, à un point donné dans le document, où le contenu graphique de cet élément doit être inclus/dessiné.
À la différence de l'élément ‘image’, l'élément ‘use’ ne peut appeler de fichiers entiers.
L'élément ‘use’ dispose des attributs optionnels x, y, width et height, qui sont utilisés pour la disposition du contenu graphique de l'élément appelé dans une région rectangulaire dans le système de coordonnées courant.
L’exemple Use01,ci-dessous, montre l'action simple d'un élément ‘use’ sur un élément ‘rect’.

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Exemple Use01 - Cas simple d'un 'use' sur un 'rect'</desc>
  <defs>
    <rect id="MonRect" width="60" height="10"/>
  </defs>
  <rect x=".1" y=".1" width="99.8" height="29.8"
            fill="none" stroke="blue" stroke-width=".2" />
  <use x="20" y="10" xlink:href="#MonRect" />
</svg>
```

L'effet visuel équivaudrait à celui du document suivant :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="10cm" height="3cm" viewBox="0 0 100 30"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Exemple Use01-ContenuGénéré - Cas simple d'un 'use' sur un 'rect'</desc>
  <!-- la section 'defs' est laissée de côté -->

  <rect x=".1" y=".1" width="99.8" height="29.8"
        fill="none" stroke="blue" stroke-width=".2" />

  <!-- Début du contenu généré. Remplacement de 'use' -->
```

```
  <g>
    <rect width="60" height="10"/>
  </g>
  <!-- Fin du contenu généré -->

</svg>
```

4.3.3.6. L'élément ‘image’

L'élément ‘image’ indique que le contenu d'un fichier entier doit être rendu dans un rectangle donné dans le système de coordonnée utilisateur courant.
Quand un élément ‘image’ appelle un fichier d'image pixélisée, tel que PNG ou JPEG, alors l'image pixélisée est enchâssée dans la région spécifié par les attributs x, y, width et height.
L'attribut preserveAspectRatio détermine à la fois la taille et le ratio d'aspect de l'image, quand elle est enchâssée dans la région spécifiée par les attributs x, y, width et height.
Un exemple :

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4in" height="3in"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Ce graphique est relié à une image externe
  </desc>
  <image x="200" y="200" width="100px" height="100px"
         xlink:href="monimage.png">
    <title>Mon image</title>
  </image>
</svg>
```

4.3.4. Les propriétés de style de SVG

SVG utilise des propriétés de style pour décrire de nombreux paramètres de document. Les propriétés de style définissent comment les éléments graphiques du contenu SVG doivent être rendus. SVG utilise des propriétés de style pour ce qui suit :
Les paramètres qui ont une nature essentiellement visuelle et qui donc se prêtent au style. Des exemples comprennent tous les attributs qui définissent la manière dont un objet est « peint », comme les couleurs de remplissage et de trait, les épaisseurs de ligne et les pointillés ;
Les paramètres impliqués dans le style du texte, comme ‘font-family’ et ‘font-size’ ;
Les paramètres qui influent sur la façon dont les éléments graphiques sont rendus, comme la spécification des tracés de rognage, les masques, les pointes de flèche, les marqueurs et les effets de filtre.

4.3.5. Les systèmes de coordonnées et les transformations

4.3.5.1. Introduction

Le canevas SVG décrit « l'espace où le contenu SVG est rendu ». Le canevas est infini dans chaque dimension de l'espace, mais le rendu s'effectue par rapport à une région rectangulaire finie du canevas. Cet région rectangulaire finie est appelée la zone de visualisation SVG.
En SVG, on peut spécifier les longueurs comme :

- (si aucun identifiant d'unité est fourni) valeurs dans l'espace utilisateur, par exemple, "15" ;
- (si un identifiant d'unité est fourni) une longueur exprimée comme une unité de mesure absolue ou relative, par exemple, "15mm" ou "5em".

Les identifiants d'unités de longueur reconnus sont : em, ex, px, pt, pc, cm, mm, in et les pourcentages.
Un nouvel espace utilisateur (i.e., un nouveau système de coordonnées courant) peut être établi à tout endroit dans un fragment de document SVG en spécifiant des transformations prenant la forme de matrices de transformation ou d'opérations de transformation simples telles que rotations, inclinaisons, mises à l'échelle et translations. L'établissement de nouveaux espaces utilisateurs via des transformations du système de coordonnées est une opération fondamentale des graphiques en 2D et représente la méthode habituelle pour le contrôle de la taille, la position, la rotation et l'inclinaison des objets graphiques.

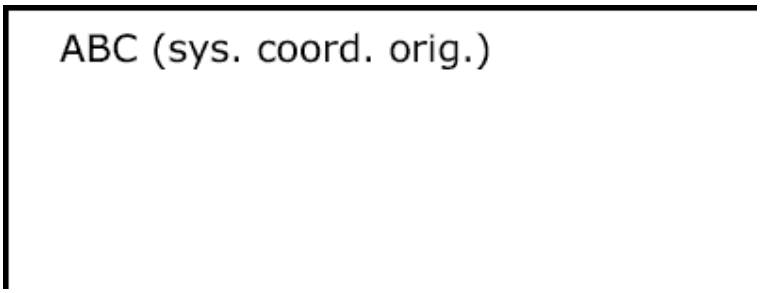
4.3.5.2. Les transformations du système de coordonnées

On peut établir un nouvel espace utilisateur (i.e., un nouveau système de coordonnées courant) en spécifiant des transformations, sous la forme d'un attribut transform sur un élément conteneur ou un élément graphique, ou d'un attribut viewBox sur un élément ‘svg’, ‘symbol’, ‘marker’, ‘pattern’ ou ‘view’ Les attributs transform et viewBox transforme les coordonnées de l'espace utilisateur et les longueurs des attributs frères sur l'élément donné et sur tous ses descendants. Les transformations peuvent être imbriquées, auquel cas l'effet des transformations est cumulatif.
L'exemple ci-dessous montre un document sans transformation. La chaîne textuelle est spécifiée dans le système de coordonnées initial.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="150px"
    xmlns="http://www.w3.org/2000/svg">
 <desc>Exemple SysCoordOrig - Transformations simples : image originale</desc>
 <g fill="none" stroke="black" stroke-width="3" >
        <!-- Dessine les axes du système de coordonnées original -->
        <line x1="0" y1="1.5" x2="400" y2="1.5" />
         <line x1="1.5" y1="0" x2="1.5" y2="150" />
 </g>
 <g>
  <text x="30" y="30" font-size="20" font-family="Verdana" >
    ABC (syst. coord. orig.)
  </text>
```

```
  </g>
</svg>
```

Et voici la représentation en SVG de cette exemple :

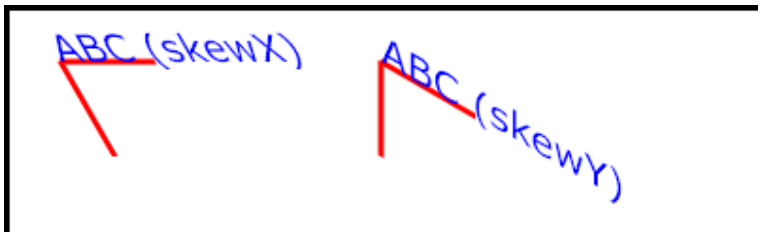


L'exemple ci-dessous définit deux systèmes de coordonnées qui sont **inclinés** par rapport au système de coordonnées d'origine.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400px" height="120px"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Exemple Inclinaison - Montre les effets de skewX et skewY</desc>
  <g fill="none" stroke="black" stroke-width="3" >
    <!-- Dessine les axes du système de coordonnées original -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="120" />
  </g>
  <!-- Établit un nouveau système de coordonnées, dont l'origine est (30,30)
       dans le syst. coord. initial et qui est incliné de 30 degrés sur X. -->
  <g transform="translate(30,30)">
    <g transform="skewX(30)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (skewX)
      </text>
    </g>
  </g>
```

```
  <!-- Établit un nouveau système de coordonnées, dont l'origine est (200,30)
       dans le syst. coord. initial et qui est incliné de 30 degrés sur Y. -->
  <g transform="translate(200,30)">
    <g transform="skewY(30)">
      <g fill="none" stroke="red" stroke-width="3" >
        <line x1="0" y1="0" x2="50" y2="0" />
        <line x1="0" y1="0" x2="0" y2="50" />
      </g>
      <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue" >
        ABC (skewY)
      </text>
    </g>
  </g>
</svg>
```

L'affichage en SVG donne :



4.3.6. Les tracés

4.3.6.1. Introduction

Un tracé représente le contour d'une forme, qui peut avoir un remplissage, ou un trait.
Un tracé se décrit en faisant appel au concept d'un point courant. Par analogie avec un dessin sur du papier, le point courant peut être assimilé à la position du crayon.
Les tracés représentent la géométrie du contour d'un objet, définis selon les instructions des éléments moveto (établit un nouveau point courant), lineto (dessine une droite), curveto (dessine une courbe à l'aide d'une courbe de Bézier cubique), arc (un arc circulaire ou elliptique) et closepath (clôt la forme courante en dessinant une ligne jusqu'au dernier moveto). Il est possible d'avoir des tracés composés (i.e., un tracé avec plusieurs sous-tracés) qui permettent des effets comme un « trou de donut » dans des objets.
On définit un tracé SVG avec l'élément 'path'.

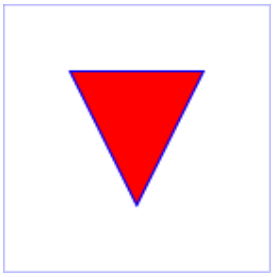
4.3.6.2. Les données de tracé

On définit un tracé avec un élément 'path' ayant un attribut d = "<données-de-tracé>", où d contient les commandes moveto, line, curve, arc, et closepath.
L'exemple triangle01 spécifie le tracé d'une forme triangulaire. (Le **M** indique une commande moveto, les **L** des commandes lineto et le **z** une commande closepath).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
```

```
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
    xmlns="http://www.w3.org/2000/svg">
  <title>Exemple triangle01- exemple simple d'un 'path'</title>
  <desc>Un tracé dessinant un triangle</desc>
  <rect x="1" y="1" width="398" height="398"
        fill="none" stroke="blue" />
  <path d="M 100 100 L 300 100 L 200 300 z"
        fill="red" stroke="blue" stroke-width="3" />
</svg>
```

L'affichage en SVG donne :



4.3.7. Les formes de base

4.3.7.1. Introduction

SVG contient le jeu des éléments de forme de base suivant :

- des rectangles (dont ceux avec des coins arrondis optionnels) ;
- des cercles ;
- des ellipses ;
- des lignes ;
- des polylignes;
- des polygones.

Mathématiquement, ces éléments de forme sont équivalents à un élément 'path' qui construirait la même forme.

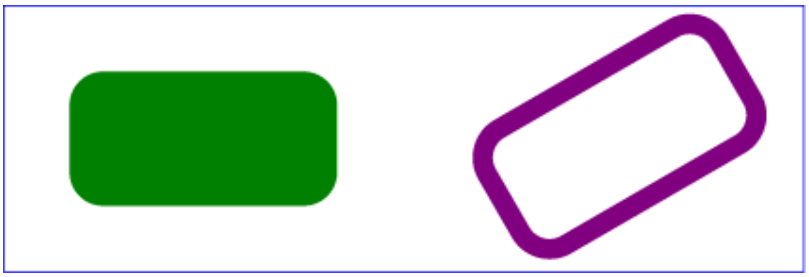
4.3.7.2. L'élément 'rect'

L'élément 'rect' définit un rectangle qui s'aligne selon l'axe du système de coordonnées utilisateur. On peut obtenir des rectangles avec des coins arrondis en donnant les valeurs appropriées aux attributs rx et ry.
L'exemple rect01 montre deux rectangles arrondis. L'attribut rx spécifie l'arrondi des coins des rectangles. Remarquer que, comme aucune valeur n'est spécifiée pour l'attribut ry, celui-ci prendra la même valeur que l'attribut rx.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
```

```
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
    xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Exemple rect01 - rectangles arrondis</desc>

  <!-- Montre le contour du canevas en utilisant un élément 'rect' -->
  <rect x="1" y="1" width="1198" height="398"
       fill="none" stroke="blue" stroke-width="2"/>

  <rect x="100" y="100" width="400" height="200" rx="50"
       fill="green" />

  <g transform="translate(700 210) rotate(-30)">
    <rect x="0" y="0" width="400" height="200" rx="50"
         fill="none" stroke="purple" stroke-width="30" />
  </g>
</svg>
```

L'affichage donne :

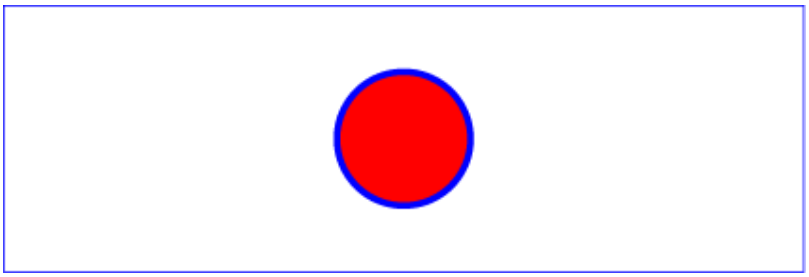


4.3.7.3. L'élément 'circle'

L'élément 'cicle' définit un cercle à partir d'un point central et d'un rayon.
L'exemple circle01 consiste en un élément 'circle' qui est rempli en rouge avec un contour bleu.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
    xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Exemple circle01 - cercle rempli de rouge avec un liseré bleu</desc>

  <!-- Montre le contour du canevas avec un élément 'rect' -->
  <rect x="1" y="1" width="1198" height="398"
       fill="none" stroke="blue" stroke-width="2"/>

  <circle cx="600" cy="200" r="100"
       fill="red" stroke="blue" stroke-width="10"  />
</svg>
```

L’affichage donne :

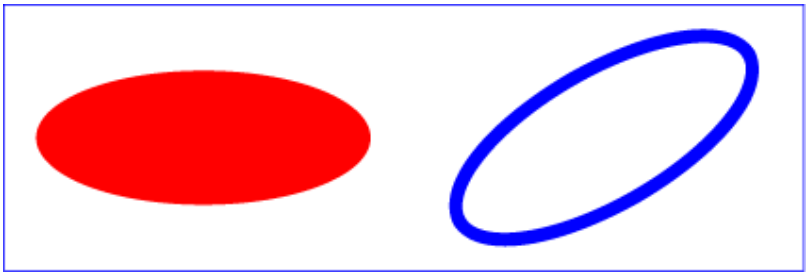


4.3.7.4. L'élément ‘ellipse’

L'élément ‘ellipse’ définit une ellipse, alignée sur les axes du système de coordonnées utilisateur et basée sur un point central et deux rayons.
L'exemple ellipse01 ci-dessous spécifie les coordonnées de deux ellipses dans le système de coordonnées utilisateur établi par l'attribut viewBox de l'élément ‘svg’ et par l'attribut transform des éléments ‘g’et ‘ellipse’. Les deux ellipses utilisent les valeurs par défaut de zéro des attributs cx et cy (le centre de l'ellipse). La seconde ellipse est tournée.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Exemple ellipse01 - exemples d'ellipses</desc>

  <!-- Montre le contour du canevas avec un élément 'rect' -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />

  <g transform="translate(300 200)">
    <ellipse rx="250" ry="100"
          fill="red"  />
  </g>

  <ellipse transform="translate(900 200) rotate(-30)"
        rx="250" ry="100"
        fill="none" stroke="blue" stroke-width="20"  />

</svg>
```

L’affichage donne :

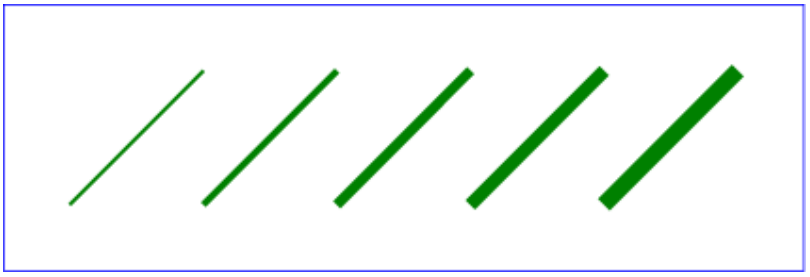


4.3.7.5. L'élément ‘line’

L'élément ‘line’ définit un segment de droite qui commence à un point et finit à un autre. L'exemple line01 ci-dessous spécifie les coordonnées de cinq segments de droite dans le système de coordonnées utilisateur établi par l'attribut viewBox de l'élément ‘svg’. Les segments de droite ont différentes épaisseurs.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Exemple line01 - droites exprimées en coordonnées utilisateur</desc>

  <!-- Montre le contour du canevas avec un élément 'rect' -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2" />

  <g stroke="green" >
    <line x1="100" y1="300" x2="300" y2="100"
          stroke-width="5"  />
    <line x1="300" y1="300" x2="500" y2="100"
          stroke-width="10"  />
    <line x1="500" y1="300" x2="700" y2="100"
          stroke-width="15"  />
    <line x1="700" y1="300" x2="900" y2="100"
          stroke-width="20"  />
    <line x1="900" y1="300" x2="1100" y2="100"
          stroke-width="25"  />
  </g>
</svg>
```

L’affichage donne :

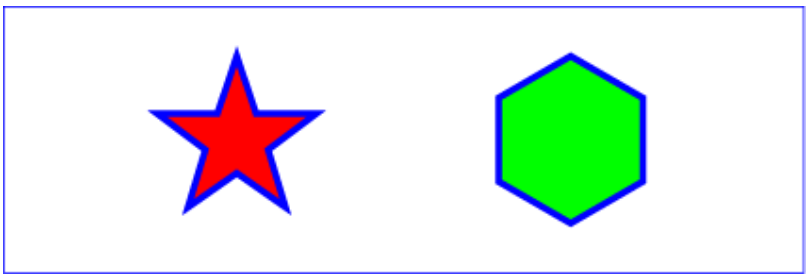


4.3.7.6. L'élément ‘polygon’

L'élément ‘polygon’ définit une forme fermée consistant en un jeu de segments de droite reliés.
L'exemple polygon01 ci-dessous spécifie deux polygones (une étoile et un hexagone) dans le système de coordonnées utilisateur établi par l'attribut viewBox de l'élément ‘svg’.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
    xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Exemple polygon01 - étoile et hexagone</desc>

  <!-- Montre le contour du canevas avec un element 'rect' -->
  <rect x="1" y="1" width="1198" height="398"
       fill="none" stroke="blue" stroke-width="2" />

  <polygon fill="red" stroke="blue" stroke-width="10"
          points="350,75  379,161 469,161 397,215
                  423,301 350,250 277,301 303,215
                  231,161 321,161" />
  <polygon fill="lime" stroke="blue" stroke-width="10"
          points="850,75  958,137.5 958,262.5
                  850,325 742,262.6 742,137.5" />
</svg>
```

L’affichage donne :



4.4. XSLT

4.4.1. Introduction

XSLT (eXtended Stylesheet Language Transformation) est un langage permettant de spécifier des règles de transformations sur un document XML dans le but de générer un document cible dérivé du document source XML
Une transformation exprimée en XSLT décrit les règles pour transformer un arbre source en un arbre résultat. La transformation est obtenue en associant des motifs à des modèles. Les motifs sont appliqués aux éléments de l'arbre source. Le modèle est instancié pour créer une partie de l'arbre résultat. L'arbre résultat est distinct de l'arbre source. Le document cible peut être d'un type quelconque; toutefois le langage distingue trois sortes de documents susceptibles d'être générés par des règles de transformations exprimées en XSLT: des documents XML, des documents HTML et tout autre types de documents.
Une transformation exprimée en XSLT est appelée feuille de styles. Une feuille de styles contient un ensemble de règles modèles. Une règle modèle est constituée de deux parties : un motif qui sert à identifier des nœuds de l'arbre source et un modèle pouvant être instancié afin de constituer une partie de l'arbre résultat.
XSLT utilise le langage d'expression défini par XPath pour sélectionner les éléments à traiter, les traitements conditionnels et la génération de texte.

4.4.2. Structure des feuilles de style

4.4.2.1. Espace de noms XSLT

L'URI de l'espace de noms XSLT est : http://www.w3.org/1999/XSL/Transform.

Les processeurs XSLT doivent utiliser le mécanisme des espaces de noms pour reconnaître les éléments et les attributs de cet espace de noms. Les éléments de l'espace de noms XSLT sont reconnus uniquement dans la feuille de styles et non dans le document source.
On utilise le préfixe xsl: pour référencer les éléments de l'espace de noms XSLT. Cependant, les feuilles de style XSLT sont libres d'utiliser n'importe quel préfixe, à condition qu'il y ait une déclaration d'espace de noms liant le préfixe à l'URI de l'espace de noms XSLT.

4.4.2.2. L'élément feuille de styles

```
<xsl:stylesheet id = id   version = number>
  <!-- Contenu : (xsl:import*, top-level-elements)-->
</xsl:stylesheet>
```

Une feuille de styles est représentée par l'élément xsl :stylesheet dans un document XML.
Xsl :transform est autorisé comme synonyme de xsl :stylesheet.
L'élément xsl :stylesheet doit avoir un attribut `version` indiquant la version de XSLT exigées par la feuille de styles.
Un élément fils de l'élément xsl :stylesheet est appelé élément de haut niveau.
L'exemple suivant montre la structure d'une feuille de styles. Les points de suspension (...) indiquent où les valeurs et les contenus d'attributs ont été omis. Bien que cet exemple montre un cas d'utilisation de chacun des éléments autorisés, les feuilles de style peuvent en contenir de zéro à plusieurs occurrences.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="..."/>
<xsl:include href="..."/>
<xsl:strip-space elements="..."/>
<xsl:preserve-space elements="..."/>
<xsl:output method="..."/>
<xsl:key name="..." match="..." use="..."/>
<xsl:decimal-format name="..."/>
<xsl:namespace-alias stylesheet-prefix="..." result-prefix="..."/>
<xsl:attribute-set name="...">
   ...
</xsl:attribute-set>
<xsl:variable name="...">...</xsl:variable>
<xsl:param name="...">...</xsl:param>
<xsl:template match="...">
   ...
</xsl:template>
<xsl:template name="...">
   ...
</xsl:template>
</xsl:stylesheet>
```

L'ordre avec lequel les fils de l'élément xsl :stylesheet apparaissent n'est significatif que pour les éléments xsl:import et les récupérations d'erreurs.
De plus, l'élément xsl :stylesheet peut contenir n'importe quel élément n'appartenant pas à l'espace de noms XSLT, à la condition que l'URI de l'espace de noms du nom étendu de l'élément soit non nulle.

4.4.3. Les expressions

Les expressions sont utilisées dans XSLT pour une variété de possibilités incluant : sélection de nœuds pour traitement; spécification des conditions pour les différentes manières de traitement d'un nœud; génération de texte à insérer dans l'arbre résultat.

Dans XSLT, une expression autonome (c'est à dire : une expression qui ne fait partie d'aucune autre expression) prend son contexte comme suit :

- Le nœud contextuel est le nœud courant
- la position contextuelle vient de la position du nœud courant dans la liste courante
- le nœuds; la première position est 1
- la dimension contextuelle est issue de la dimension de la liste courante de nœuds
- les liens variables sont ceux dans le périmètre de l'élément ayant un attribut dans lequel l'expression se produit

4.4.4. Règles modèle

4.4.4.1. Modèle de traitement

Une liste de nœuds sources est traitée pour créer un fragment de l'arbre résultat. L'arbre résultat est construit par traitement d'une liste contenant juste le nœud racine. Une liste de nœuds sources est traitée par rajout successif des arbres résultants du traitement successif de chaque nœud de la liste. Un nœud est traité en trouvant toutes les règles modèle dont les motifs correspondent au nœud, et en choisissant le meilleur parmi elles; la règle modèle retenue est alors instanciée en considérant le nœud comme nœud courant et la liste de nœuds source comme liste courante de nœuds Typiquement, un modèle contient des instructions traitant une liste de nœuds source sélectionnés. Le processus de correspondance, instanciation et sélection se poursuit récursivement et se termine lorsque aucun nouveau nœud source n'est sélectionné pour traitement.

4.4.4.2. Modèles

Les règles modèle identifient les nœuds auxquels elles s'appliquent en utilisant un motif. Un motif spécifie un ensemble de conditions sur un nœud. Un nœud satisfaisant les conditions correspond au motif; un nœud qui ne satisfait pas les conditions ne correspond pas au motif. Une expression, qui est aussi un motif, donne toujours comme résultat un objet qui est un ensemble de nœuds Un nœud correspond à un motif si le nœud est membre du résultat de l'évaluation du motif, considéré alors comme expression respectant un certain contexte; ce cas se produit quand le nœud contextuel est le nœud en train d'être mis en correspondance ou l'un de ses ancêtres.
Voici quelques exemples de motifs:

- para correspond à n'importe quel élément para
- * correspond à n'importe quel élément
- chapter|appendix trouve tous les éléments chapter et appendix
- olist/item trouve tous les éléments item ayant olist comme parent
- appendix//para correspond à n'importe quel élément para dont un ancêtre est appendix
- / correspond au noeud racine
- text() correspond à n'importe quel noeud textuel
- processing-instruction() correspond à n'importe quel instruction de traitement
- node() correspond à n'importe quel noeud autre qu'un noeud attribut et que le noeud racine
- id("W11") correspond à l'élément ayant l'identifiant unique W11
- para[1] correspond au premier des éléments para d'un parent
- items/item[position()>1] correspond à tous les éléments item, autres que le premier, et dont le parent est items
- item[position() mod 2 = 1] serait vrai pour tous les fils item de numéro d'ordre impair par rapport à leur parent.
- div[@class="appendix"]//p correspond à tous les éléments p ayant un ancêtre div pour lequel l'attribut class prend la valeur appendix
- @class correspond à tous les attributs class (pas à tous les éléments ayant l'attribut class)
- @* correspond à n'importe quel attribut

4.4.4.3. Définition de règles modèle

<!--Catégorie : élément de haut niveau -->
<**xsl:template**
match = pattern
name = qname
priority = number
mode = qname>
<!-- Contenu : (xsl:param*, template)-->
</xsl:template>

Une règle modèle est spécifiée en utilisant l'élément xsl:template. L'attribut `match` est un motif qui identifie le nœud source ou les nœuds pour lesquels la règle s'applique. L'attribut `match` est exigé à moins que l'élément xsl:template ait un attribut name. Le contenu de l'élément xsl:template est le modèle qui est instancié lorsque la règle modèle est appliquée.

Comme décrit dans ce qui suit, l'élément xsl:apply-templates traite d'une manière récursive les fils de l'élément source.

4.4.4.4 Application des règles modèle

<!--Catégorie : instruction -->
<**xsl:apply-templates**
select = node-set-expression
mode = qname>
<!-- Contenu : (xsl:sort | xsl:with-param)*-->
</xsl:apply-templates>

L'exemple suivant crée un bloc pour l'élément chapter et traite ses fils directs. [6]

```
<xsl:template match="chapter">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

En l'absence de l'attribut select, l'instruction xsl:apply-templates traite tous les fils du nœud courant, y compris les nœuds textuels.
Un attribut `select` peut être utilisé pour traiter les nœuds sélectionnés par une expression au lieu de traiter tous les éléments. La valeur de l'attribut select est une expression. L'expression doit retourner un ensemble de nœuds L'ensemble des nœuds sélectionnés est traité dans l'ordre du document, à moins qu'une spécification d'ordonnancement ne soit présente.
L'exemple suivant traite tous les fils author de author-group :

[6] Les exemples proviennent du site : http://www.w3.org/TR/1999/REC-xslt-19991116

```
<xsl:template match="author-group">
  <fo:inline-sequence>
    <xsl:apply-templates select="author"/>
  </fo:inline-sequence>
</xsl:template>
```

4.4.5. Modèles nommés

```
<!--Catégorie : instruction -->
<xsl:call-template
  name = qname>
  <!-- Contenu : xsl:with-param*-->
</xsl:call-template>
```

Les modèles peuvent être invoqués par leur nom. L'utilisation de l'attribut name de l'élément xsl:template permet de spécifier un nom de modèle. L'élément xsl:call-template permet d'appeler un modèle par son nom; il a un attribut obligatoire name qui identifie le modèle à appeler. Au contraire de l'élément xsl:apply-template, xsl:call-template ne modifie pas le nœud courant ou la liste courante de nœuds

4.4.6. Création de l'arbre résultant

Cette section décrit les instructions qui créent directement des nœuds dans l'arbre résultant.

4.4.6.1. Créer des éléments

```
<!-- Category: instruction -->
<xsl:element
  name = { qname }
  namespace = { uri-reference }
  use-attribute-sets = qnames>
  <!-- Content: template -->
</xsl:element>
```

L'élément xsl:element permet la création d'un élément dans l'arbre de sortie. Le nom étendu de l'élément à créer est spécifié par l'attribut obligatoire name et un attribut optionnel namespace. Le contenu de l'élément xsl:element est le modèle des attributs et enfants de l'élément créé.

4.4.6.2. Créer des attributs

```
<!-- Category: instruction -->
<xsl:attribute
  name = { qname }
  namespace = { uri-reference }>
  <!-- Content: template -->
</xsl:attribute>
```

L'élément xsl:attribute peut être utilisé pour ajouter des attributs aux éléments résultats qui sont créés soit par un élément résultat littéral dans la feuille de style soit par des instructions telles que xsl:element. Le nom étendu de l'attribut à créer est spécifié par l'attribut obligatoire name et un attribut optionnel namespace. Instancier un élément xsl:attribute ajoute un nœud attribut à l'élément nœud résultat. le contenu de l'élément xsl:attribute est un modèle de la valeur de l'attribut créé.

4.4.6.3. Créer du texte

Un modèle peut aussi contenir des nœuds textuels. Chaque nœud textuel du modèle créera un nœud textuel de même valeur textuelle dans l'arbre résultant. Les nœuds textuels adjacents d'un arbre résultat seront automatiquement fusionnés.

```
<!-- Category: instruction -->
<xsl:text
  disable-output-escaping = "yes" | "no">
  <!-- Content: #PCDATA -->
</xsl:text>
```

4.4.6.4. Créer des commentaires

```
<!-- Category: instruction -->
<xsl:comment>
  <!-- Content: template -->
</xsl:comment>
```

L'élément commentaire xsl:comment sert à créer un nœud commentaire dans l'arbre résultant. Le contenu de l'élément xsl:comment sert de modèle pour la valeur textuelle du nœud commentaire.
Par exemple :
<xsl:comment>Ce fichier est généré automatiquement. Ne pas éditer!</xsl:comment>
Créera ce commentaire :
<!-- Ce fichier est généré automatiquement. Ne pas éditer!-->

4.4.6.5. Copier

```
<!-- Catégorie : instruction -->
  <xsl:copy>
     <!-- Content: template -->
  </xsl:copy>
```

L'élément xsl:copy fournit un moyen simple pour copier le nœud courant. L'instanciation de l'élément xsl:copy crée une copie du nœud courant. Les nœuds espace de noms du nœud courant sont également copiés automatiquement, mais les attributs et les enfants du nœud ne sont pas copiés automatiquement. Le contenu de l'élément xsl:copy est le modèle des attributs et enfants à créer sur le nœud résultant ; le contenu est instancié uniquement pour les nœuds dont le type leur permet d'avoir des attributs ou des enfants (par exemple les nœuds racines et les nœuds d'éléments).

4.4.6.6. L'élément xsl:value-of

```
<!-- Catégorie : instruction -->
<xsl:value-of
  select = string-expression  />
```

A l'intérieur d'un modèle, l'élément xsl:value-of peut être utilisé pour calculer le texte généré, par exemple en extrayant du texte de l'arbre source ou en insérant le contenu d'une variable. L'élément xsl:value-of permet de faire cela par utilisation d'une expression spécifiée comme étant la valeur de l'attribut select. Les expressions peuvent également être utilisées à l'intérieur de valeurs d'attributs d'éléments résultats littéraux en entourant l'expression d'accolades ({}).

L'élément xsl:value-of est instancié pour créer un nœud textuel de l'arbre résultant. L'attribut obligatoire `select` est une expression ; cette expression est évaluée et l'objet résultant est converti en une chaîne de caractères. La chaîne de caractères spécifie la valeur textuelle du nœud textuel créé. Si la chaîne est vide, le nœud textuel ne sera pas créé. Le nœud textuel créé sera fusionné avec tout autre nœud textuel adjacent.
L'élément xsl :copy-of peut être utilisé pour copier un ensemble de nœuds vers l'arbre résultat sans les convertir en chaîne.
Par exemple, les déclarations suivantes créent un paragraphe HTML à partir de l'élément person et de ses attributs given-name et family-name. Le paragraphe contiendra la valeur de l'attribut given-name du nœud courant suivie d'un espace et de la valeur de l'attribut family-name du nœud courant.

```
<xsl:template match="person">
  <p>
   <xsl:value-of select="@given-name"/>
   <xsl:text>
</xsl:text>
  <xsl:value-of select="@family-name"/>
  </p>
</xsl:template>
```

4.4.7. Répétition

```
<!-- Category: instruction -->
<xsl:for-each
  select = node-set-expression>
  <!-- Content: (xsl:sort*, template) -->
</xsl:for-each>
```

Il est utile de pouvoir spécifier directement le modèle pour les nœuds sélectionnés lorsque le résultat est une structure régulière connue. L'instruction xsl:for-each contient un modèle qui est instancié pour chaque nœud sélectionné par l'expression spécifiée en tant que valeur de l'attribut `select`. L'attribut `select` est obligatoire. Le résultat de l'évaluation de l'expression doit être un ensemble de nœuds Le modèle est instancié avec le nœud sélectionné identifié comme le nœud courant, et avec la liste de tous les nœuds sélectionnés identifiée comme la

liste courante de nœuds. À moins qu'une spécification d'ordre ne soit présente, les nœuds sont traités dans l'ordre induit par le document .
Par exemple, Considérons un document XML avec la structure suivante :

```
<customers>
  <customer>
    <name>...</name>
    <order>...</order>
    <order>...</order>
  </customer>
  <customer>
    <name>...</name>
    <order>...</order>
    <order>...</order>
  </customer>
</customers>
```

ce qui suit permet de créer un document HTML contenant une table contenant une ligne pour chaque élément `customer`

```
<xsl:template match="/">
  <html>
    <head>
      <title>Customers</title>
    </head>
    <body>
      <table>
        <tbody>
          <xsl:for-each select="customers/customer">
            <tr>
              <th>
                <xsl:apply-templates select="name"/>
              </th>
              <xsl:for-each select="order">
                <td>
                  <xsl:apply-templates/>
                </td>
              </xsl:for-each>
            </tr>
          </xsl:for-each>
        </tbody>
      </table>
    </body>
  </html>
</xsl:template>
```

4.4.8. Traitement conditionnel

XSLT propose deux instructions qui permettent le traitement conditionnel dans un modèle : xsl:if et xsl:choose. L'instruction xsl:if sert à exprimer la condition simple si-alors ; L'instruction xsl:choose permet quant à elle un choix parmi plusieurs possibilités.

4.4.8.1. Traitement conditionnel par xsl :if

```
<!-- Category: instruction -->

<xsl:if
  test = boolean-expression>
  <!-- Content: template -->
</xsl:if>
```

L'élément xsl:if a un attribut test qui spécifie une expression. Le contenu est un modèle. Après évaluation de l'expression, le résultat est converti en booléen. Si le résultat est vrai alors le contenu du modèle est instancié, sinon rien n'est créé . Dans l'exemple suivant, les noms dans un groupe de noms sont formatés sous la forme d'une liste de noms séparés par des virgules :

```
<xsl:template match="namelist/name">
  <xsl:apply-templates/>
  <xsl:if test="not(position()=last())">, </xsl:if>
</xsl:template>
```

4.4.8.2. Traitement conditionnel par xsl:choose

```
<!-- Category: instruction -->

<xsl:choose>
  <!-- Content: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>
<xsl:when
  test = boolean-expression>
  <!-- Content: template -->
</xsl:when>
<xsl:otherwise>
  <!-- Content: template -->
</xsl:otherwise>
```

L'élément xsl:choose permet de faire un choix parmi plusieurs alternatives. Il consiste en une séquence d'éléments xsl:when suivis par un élément xsl:otherwise optionnel. Chacun des éléments xsl:when possède un attribut test unique qui spécifie une expression. Le contenu des éléments xsl:when et xsl:otherwise est un modèle. Le traitement d'une instruction xsl:choose, consiste à tester à tour de rôle chacun des éléments xsl:when en évaluant l'expression et en convertissant l'objet résultant en un booléen. Seul le contenu du premier élément xsl:when évalué à vrai est instancié. Si aucun des éléments xsl:when n'est évalué à vrai, c'est le contenu de l'élément xsl:otherwise qui est instancié. Si aucun des éléments xsl:when n'est évalué à vrai et qu'il n'y a pas d'élément xsl:otherwise alors rien n'est créé.

4.4.9. Tri

```
<xsl:sort
  select = string-expression
  lang = { nmtoken }
  data-type = { "text" | "number" | qname-but-not-ncname }
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" } />
```

Le tri est spécifié en ajoutant des éléments xsl:sort comme fils d'un élément xsl :apply-templates ou d'un élément xsl:for-each. Le premier fils xsl:sort spécifie la clé primaire de tri, le deuxième fils xsl:sort spécifie la clé de tri secondaire et ainsi de suite. Lorsqu'un élément xsl :apply-templates ou un élément xsl:for-each a un ou plusieurs fils xsl:sort, au lieu de traiter les nœuds sélectionnés dans l'ordre où ils apparaissent dans le document, ces derniers sont triés en fonction des clés de tri spécifiés et traités dans l'ordre induit par ce tri. Lorsque des éléments xsl:sort sont utilisés dans xsl:for-each, ils doivent apparaître en premier.
Lorsqu'un modèle est instancié par xsl :apply-templates et xsl:for-each, la liste de nœuds courante consiste en la liste complète des nœuds en cours de traitement dans l'ordre spécifié par le tri.
xsl:sort possède un attribut select dont la valeur est une expression. Pour chaque nœud à traiter, l'expression est évaluée avec ce nœud pris comme le nœud courant et avec la liste complète non triée des nœuds en cours de traitement prise comme la liste courante de nœuds L'objet résultant est convertit en une chaîne de caractères; cette chaîne de caractères résultat est utilisée comme clé de tri pour ce nœud. La valeur par défaut de l'attribut select est le caractère point "." dont l'occurrence induit l'utilisation comme clé de tri de la représentation sous forme de chaîne de caractères, du nœud courant.
Cette chaîne de caractères sert de clé de tri pour le nœud. Les attributs optionnels suivants de xsl:sort contrôlent comment les clés de tri de la liste sont ordonnées.
order indique si les chaînes de caractères devraient être triées dans un ordre ascendant ou descendant; la valeur ascending spécifie un ordre ascendant; la valeur descending spécifie un ordre descendant; par défaut, l'ordre est ascending
lang indique la langue des clés de tri; En l'absence de valeur de l'attribut lang, la langue est déterminée à partir de l'environnement système.
data-type indique le type de données des chaînes de caractères ; Les valeurs suivantes sont permises:
text indique que les clés de tri doivent être triés correctement dans l'ordre lexicographique de la langue spécifiée par l'attribut lang
number indique que les clés de tri doivent être converties en nombre et ensuite triées en fonction de leur valeur numérique. Ces clés de tri sont converties en nombres. Dans ce cas, l'attribut lang est ignoré.

4.4.10. Variables et paramètres

```
<!-- Category: top-level-element -->

<!-- Category: instruction -->
<xsl:variable
  name = qname
  select = expression>
```

```
  <!-- Content: template -->
</xsl:variable>
<!-- Category: top-level-element -->
<xsl:param
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:param>
```

Une variable est un nom pouvant être rattaché à une valeur. La valeur à laquelle est rattachée la variable (la **valeur** de la variable) peut être un objet de n'importe quel type que les expressions peuvent retourner. Il y a deux éléments qui peuvent être rattachés à des variables: xsl:variable et xsl:param. La différence est que la valeur spécifiée pour la variable de xsl:param est seulement une valeur de lien par défaut; lorsque le modèle ou la feuille de style dans lequel l'élément xsl:param se produit est invoqué, des paramètres peuvent être passés pour être utilisés à la place des valeurs par défaut.
xsl:variable et xsl:param ont un attribut name obligatoire qui indique le nom de la variable. La valeur de l'attribut name est un Qname.

4.4.10.1. Fragments d'arbre résultat

Les variables introduisent un type de données additionnel dans le langage d'expression. Ce type de données est appelé **fragment d'arbre résultat**. Une variable peut être liée à un fragment d'arbre résultat au lieu d'être liée à l'un des quatre types de base de XPath (string, number, boolean, node-set). Un fragment d'arbre résultat est un fragment de l'arbre résultat. Il est traité de la même manière qu'un ensemble de nœuds contenant uniquement un simple nœud racine. Les opérations autorisées sur un fragment d'arbre résultat ne sont, cependant, qu'un sous-ensemble des opérations permises sur un ensemble de nœuds Une opération n'est permise sur un fragment d'arbre résultat que si elle pouvait l'être sur une chaîne de caractères (L'opération sur la chaîne de caractères peut entraîner sa conversion en un nombre ou en un booléen en premier lieu). Il n'est particulièrement pas autorisé d'utiliser les opérateurs /, //, et [] sur les fragments d'arbre résultat. L'exécution d'une opération valide sur un fragment d'arbre résultat se déroule de la même manière que si cette opération est exécutée sur un ensemble de nœuds équivalent au fragment d'arbre résultat.
Lors de la copie d'un fragment d'arbre résultat dans l'arbre résultat, tous les nœuds fils du nœud racine de l'ensemble des nœuds équivalent au fragment d'arbre résultat sont ajoutés en séquence à l'arbre résultat.
Les expressions ne peuvent retourner des valeurs de type fragment d'arbre résultat que par des références à des variables de type fragment d'arbre résultat ou en appelant des fonctions d'extension retournant un fragment d'arbre résultat ou récupérant les propriétés système dont les valeurs sont des fragments d'arbre résultat.

4.4.10.2. Valeurs des variables et des paramètres

Un élément de lien de variable peut spécifier la valeur de la variable selon trois alternatives. Si l'élément de lien de variable dispose d'un attribut select, alors la valeur de l'attribut doit être une expression et la valeur de la variable est l'objet résultant de l'évaluation de l'expression. Dans ce cas le contenu doit être vide.

Si l'élément de lien de variable ne dispose pas d'un attribut select et que son contenu n'est pas vide (i.e. l'élément de lien de variable a au moins un nœud fils), alors ce contenu indique la valeur. Le contenu de l'élément de lien de variable est un modèle qui, lorsqu'il est instancié donne la valeur de la variable. La valeur est un fragment d'arbre résultat équivalent à l'ensemble de nœuds contenant un nœud racine unique. Ce nœud racine a une séquence de nœuds fils résultant de l'instanciation du modèle. L'URI de base des nœuds dans le fragment d'arbre résultat est l'URI de base de l'élément de lien de variable.
Si le contenu de l'élément de lien de variable est vide et cet élément n'a pas un attribut select, alors la valeur de la variable est une chaîne de caractères vide. Ainsi

<xsl:variable name="x"/>
est équivalent à
<xsl:variable name="x" select=""""/>

4.4.10.3. Variables et paramètres de haut niveau

Les éléments xsl:variable et xsl:param sont tous deux autorisés à être utilisés comme éléments de haut niveau (top-level). Un élément lien de variable de haut niveau déclare une variable globale visible partout. Un élément xsl:param de haut niveau déclare un paramètre pour la feuille de style; XSLT ne définit pas le mécanisme permettant de transmettre les paramètres à la feuille de style. Une feuille de style ne peut contenir plus d'un lien à une variable de haut niveau ayant le même nom. Au niveau le plus haut, l'expression ou le modèle spécifiant la valeur d'une variable est évalué dans le même contexte le traitement du nœud racine d'un document source : le nœud courant est le nœud racine du document source et la liste courante de nœuds est une liste contenant uniquement le nœud racine du document source. Si le modèle ou l'expression spécifiant la valeur d'une variable globale *x* fait référence à une variable globale *y*, alors la valeur de *y* doit être calculée avant la valeur de *x*. S'il était impossible de traiter toutes les variables globales de la sorte alors c'est une erreur.
L'exemple suivant montre la déclaration d'une variable globale para-font-size, qui est référencée dans la valeur d'un attribut modèle.

```
<xsl:variable name="para-font-size">12pt</xsl:variable>
<xsl:template match="para">
 <fo:block font-size="{$para-font-size}">
   <xsl:apply-templates/>
 </fo:block>
</xsl:template>
```

4.4.10.4. Les variables et les paramètres dans les modèles

De même qu'il est possible d'utiliser dans les élément de haut niveau, xsl:variable et xsl:param sont aussi permis dans les modèles. Dans un modèle, xsl:variable est autorisé partout où une instruction l'est. Dans ce cas, le lien est visible pour tous les frères suivant ainsi que leurs descendants. Le lien n'est pas visible par l'élément xsl:variable lui-même. xsl:param est permis lorsqu'il est fils au début d'un élément xsl:template. Dans ce cas précis, le lien est visible pour tous les frères suivant et à leurs descendants. Le lien n'est pas visible par l'élément xsl:param lui-même.

4.4.10.5. Passage de paramètres aux modèles

```
<xsl:with-param
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:with-param>
```

Le passage de paramètres aux modèles se fait par le moyen de l'élément xsl:with-param. L'attribut name obligatoire indique le nom du paramètre (la variable dont la valeur doit être remplacée). La valeur de l'attribut name est un Qname. xsl:with-param est permis aussi bien dans xsl:call-template que dans xsl:apply-templates. La valeur du paramètre est spécifiée de la même manière que pour xsl:variable et xsl:param. Le nœud courant et la liste courante de nœuds utilisés pour calculer la valeur spécifiée par l'élément xsl:with-param sont les mêmes que ceux utilisés pour l'élément xsl:apply-templates ou pour l'élément xsl:call-template dans lequel il apparaît
L'exemple suivant défini un modèle nommé pour un numbered-block avec un argument pour contrôler le format du nombre.

```
<xsl:template name="numbered-block">
  <xsl:param name="format">1. </xsl:param>
  <fo:block>
    <xsl:number format="{$format}"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="ol//ol/li">
  <xsl:call-template name="numbered-block">
    <xsl:with-param name="format">a. </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Chapitre 5 : La visionneuse de requêtes

Chapitre 5 : La visionneuse de requêtes

5.1. Introduction

La visionneuse est utilisée pour afficher le résultat d'une ou plusieurs requêtes. L'affichage peut contenir aussi bien des données spatiales que thématiques. Il peut être sous la forme d'un tableau ou d'une carte. Par exemple [7], le résultat d'une requête demandant les « noms de rues qui ont une intersection avec les Champs Elysées et qui sont à moins de 500 mètres de l'Arc de Triomphe » peut être affiché comme une table avec une information textuelle sur les rues sélectionnées ou comme une carte affichant la zone de Paris répondant aux critères de la requête. L'affichage peut afficher simultanément le résultat de plusieurs requêtes.

La visionneuse de requêtes dépend de manière implicite ou explicite de la définition des règles de présentation (voir éditeur de légende).

Les utilisateurs peuvent aussi raffiner les prédicats de la requête dont le résultat est actuellement affiché. Par exemple, l'utilisateur doit être capable de modifier facilement la requête ci-dessus pour changer la distance à 700 mètres, sans passer par le processus de formulation de requêtes en entier. Le prédicat peut aussi être spatial. L'utilisateur désirant récupérer des postes aux alentours d'un hôtel donné peut formuler la requête suivante "l'emplacement de bureaux de poste à 1 km d'ici" et fournir la valeur pour "ici" en indiquant (par un click) l'emplacement de l'hôtel dans une carte affichée. La requête pourrait aussi être l'emplacement de bureaux de poste dans un secteur. L'utilisateur définira le secteur en désignant deux ou plusieurs points sur une carte affichée. Finalement, le prédicat spatial pourrait être un ensemble d'entités. L'utilisateur définit les entités en utilisant les fonctionnalités de sélection. Ainsi, le constructeur de requêtes emploie les valeurs des paramètres spécifiés dans la visionneuse pour raffiner les prédicats.

L'utilisateur peut définir les paramètres de la requête pour exécuter des requêtes prédéterminées. Tous les prédicats de la requête ne sont pas statiques. Par exemple, une requête peut être conçue pour extraire des données sur des pays. Pour éviter la création d'autant de requêtes que de pays, la requête inclue l'identification du pays comme un paramètre dynamique. L'utilisateur désirant employer cette requête sera incité à choisir l'identification du pays dans la liste des valeurs autorisées. Dans ce cas, l'utilisateur doit seulement choisir des valeurs ou des gammes de valeurs dans une liste de valeurs. Les requêtes prédéterminées peuvent aussi avoir des paramètres spatiaux.

L'utilisateur peut aussi définir le contexte temporel et de multi-représentation d'une requête. Par exemple, si la requête prédéfinie demande une information selon un attribut temporel, l'utilisateur choisit une valeur ou une gamme de valeurs pour l'attribut temporel. De la même façon si plusieurs représentations sont attachées à une requête, l'utilisateur peut passer d'une représentation à une autre. En ce qui concerne les dimensions temporelles et de multi-représentation, un contexte par défaut peut être spécifié, par exemple, par ex : afficher les

[7] Murmur Project - Workpackage 4, The MURMUR implementation
Deliverable 12, Query Editor Specification
Reference : MM-WP4-DLA-012
Version : 9 date : 12/10/2001

résultats de la requête qui sont actuellement valables en ce qui concerne la dimension temporelle.

La visionneuse de requêtes doit fournir la réponse à trois questions:

- Quelles informations afficher ?
- Comment afficher les données spatiales ?
- Comment afficher les données thématiques ?

La définition de requête fournit une réponse à la première question. Une définition de légende doit être créée pour répondre à la deuxième. En effet, la requête a toute l'information nécessaire pour extraire les données spatiales et thématiques. La visionneuse de requêtes a besoin de plus d'information sur les paramètres de représentation des types des entités. L'éditeur de légende est utilisé pour définir l'information d'affichage pour dessiner les données spatiales. Pour permettre l'exposition de données pendant la conception de la requête, une légende par défaut peut être associée aux attributs et géométries spatiaux de MADS. Autrement, autant de légendes que nécessaire peuvent être définies pour afficher des données spatiales

Les fonctionnalités de la visionneuse de requêtes sont :

- Les fonctions **"Data set"** pour spécifier les données à afficher.
- Les fonctions "**Présentation"** pour modifier la façon dont les données sont affichées.
- **"Navigation"** pour changer le point de vue des données.
- **"Sélection"** pour sélectionner un sous-ensemble d'entités.
- **"Imprimer"** pour imprimer les données.

5.2. Data set

Pour savoir quelles données afficher, la visionneuse de requêtes utilise une définition de requête.
Quand l'utilisateur conçoit la requête, il peut, à n'importe quel moment, voir le résultat dans l'éditeur de la visionneuse. Il doit simplement effectuer un glisser/déposer de la requête sur la visionneuse de requêtes ou utiliser la fonction « ouvrir requête » de la visionneuse de requêtes.
Les fonctionnalités principales de la visionneuse de requêtes sont :

- **Ouvrir requête**
- **Ajouter requête**
- **Changer paramètres**
- **Ouvrir une autre fenêtre**
- **Fermer fenêtre**

Ces fonctionnalités sont décrites ci-dessous

figure 5.1 Barre de gestion des données

Ouvrir requête provoque l'extraction des données et l'ouverture des fenêtres graphique et d'attribut. La fenêtre graphique affiche les données spatiales tandis que la fenêtre d'attributs affiche les données thématiques.
L'utilisateur choisit dans une liste la requête à ouvrir. La définition de la requête spécifie quelles données devront être extraites.
Si des paramètres dynamiques sont définis dans les prédicats de la requête, l'utilisateur doit d'abord spécifier leurs valeurs avant que les données puissent être extraites. Ces paramètres sont thématiques, spatiaux ou temporelles. Si le paramètre est thématique ou temporel, l'utilisateur choisit la valeur dans une liste prédéfinie ou une gamme de valeurs (identificateur de pays, gamme de temps,…). Le paramètre spatial peut être une zone spatiale à choisir (point, rectangle, cercle,..) ou des entités à sélectionner dans la fenêtre graphique. Les données extraites dépendent des valeurs des paramètres dynamiques.
Tous les attributs spatiaux et thématiques extraits seront affichés. Par exemple, si la requête trouve plusieurs géométries correspondant à de multiples représentations ou à des temps multiples, par défaut, toutes les géométries seront affichées.
À tout moment, l'utilisateur peut **changer les paramètres** des requêtes affichées. On demande à l'utilisateur de faire les mêmes choix qu'à l'ouverture en accord avec les paramètres dynamiques de la requête. La requête est ré-exécutée pour afficher les données correspondant aux nouveaux paramètres.

Ajouter requête est utilisé pour ajouter des données à une fenêtre spatiale existante. L'utilisateur choisit une requête dans une liste de requêtes. Pour des données spatiales, les données correspondantes sont extraites et ajoutées aux données déjà affichées. Pour des données thématiques, une nouvelle fenêtre d'attribut est ouverte contenant les données nouvellement extraites.

L'éditeur peut gérer plusieurs fenêtres tandis que l'utilisateur peut **ouvrir une autre fenêtre**. La nouvelle fenêtre a les mêmes fonctionnalités que la principale. Donc, l'utilisateur peut montrer des données de plusieurs requêtes dans la même fenêtre ou dans différentes fenêtres. Chaque fenêtre montre les données d'unes ou plus requêtes. En conséquence, il voit les différentes représentations d'un même objet dans des fenêtres différentes ou bien dans une même fenêtre. Quand l'utilisateur ouvre une nouvelle fenêtre, il peut spécifier des relations de navigation avec d'autres fenêtres. Ces rapports impliquent que, quand il navigue dans une fenêtre, les autres fenêtres suivront en conséquence. Les relations de navigation peuvent forcer la même échelle ou le même zoom ou les deux dans les fenêtres liées. Ce cas est intéressant pour des représentations multiples dans des fenêtres différentes : l'utilisateur voit toujours le même secteur dans les fenêtres. On peut aussi avoir une fenêtre utilisée comme une vue générale d'une autre. La vue générale montre toujours l'encadrement de sa fenêtre parente.

Quand un utilisateur **ferme une fenêtre**, cette fenêtre et toutes les fenêtres associées sont fermées.

5.3. Le traitement du fichier GML

5.3.1 Introduction

Lors du développement de cet outil de visionnage, l'éditeur de requête n'était pas encore disponible, ce qui a impliqué que certaines fonctions du « data set » n'ont pas été développées et dans certains cas remplacées par des fonctions remplissant le même travail. Le bouton **ouvrir requête** a été remplacés par un bouton **ouvrir fichier** (figure 5.2), permettant la sélection et l'ouverture d'un fichier GML.

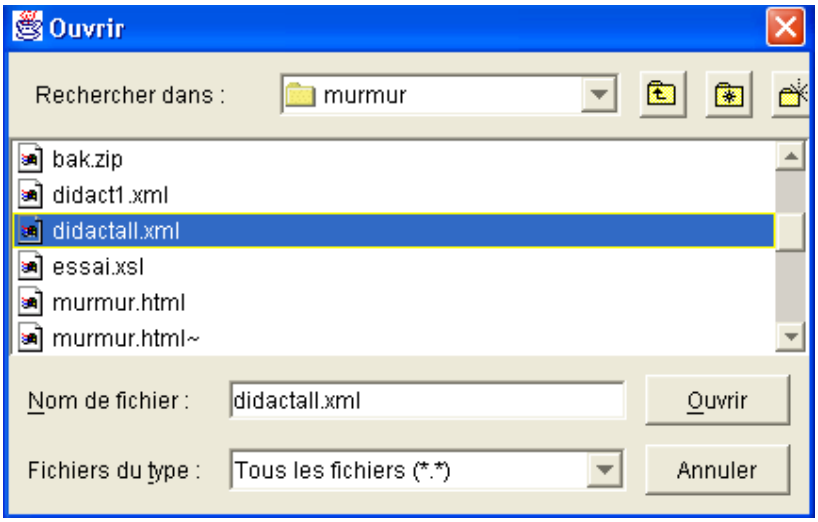


Figure 5.2 : ouverture d'un fichier GML

Une fois le fichier sélectionné, et qu'on appuie sur le bouton ouvrir, une série de traitements (la transformation) seront exécutés sur les données du fichier afin d'aboutir sur un affichage correct dans la visionneuse d'images.

5.3.2. La transformation

Nous allons diviser la transformation en quatre étapes afin de faciliter la compréhension. Ces étapes sont :

-La sélection des données à traduire
-Le changement de repère
-Le traitement des données
-L'affichage des données transformées

5.3.2.1 La sélection des données

Pour arriver à un résultat affichable sous forme de carte, nous nous intéresserons ici à la sélection des données spatiales et non des données alphanumériques (thématiques).

Voici un échantillon de données extrait d'un fichier GML contenant les données sur les différentes régions de l'Allemagne :

```
<ROWSET xmlns="http://www.didact.com"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.didact.com didact.xsd">

<gml:boundedBy>
   <gml:Box  srsName="http://www.opengis.net/gml/srs/epsg.xml#4326" >
         <gml:coordinates>5.868,47.2701  15.0443,55.066</gml:coordinates>
   </gml:Box>
</gml:boundedBy>

<gml:featureMember>
      <ROW num="1">
           <NAME>BERLIN</NAME>
           <DENSIT>3897.9</DENSIT>
           <GEOM>
             <gml:Polygon>
                  <gml:outerBoundaryIs>
                    <gml:LinearRing>
                        <gml:coordinates>13.094592,52.413958 13.104186,52.428288
13.110302,52.426742 13.117671,52.43573 13.126701,52.440826 13.11607,52.456623
13.121891,52.47919 13.105017,52.412909 13.104765,52.416079 13.10094,52.415315
13.101623,52.411559 13.094592,52.413958 </gml:coordinates>
                     </gml:LinearRing>
                  </gml:outerBoundaryIs>
             </gml:Polygon>
           </GEOM>
      </ROW>
</gml:featureMember>
```

Les données qui nous intéressent sont celles comprises entre les balises <gml:coordinates> et </gml:coordinates>.

En premier lieu, on extrait les coordonnées de <gml :Box>. La première valeur est construite avec les valeurs minimums mesurées sur tous les axes, le second avec les valeurs maximum mesurées sur tous les axes. Cela nous permettra de créer un cadre englobant la carte dans sa totalité. On stocke ces données dans une variable globale, afin de pouvoir les utiliser par après.

En XSLT, on obtient ceci :

```
<xsl:variable name="boxCoord" select="//gml:Box/gml:coordinates/." />
```

En deuxième lieu, on s'occupe de l'extraction des données spatiales.

```
<xsl:template
match="//gml:Polygon/gml:outerBoundaryIs/gml:LinearRing/gml:coordinates">
<xsl:variable name="clist" select="."/>
```

Le résultat est stocké dans une variable locale « clist » sous forme d'une liste de coordonnées. Ces deux variables « boxCoord » et « clist » serviront d'entrée pour les étapes suivantes.

5.3.2.2. Le changement de repère

Tout d'abord une remarque importante :
Le traitement des données ne sera pas effectué en XSLT mais en Java. Ceci est dû au fait que XSLT ne gère pas les boucles et donc on est obligé d'utiliser des méthodes récursives pour contourner cette limitation. S'il s'agit de traiter une liste de quelques coordonnées, cela ne pose aucun problème, mais dès qu'il s'agit de listes contenants des milliers de valeurs de coordonnées, le problème se pose, en effet lors de la compilation du programme, on obtient une erreur de dépassement de capacité. D'où l'obligation d'utiliser des appels de fonctions programmées en Java. De plus Java est un langage très puissant permettant un traitement rapide de grandes listes de données.
Pour pouvoir effectuer des appels à des fonctions externes, les trois lignes de code suivantes doivent être introduites dans la balise <xsl :stylesheet> :

- **xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"**
  org.apache.xalan.xslt.extensions.Redirect est le nom de la classe Java qui implémente l'extension et redirect est le préfixe (de l'espace de nom) utilisé pour appeler l'extension.
- **xmlns:mur="murmur"**
  la déclaration d'un espace de nom pour l'extension « mur »
- **extension-element-prefixes="mur"**
  désigne « mur » comme un préfixe d'extension

La ligne de code qui suit montre un appel de la méthode transfoBox(String) de la classe TransformCoord en utilisant la variable boxCoord :
<xsl:variable name="newBoxCord" select="mur:TransformCoord.transfoBox($boxCoord)"/>

Dans <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326" >, l'attribut srsName décrit le système de référence utilisé, ici on utilise epsg :4326, qui est un standard décrivant la représentation en latitude-longitude (degrés décimal) basé sur le « WGS84 datum ». Or, on sait que SVG ne supporte pas cette métrique, donc on va devoir utiliser des mètres pour représenter des cartes.
La transformation des coordonnées latitude-longitude va être effectuée en considérant que la terre est une sphère parfaite, et ceci afin de faciliter les calculs.

La Figure 5.3 illustre la situation d'un point M dans l'espace. Pour un repère d'origine O déterminé, ce point M sera défini par (x, y, z) dans le système de coordonnées cartésiennes et (r, φ, θ) dans le système de coordonnées sphériques, où φ représente la longitude, et θ la latitude.

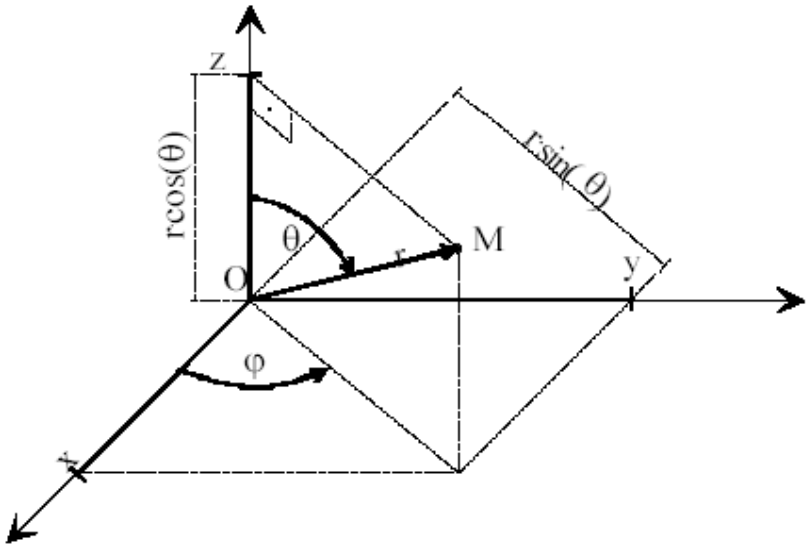


figure 5.3 coordonnées cartésiennes et coordonnées sphériques

Les transitions d'un système de coordonnées vers un autre se font de la manière suivante :

coord. cartésiennes → coord. sphériques	$r = \sqrt{x^2 + y^2 + z^2}$ $\theta = \arccos\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right)$ $\varphi = \operatorname{arctg}\left(\frac{y}{x}\right)$
coord. sphériques → coord. cartésiennes	$x = r \cdot \sin(\theta) \cdot \cos(\varphi)$ $y = r \cdot \sin(\theta) \cdot \sin(\varphi)$ $z = r \cdot \cos(\theta)$

On va procéder au changement de repère de la manière suivante :

Le système de coordonnées sphériques est défini par le repère **mobile (**M ,U_r, U_θ, U_φ). Soit M' la projection de M sur le plan xOy.

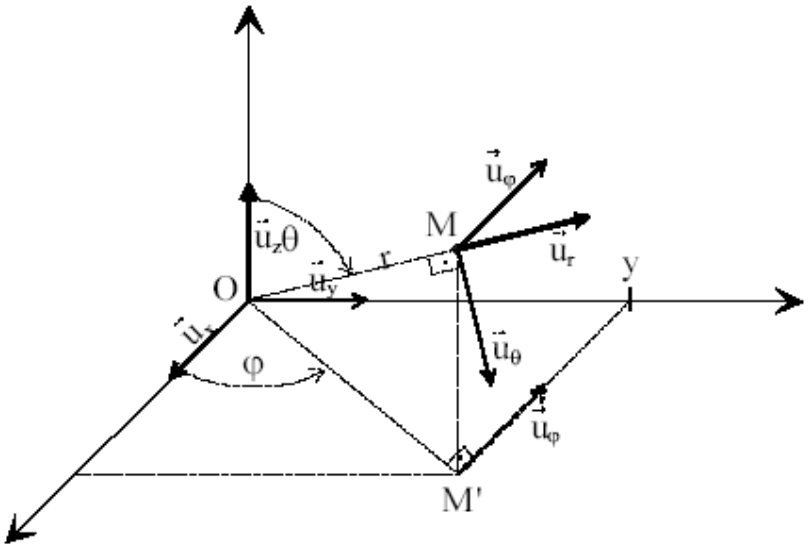


figure 5.4 changement de repère

U_r est parallèle au segment OM.

U_θ est perpendiculaire à U_r dans le plan MOz et son sens est celui de θ.

U_φ est perpendiculaire à U_r dans le plan xOy et son sens est celui de φ.

On calcule la moyenne des coordonnées de la balise <gml:Box>, et on place ce point en M, il va ainsi servir d'origine de notre nouveau repère. Puis on calcule les nouvelles coordonnées de chaque point de la carte, en utilisant la matrice de changement de repère :

$$\begin{pmatrix} \sin(\theta)\cdot\cos(\varphi) & \sin(\theta)\cdot\sin(\varphi) & \cos(\theta) \\ \cos(\theta)\cdot\cos(\varphi) & \cos(\theta)\cdot\sin(\varphi) & -\sin(\theta) \\ -\sin(\varphi) & \cos(\varphi) & 0 \end{pmatrix}$$

Il faut d'abord traduire les coordonnées des points exprimés en latitude-longitude en coordonnées cartésiennes avant d'effectuer le changement de repère.
Ensuite on fait une projection sur le plan (M, U_θ, U_φ), la projection se fait de manière très simple puisqu'il suffit de ne pas considérer l'axe U_r.

5.3.2.3. Le traitement des données

```
<xsl:template
match="//gml:Polygon/gml:outerBoundaryIs/gml:LinearRing/gml:coordinates">
<xsl:variable name="clist" select="."/>
<xsl:variable name="newCoord" select="mur:TransformJava.transfo($tclist)"/>
```

Les données contenues dans la variable clist se présentent sous la forme d'un String contenant les paires de coordonnées de chaque point.
Le traitement des données se fait par l'appel de la méthode transfo(String) (renvoie un String) de la classe TransformJava. Cette méthode va extraire les coordonnées une par une, les convertir en Double afin que les calculs mathématiques sur ces coordonnées puissent être effectués. Puis former un nouveau String avec les valeurs obtenues et en respectant les contraintes imposées par SVG pour la définition d'un path, à savoir, la première paire de coordonnées doit être précédée d'un M, et toutes les autres d'un L, le path doit être fermé par le symbole z et pour finir la méthode doit retourner le String obtenu. Les nouvelles coordonnées seront stockées dans la variable newCoord.

5.3.2.4. L'affichage des données transformées

Les données transformées vont être stockées dans un fichier au format SVG.
Pour cela le fichier devra contenir une balise <svg> avec un attribut viewBox possédant la valeur de la variable newBoxCord.

Voici la règle permettant d'effectuer cette opération :

```
<xsl:template match="/">
<xsl:variable name="newBoxCord" select="mur:TransformCoord.transfoBox($boxCoord)"/>
        <svg viewBox = "{$newBoxCord}">
                <xsl:apply-templates />
        </svg>
</xsl:template>
```

Les nouvelles coordonnées vont devoir être stockées dans un élément <path> avec un attribut <d> contenant les valeurs des coordonnées contenues dans la variable newCoord.
L'attribut "style" est utilisé pour décrire la manière d'afficher les coordonnées.
Ici on a juste choisit de leur affecter une couleur.

Voici la règle permettant d'effectuer cette opération :

```
<xsl:element name="path" >
        <xsl:attribute name="style">stroke:$colorc;fill:$remplir</xsl:attribute>
        <xsl:attribute name="d">
            <xsl:value-of select="$newCoord"/>
        </xsl:attribute>
</xsl:element>
```

Et voici un exemple fichier de sortie en SVG obtenu après transformation :

```
<path style="stroke:black;fill:blue" d="M 1323.756505 0.263145 L
1325.389258 2.0467457 L 1327.193817 2.6781603 L 1323.756501 -0.26314 z"/>
```

Ce qui suit est le listing du stylesheet permettant d'effectuer la transformation :

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xlink="http://www.w3.org/TR/xlink"
xmlns:gml="http://www.opengis.net/gml"
xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
xmlns:mur="murmur"
extension-element-prefixes="mur">

<xsl:output method="xml"
doctype-system="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-
stylable.dtd"
doctype-public="-//W3C//DTD SVG 20000303 Stylable//EN"/>

<xsl:variable name="boxCoord" select="//gml:Box/gml:coordinates/." />
<xsl:variable name="colorc">black</xsl:variable>
<xsl:variable name="remplir" select = "none" />

<xsl:template match="/">
<xsl:variable name="newBoxCord" select="mur:TransformCoord.transfoBox($boxCoord)"/>
        <svg xml:space="preserve" viewBox = "{$newBoxCord}">
                <xsl:apply-templates />
        </svg>
</xsl:template>

<xsl:template
match="//gml:Polygon/gml:outerBoundaryIs/gml:LinearRing/gml:coordinates">
        <xsl:variable name="clist" select="."/>
        <xsl:variable name="tclist" select="normalize-space($clist)" />
        <xsl:variable name="newCoord" select="mur:TransformJava.transfo($tclist)"/>
        <xsl:element name="path" >
            <xsl:attribute name="style">stroke:$colorc;fill:$remplir</xsl:attribute>
            <xsl:attribute name="d">
                <xsl:value-of select="$newCoord"/>
            </xsl:attribute>
        </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

Et voici un fichier GML représentant l'Allemagne transformé en SVG :

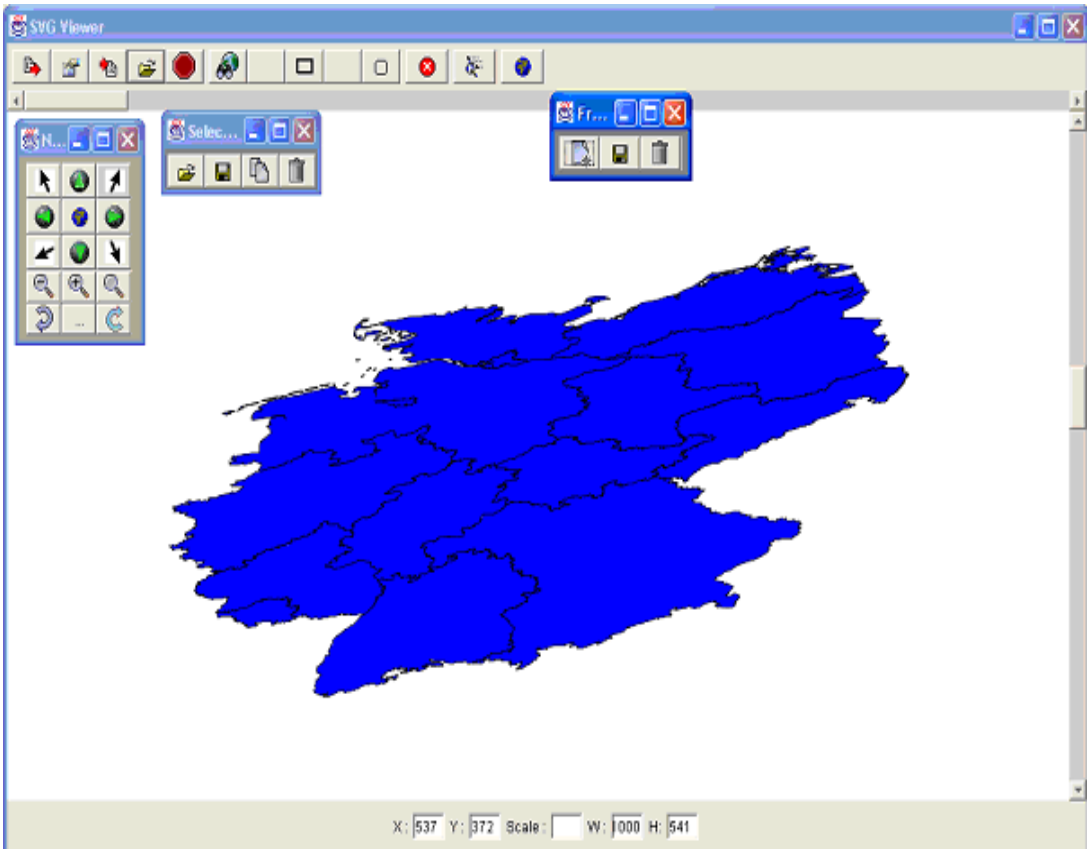


figure 5.5 carte d'Allemagne après transformation

5.4. Présentation

Après avoir choisi les requêtes, l'utilisateur peut raffiner les présentations des données spatiales et thématiques.
La visionneuse a besoin d'une légende pour afficher les données spatiales. Pour éviter la création d'une légende pour chaque requête, si l'utilisateur ne spécifie pas de légende pour sa requête, la visionneuse utilise la légende par défaut pour afficher les données spatiales.

L'utilisateur pourrait **appliquer la légende** pour changer les paramètres des données spatiales affichés. L'utilisateur choisit une légende dans une liste de légendes prédéterminées. Les légendes sont définies par l'éditeur de légende.

Afficher légende est utilisé pour afficher la légende et sa signification. Comme les données spatiales peuvent être affichées avec des légendes différentes, l'utilisateur peut voir la signification de la représentation

L'utilisateur peut traiter un type d'entité via les opérations suivantes ;

- Cacher ou montrez un type d'entité
- Modifiez la gamme d'échelle de visibilité des types d'entité
- Choisir l'ordre d'affichage des types d'entité

- Définir un état sélectionnable de type d'entité
- Choisir des valeurs de représentation et de temps

Cacher un type d'entité est utilisé pour cacher toutes les données spatiales et thématiques liées à ce type d'entité. Si l'utilisateur a ouvert plusieurs requêtes ou si une requête contient plusieurs types d'entités, l'utilisateur peut cacher un type d'entité temporairement.

Montrer un type d'entité, au contraire, montrera les données cachées liées à ce type d'entité.

Chaque type d'entité a une gamme d'échelle qui spécifie sur quelle échelle les entités sont visibles. **Modifiez la gamme d'échelle de visibilité des types d'entité** modifiera les entités affichées selon l'échelle actuelle. Ainsi, quand l'utilisateur appelle la fonction de navigation du zoom, les données affichées sont modifiées selon la gamme d'échelle du type d'entité.

L'ordre d'affichage des données spatiales est important pour voir les données correctement. Par exemple, si la requête trouve des parcelles et des bâtiments, la zone des parcelles doit être affichée sous la zone des bâtiments. Ainsi, l'utilisateur peut choisir l'ordre d'affichage des types d'entité pour spécifier dans quel ordre les données seront affichées.

L'utilisateur peut choisir des données spatiales ou thématiques de manière interactive. Pour faciliter ce processus de choix, l'utilisateur peut choisir l'entité géographique dont les entités peuvent être choisies. Il **définit l'état sélectionnable** de types d'entité. Seulement les entités de types d'entité sélectionnables peuvent être choisies en employant des fonctions de filtre.

Si la requête trouve des attributs multi-valeurs spatiaux ou thématiques dépendants du temps ou de la représentation, toutes les valeurs seront affichées. En effet, si l'attribut multi-valeurs est spatial, toute la géométrie correspondante sera affichée dans les fenêtres graphiques. Pour des attributs multi-valeurs thématiques, les valeurs apparaissent dans des « sous-formes ». L'utilisateur peut, pour chaque type d'entité **choisir une valeur dans le temps autorisé ou dans les valeurs de représentation** pour réduire les attributs multi-valeurs affichés.

Comme on l'a vu, les données spatiales sont affichées dans la fenêtre graphique en utilisant une légende qui définit comment les entités doivent être dessinées. Dans la fenêtre graphique, les coordonnées actuelles et l'échelle sont indiquées. Toutes les valeurs trouvées des attributs spatiaux multi-valeurs sont affichées dans cette fenêtre.
Les données thématiques sont affichées dans la fenêtre d'attribut. À chaque entité simple correspond une entrée dans une liste contenant toutes les valeurs des attributs thématiques de cette entité. Quant aux données spatiales, la requête définit quelles données thématiques doivent être affichées et le nom donné à ces attributs.

Les données thématiques peuvent être affichées sous forme de (voir figure 5.6) :

- Un tableau
- Un arbre
- Une forme

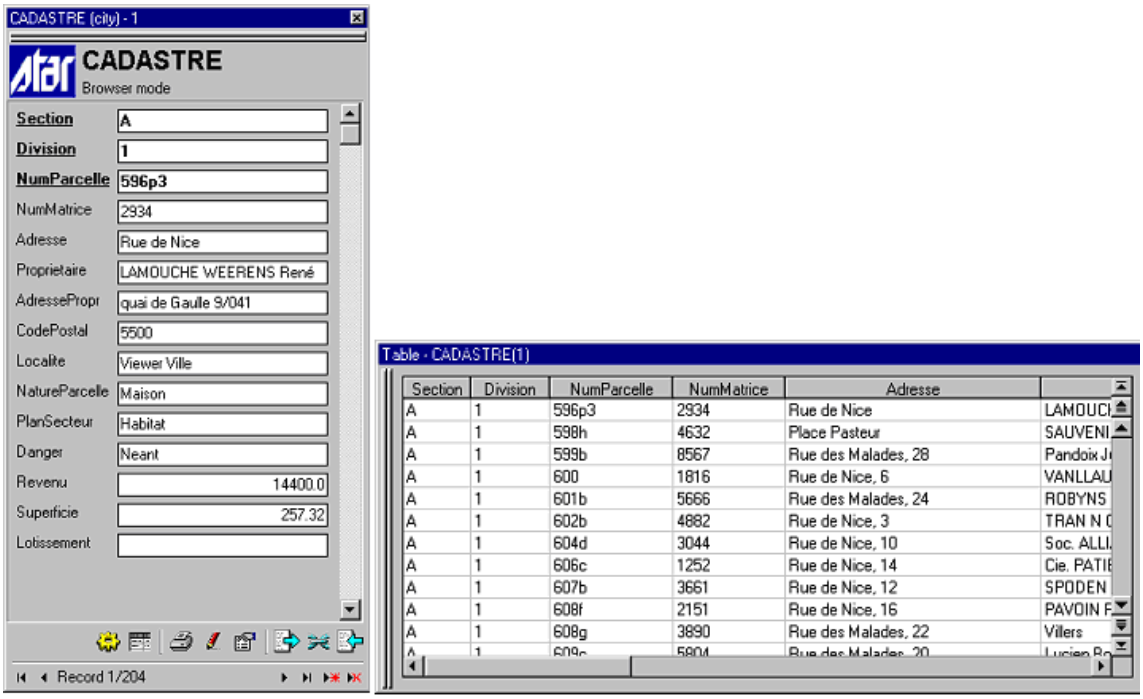


figure 5.6 affichage des données thématiques sous forme d'un arbre et d'un tableau

L'utilisateur doit être capable de :

- Choisir la manière de présenter les attributs (table, arbre, forme)
- Changer le nom (étiquette) d'un attribut
- Trier les instances
- Cacher ou montrer des attributs
- Choisir l'ordre des attributs

Il y a trois manières de présenter les données thématiques. L'utilisateur peut, à tout moment, **choisir la manière de présenter** les attributs qu'il veut afficher.

Les noms d'attribut par défaut sont spécifiés dans la requête. L'utilisateur est capable de **changer le nom d'un attribut.**

Pour changer l'ordre des entités, l'utilisateur peut **trier les instances**.

L'utilisateur peut aussi **cacher ou montrer les attributs** pour avoir plus ou moins d'information.

Il peut aussi **choisir l'ordre d'affichage des attributs**

5.5. L'extraction des données thématiques

Dans cette partie, nous nous intéresserons aux données thématiques et non plus aux données spatiales.
Le problème qui se pose est celui de déterminer quelles sont les entités que l'on va devoir extraire du fichier GML, puisque à priori nous nous connaissons pas leurs noms.

Heureusement, les fichiers GML sont toujours associés à des schémas d'application qui définissent la structure des entités qui seront utilisées dans le fichier GML.
Voici un exemple de schéma d'application :

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.didact.com"
xmlns:gml="http://www.opengis.net/gml" xmlns:ex="http://www.didact.com"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
        <!-- import constructs from the GML Feature and Geometry schemas -->
        <import namespace="http://www.opengis.net/gml" schemaLocation="feature.xsd"/>

<element name="ROWSET" substitutionGroup="gml:_FeatureCollection">
        <complexType>
                <complexContent>
                        <extension base="ex:RowsetType"/>
                </complexContent>
        /complexType>
</element>

<element name="ROW" type="ex:RowType" substitutionGroup="gml:_Feature"/>
   <complexType name="RowType">
     <complexContent>
       <extension base="gml:AbstractFeatureType">
         <sequence>
                <element name="NAME" type="string" nillable="true" minOccurs="0"/>
                <element name="DENSIT" type="string" nillable="true" minOccurs="0"/>
                <element name="GEOM" nillable="true" minOccurs="0">
                        <complexType>
                                <choice>
                                        <element ref="gml:Point" minOccurs="0"/>
                                        <element ref="gml:Polygon" minOccurs="0"/>
                                        <element ref="gml:LineString" minOccurs="0"/>
                                        <element ref="gml:MultiPolygon" minOccurs="0"/>
                                        <element ref="gml:MultiGeometry" minOccurs="0"/>
                                </choice>
                        </complexType>
                </element>
         </sequence>
         <attribute name="num" type="string" use="required"/>
       </extension>
     </complexContent>
   </complexType>
</schema>
```

A partir de ce schéma d'application, on obtient les informations suivantes :

On note que ce schéma personnalisé définit l'espace de noms 'ex' ; il importe les concepts des entités et de la géométrie GML de l'espace de noms 'gml'.

L'élément <ROWSET> est une instance du type défini par l'utilisateur 'ex: RowsetType' qui est dérivé par extension du type 'gml:AbstractFeatureCollectionType'. Le type 'ex:RowType' est dérivé par extension du type 'gml:AbstractFeatureType' qui est défini dans le schéma des entités GML ; ces dérivations assurent que le schéma de l'application est conforme avec les spécifications d'implémentation de GML du modèle des entités simples de l'OGC.
L'élément <ROW> est une instance du type défini par l'utilisateur 'ex: RowType'.
" substitutionGroup="gml:_Feature" est utilisé pour restreindre l'appartenance à la collection ROWSET.
RowType est défini comme étant de type complexe, et pourra donc contenir plusieurs éléments. Ces éléments sont : <NAME>, <DENSIT> et <GEOM> qui est lui aussi de type complexe et ne peut contenir qu'un seul des éléments suivants : <gml.Point>, "gml:Polygon", "gml:LineString", "gml:MultiPolygon", "gml:MultiGeometry".
L'élément <ROW> doit obligatoirement posséder l'attribut « num » de type String.
De cette analyse, on déduit qu'un fichier GML associé à ce schéma d'application aura la structure suivante :

```
<ROWSET …>
<gml:featureMember>
      <ROW num="x">
           <NAME>String</NAME>
           <DENSIT>String</DENSIT>
           <GEOM>
             <gml:Polygon> ou <gml:Point> ou …

                          CONTENU

            </gml:Polygon> ou …..
          </GEOM>
      </ROW>
</gml:featureMember>
</ROWSET>
```

Nous allons à présent essayer développer un stylesheet pour extraire les données qui nous intéressent.
Tout d'abord, une remarque importante, il est impératif d'ajouter ces deux lignes dans la balise <xsl :stylesheet> de notre stylesheet pour que celle-ci puisse traiter les schémas d'application :

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Pour extraire le nom de la collection d'entité on écrit la règle suivante :

```
<xsl:variable name="nameFeatureCollection"
select="/xsd:schema//xsd:element[@substitutionGroup='gml:_FeatureCollection']/@name"/>
```

On accède donc à l'élément dont l'attribut substitutionGroup a la valeur gml :_FeatureCollection et on sauve la valeur de l'attribut name qui contient le nom de la collection d'entités dans la variable nameFeatureCollection.

On fait de même pour les entités de la collection d'entités.

```
<xsl:variable name="nameFeature"
select="/xsd:schema//xsd:element[@substitutionGroup='gml:_Feature']/@name"/>
```

Avec la règle suivante, on extrait le type de l'entité appartenant à la collection d'entité

```
<xsl:variable name="temp"
select="/xsd:schema//xsd:element[@substitutionGroup='gml:_Feature']/@type"/>
<xsl:variable name="typeFeature" select="substring-after($temp,'ex:')"/>
```

substring-after($temp,'ex:') permet d'extraire le nom du type après avoir enlever le préfixe « ex: », la valeur est alors sauvée dans la variable typeFeature.

Pour chaque entité de type complexe, on va devoir extraire les noms des différentes entités qui la composent.

```
<xsl:template match="/xsd:schema//xsd:complexType">

        <xsl:if test="@name=$typeFeature">
                <xsl:for-each select=".//xsd:element">
                        <xsl:if test="@type">
                                <xsl:value-of select="@name"/>
                        </xsl:if>
                </xsl:for-each>
        </xsl:if>
</xsl:template>
```

Maintenant que nous avons en notre possession tous les éléments nécessaires, ce que nous proposons c'est d'utiliser des appels de fonctions externes à chaque fois qu'on extrait une donnée et de stocker le tout dans un tableau structuré pour faciliter l'utilisation par après.

Exemple d'appel de fonctions :

```
<xsl:variable name="bool1" select="mur:SchemaExtraction.setFeature($nameFeature)"/>
<xsl:variable name="bool2"
select="mur:SchemaExtraction.setFeatureCollection($nameFeatureCollection)"/>
```

Par la suite, il faudra développer des fonctions en Java et en utilisant JAXP de préférence, afin d'extraire directement les données thématiques en ce basant sur la table obtenue précédemment et de les mettre dans un tableau thématique.

Voici un exemple de listing permettant d'extraire les données d'un schéma d'application :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"  xmlns:mur="murmur"
extension-element-prefixes="mur">

<xsl:variable name="temp"
select="/xsd:schema//xsd:element[@substitutionGroup='gml:_Feature']/@type"/>
<xsl:variable name="typeFeature" select="substring-after($comp,'ex:')"/>

<xsl:variable name="nameFeature"
select="/xsd:schema//xsd:element[@substitutionGroup='gml:_Feature']/@name"/>

<xsl:variable name="nameFeatureCollection"
select="/xsd:schema//xsd:element[@substitutionGroup='gml:_FeatureCollection']/@name"/>

<xsl:template match="/xsd:schema//xsd:complexType">
        <xsl:if test="@name=$rest">
                <xsl:for-each select=".//xsd:element">
                        <xsl:if test="@type">
                                <xsl:value-of select="@name"/>
                        </xsl:if>
                </xsl:for-each>
        </xsl:if>
</xsl:template>

<xsl:template match="/xsd:schema">
<xsl:variable name="bool1" select="mur: SchemaExtraction.setFeature($nameFeature)"/>
<xsl:variable name="bool2" select="mur:
SchemaExtraction.setFeatureCollection($nameFeatureCollection)"/>
</xsl:template>
</xsl:stylesheet>
```

5.6. Navigation

Le résultat d'une requête est une collection d'entité. Chaque instance contient des valeurs d'attributs spatiaux et thématiques. L'utilisateur peut naviguer à travers ces instances en employant des fonctionnalités de navigation.

Pour les données spatiales, les fonctions de navigation sont : zoom, pan, …
Pour les données thématiques, les fonctions de navigation sont : suivant, précédent,…

L'utilisateur naviguera dans les données spatiales en utilisant :

- Aller aux 4 points cardinaux et sur les diagonales
- Zoom sur la vue générale
- Zoom out, zoom in
- Zoom centré sur un point
- Bouger latéralement
- Zoom sur une zone définie par deux points
- Position des coordonnées X, Y et échelle
- Changer d'échelle
- Fenêtre avant/suivant
- Zoom sur une sélection prédéfinie
- Zoom sur les valeurs min. et max. des entités

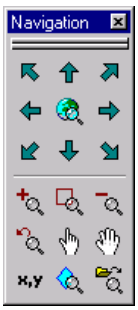


figure 5.7 Barre de navigation

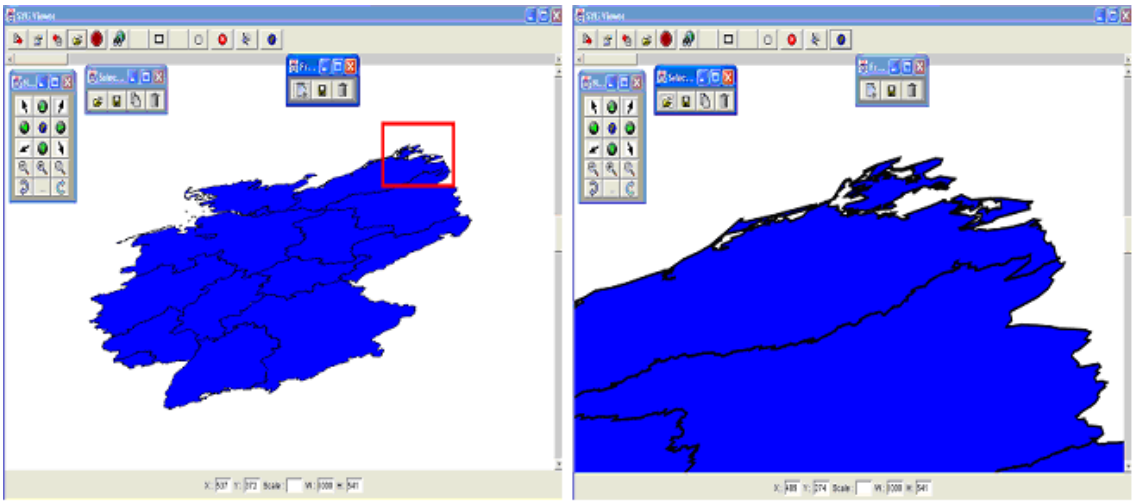


figure 5.8 zoom sur une région

La fenêtre courante peut être sauvée.

L'utilisateur va gérer les fenêtres en utilisant :

- Sauver la fenêtre courante
- Rappeler une fenêtre sauvée
- Effacer une fenêtre sauvée

figure 5.9 Barre de fenêtre

L'utilisateur va naviguer à travers les données thématiques en utilisant :

- Suivant et précédent
- Dernier et Premier
- Scrolling
- Recherche sur les valeurs d'attributs

Les fonctions **suivant et précédent** sont utilisées pour voir les instances suivantes ou précédentes dans les entités tandis que les fonctions **dernier, premier** sont utilisées pour voir les dernières ou premières instances.

L'utilisateur peut aussi vouloir trouver des entités avec des valeurs d'attribut spécifiques. Donc, il peut faire des opérations de recherche sur des valeurs d'attribut.

Comme les données graphiques et des données thématiques sont affichées dans des fenêtres différentes, l'utilisateur peut vouloir voir les données qui se correspondent dans les deux fenêtres. L'utilisateur peut demander de trouver des données thématiques correspondant aux données spatiales. Ou, au contraire, il peut demander de trouver des données spatiales correspondant aux données thématiques.

5.7. Sélection

L'utilisateur peut choisir des entités pour raffiner les prédicats des requêtes.
Il peut choisir des entités ou bien dans la fenêtre graphique ou bien dans la fenêtre d'attribut.
Et quand il a fini de choisir, il peut sauver ce choix.

Cette sélection des entités peut être :

- Sauver
- Appliquer
- Copier
- Effacer

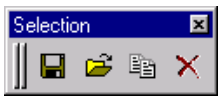


figure 5.10 Barre de sélection

Sauvez la sélection actuelle provoque la création d'une collection d'entités identifiées à laquelle on donne un nom. Le constructeur de Requête peut employer cet ensemble d'entités pour raffiner les prédicats de la requête.

Appliquer une sélection précédemment sauvée active cette sélection. Alors, l'utilisateur peut modifier le contenu de la sélection en employant des fonctions de filtre.

L'utilisateur peut aussi **copier** ou **supprimer** une sélection existante.

Toutes les entités sélectionnées seront mises en évidence, et dans le graphique et dans les fenêtres d'attribut.

Pour filtrer l'information, l'utilisateur choisira des entités. Tant qu'il n'a pas effacé le filtre, il ajoute des entités à la sélection actuelle en employant les fonctions suivantes.

Il peut **sélectionner les entités spatiales** :

- Par un point
- Entièrement et/ou Partiellement par un rectangle défini par deux ou trois points.
- Entièrement et/ou Partiellement par un cercle.
- Entièrement et/ou Partiellement par un polygone défini par x points.
- Entièrement et/ou Partiellement par une zone existante.

figure 5.11 Barre des filtres graphiques

Si l'utilisateur clique sur un point, toutes les entités dont la géométrie croise ce point est ajouté à la sélection.

Si l'utilisateur clique sur deux ou trois points pour définir un rectangle, toutes les entités dont la géométrie est dans le rectangle sont ajoutées à la sélection. L'utilisateur peut spécifier si la géométrie doit être entièrement ou en partie dans le rectangle.

L'utilisateur peut aussi choisir des entités **entièrement et/ou en partie dans un cercle**. L'utilisateur définit le cercle par le point de centre et une distance ou par le point de centre et un autre point.

L'utilisateur peut cliquer sur x points pour **définir un polygone**. Toutes les entités dont la géométrie est entièrement et/ou en partie dans ce polygone sont ajoutées à la sélection.

L'utilisateur peut aussi pointer sur une zone existante pour l'ajouter à la sélection d'entités dont la géométrie est entièrement et/ou en partie dans cette zone.

Il peut choisir **les entités thématiques** par :

- Sélectionner les instances une par une ou par gamme
- Spécifier les valeurs des attributs des types d'entités

5.8. Imprimer

Toutes les fenêtres décrites ci-dessus peuvent être imprimées individuellement ou simultanément.

L'utilisateur choisit les différentes fenêtres qu'il veut imprimer (graphique, fenêtre d'attribut ou fenêtres de légende).
Il définit les paramètres de page (la taille, l'orientation..), la position et la taille des différentes fenêtres, l'échelle d'impression des fenêtres graphiques, …
Le résultat peut être **prévisualisé** sur l'écran.

Chapitre 6 : L’éditeur de légende

Chapitre 6 : L'éditeur de légende [8]

6.1. Introduction

La requête définit quelles données doivent être extraites tandis que la légende définit comment les données spatiales doivent être affichées.

En effet, les données spatiales sont utilisées pour montrer les valeurs des attributs thématiques (le type de bâtiments, la densité de population dans un pays, la nature du sol, le niveau de pollution de rivières, …).
Une même donnée spatiale peut être utilisée pour montrer des attributs différents.
Une légende définit les paramètres d'affichage comme la couleur, le symbole, la taille, … des données spatiales selon les valeurs des attributs.

La légende définit:

- Les attributs thématiques concernés
- La gamme des valeurs pour définir des classes
- La symbologie (couleur, symbole, taille, …) associé aux classes
- L'étiquette qui montre les valeurs d'attribut comme le texte

La légende est liée aux attributs et plus précisément, avec la gamme significative des valeurs des attributs pour la définition des classes. C'est pourquoi la légende est associée à une requête ou à un groupe de requêtes employant ces attributs et ayant de la même gamme de valeurs.

Les fonctionnalités principales de l'éditeur de légende sont :

- **Sauver légende**
- **Appliquer légende**
- **Copier légende**
- **Effacer légende**
- **Afficher légende**
- **Appliquer requête**

Sauvez légende sauvera tous les paramètres actuels : attributs, classes, symbologie et étiquettes
L'utilisateur peut **appliquer une légende existante** pour faire des modifications.
L'utilisateur peut **copier** et **supprimer** une légende existante.

[8] Murmur Project - Workpackage 4, The MURMUR implementation
Deliverable 12, Query Editor Specification
Reference : MM-WP4-DLA-012
Version : 9 date : 12/10/2001

Afficher légende ouvrira une fenêtre de légende où la représentation graphique et sa signification sont affichées.
Pour créer une légende, l'utilisateur peut **appliquer une requête** pour définir les classes.
Pour définir une légende, l'utilisateur définira :

- Les paramètres thématiques définissant les attributs, la gamme de valeurs et la symbologie associés
- Les paramètres de la symbologie définissants, pour une symbologie, la couleur, taille, symbole, …
- Les paramètres d'étiquette définissant le contenu de texte, position, ….

Les groupes des paramètres thématiques, de symbole et des paramètres d'étiquette peuvent être sauvés séparément.
La légende est l'union des groupes de paramètres thématique, de symbole et d'étiquette.

6.2. Les paramètres thématiques

Les paramètres thématiques contiennent :

- Les attributs thématiques utilisés
- Les attributs spatiaux sur lesquels le thématique sera effectué.
- Définir des gammes de valeurs pour les attributs, ou bien manuellement ou bien automatiquement pour définir les classes.
- Pour chaque classe, choisir la représentation dans une table de symbologie.

L'utilisateur peut :

- Sauver les paramètres thématiques
- Appliquer des paramètres thématiques
- Supprimer des paramètres thématiques
- Copier des paramètres thématiques

L'utilisateur peut sauver tous les paramètres définissant le thématique. L'utilisateur peut appliquer des paramètres préexistants thématiques et les modifier. L'utilisateur peut copier ou supprimer des paramètres préexistants thématiques.

6.3. Les paramètres de la symbologie

Pour afficher une partie des données spatiales, L'éditeur de la visionneuse a besoin de paramètres d'affichage comme la couleur, la taille, le symbole, …
Le groupe de paramètres est réuni pour définir un type de représentation.
La table de symbologie réunit un groupe de représentations.

L'utilisateur peut :

- Sauver une table de symbologie
- Effacer une table de symbologie
- Copier une table de symbologie

- Appliquer une table de symbologie

L'utilisateur peut sauver tous les paramètres définissant la table de symbologie. L'utilisateur peut appliquer une table de symbologie préexistence et la modifier. L'utilisateur peut copier ou supprimer une table de symbologie préexistence.

Pour définir une table de symbologie, l'utilisateur doit :

- Choisir la description de la représentation à utiliser quand la légende est affichée.
- Choisir la gamme d'échelle à afficher.
- Choisir la couleur d'affichage dans une bibliothèque de couleurs.
- Éditer les symboles : Créez, modifiez et supprimez des symboles de formats STAR, GIF, JPEG, EMF et de fonte TrueType.
- Choisir le symbole à être associé aux points dans une bibliothèque de symboles.
- Définir la taille du symbole selon l'échelle ainsi que sa taille maximale et minimale.
- Choisir le symbole et les paramètres de répétition à appliquer à la ligne répétitivement.
- Éditer les modèles : Créez, modifiez et supprimez les modèles pour les surfaces.
- Choisir le (le vecteur ou raster) modèle à associer aux surfaces dans une bibliothèque de modèles.
- Choisir la fonte (Arial, Times New Roman, …), style de fonte (Normal, Bold, Italique, …), taille, effets (Strikethrough, Ombre, Indice, …) et espacement de caractère.
- Choisir le texte ou l'emplacement du symbole comme un halo ou une surface opaque pour une meilleure lisibilité du texte.

Il y a plusieurs modes d'affichage par gamme d'échelle pour un même objet.

6.4. Les paramètres d'étiquette

L'étiquette définit les textes à afficher selon les attributs. Une étiquette sera définie par les attributs à afficher, le type, …

L'utilisateur peut :

- Sauver une étiquette
- Supprimer une étiquette
- Copier une étiquette
- Appliquer une étiquette

L'utilisateur peut **sauver** tous les paramètres définissant l'étiquette. L'utilisateur peut **appliquer** une étiquette préexistante et la modifier. L'utilisateur peut copier ou supprimer une étiquette préexistante.

Pour définir une étiquette, l'utilisateur doit définir :

- Les attributs thématiques à afficher.
- Les attributs spatiaux sur lesquels l'étiquette sera appliquée.
- La forme d'étiquette :

    - Bulle de renseignements. Le texte de l'étiquette apparaît pour l'élément pointé.

- Forme dynamique avec des critères sur les éléments graphiques à étiqueter (toutes les entités, sur choix dynamique, sur choix, …). Les textes de l'étiquette apparaissent pour les éléments graphiques choisis.
- Forme statique avec des critères sur les éléments graphiques à étiqueter (toutes les entités, sur choix dynamique, sur choix, …). Les textes de l'étiquette apparaissent pour les éléments graphiques choisis. Ces textes peuvent être sauvés et mis à jour si les attributs ont été modifiés.

Les étiquettes sont automatiquement placées près du point, le long des axes curvilignes, dans le centre géométrique de la zone... L'utilisateur peut **déplacer les étiquettes statiques et dynamiques** de manière interactive et sauver la position des étiquettes statiques et dynamiques pour permettre une position identique lors d'un prochain affichage ou mise à jour.

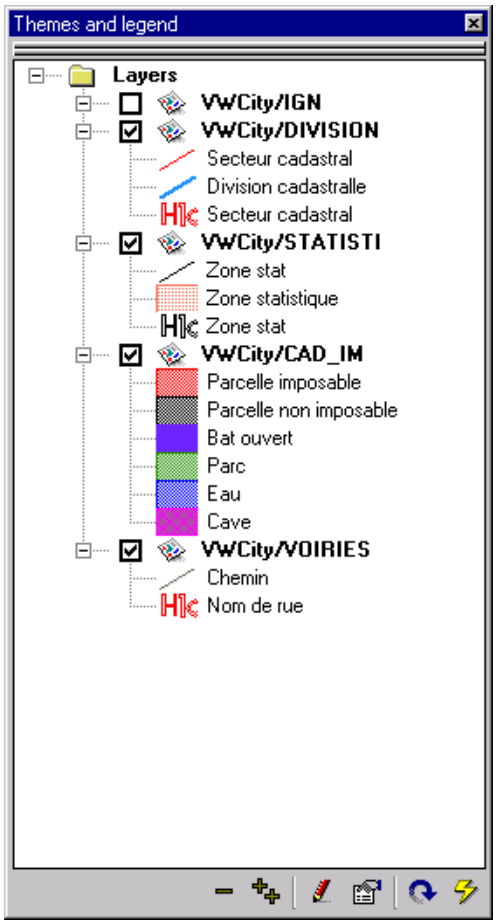


figure 6.1 Exemple de légende d'une carte

Conclusion

Nous avons développé dans le chapitre 5 « la visionneuse de requêtes » la procédure employée pour effectuer la transformation d'un document GML en un document SVG, les outils utilisés pour l'affichage ainsi que l'implémentation en JAVA du code utilisé pour gérer cette affichage (les opérations de zoom, de navigation,…) sont détaillés dans le mémoire écrit par Choukri Benali, qui a été mon partenaire sur ce projet.
Les objectifs premiers de ce projet à savoir la transformation et l'implémentation d'une interface graphique ont été atteints, même s'il subsiste quelques améliorations à ajouter (notamment en ce qui concerne la projection des cartes).
En ce qui concerne l'affichage des données thématiques, un stylesheet ainsi qu'un modèle permettent l'utilisation de ce stylesheet ont été introduits, il ne reste que la phase d'implémentation en code Java à effectuer pour arriver au résultat escompté.

Ce projet, étant dans sa première version, a nécessité de notre part un grand travail de recherche et d'investissement afin de réunir tous les outils et toutes la documentation nécessaires à l'aboutissement de ce projet , ainsi qu'une longue phase d'apprentissage notamment celui des langages XSLT, SVG et GML.

Une suite directe à notre travail consistera à faire ce que nous n'avons pas eu le temps de faire dans les délais impartis, à savoir le développement de l'éditeur de légende ainsi que des fonctionnalités de l'interface graphique qui n'ont pas été encore implémentés. Nos deux travaux (le mien et celui de Benali) pourront servir de base pour la suite de ce projet.

Bibliographie

Ouvrages :

XSLT
Doug Tidwell

Core JAVA
Gary Cornell Cay S. Horstmann
Second edition, 1997.

Graphic JAVA, Mastering the JFC
David M. Geary
Third Edition, 1999.

Multi-représentations dans les bases de données géographiques
Thèse de doctorat N° 2430 , 2001
Christelle VANGENOT
LBD DI, Ecole Polytechnique Fédérale de Lausanne, CH 1015 Lausanne

Publications :

GIS Databases: From Multiscale to MultiRepresentation[1]
Stefano Spaccapietra*, Christine Parent, Christelle Vangenot***

* Swiss Federal Institute of Technology Lausanne (EPFL)
EPFL-DI-LBD, 1015 Lausanne, Switzerland
{spaccapietra, vangenot}@epfl.ch
** University of Lausanne - HEC Inforge
1015 Lausanne, Switzerland
Christine.Parent@hec.unil.ch

MurMur: A Research Agenda on Multiple Representations
Stefano Spaccapietra*,Christelle Vangenot*, Christine Parent, Esteban Zimanyi*****

*Database Laboratory, Swiss Federal Institute of Technology
1015 Lausanne, Switzerland
**HEC/INFORGE, Université de Lausanne
1015 Lausanne, Switzerland
***Department of Informatics, Université Libre de Bruxelles
1050 Bruxelles, Belgique

Sites web :

SVG et Cartographie :
http://www.w3.org/TR/2001/REC-SVG-20010904/
http://www.carto.net/papers/svg/index_f.html
http://www.ihsenergy.com/epsg/guid7.html

GML :
http://opengis.net/gml/01-029/GML2.html

XSLT :
http://tecfa.unige.ch/guides/tie/html/xml-xslt/xml-xslt.html
http://www.w3.org/TR/1999/REC-xslt-19991116
http://www.ibiblio.org/xml/books/bible2/chapters/

Logiciels et outils :

Jbuilder
Version 6.0, Personal edition
http://www.borland.com

Batik
Version 1.0
http://xml.apache.org/batik