CUEFA
GRENOBLE UNIVERSITÉ

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

PRÉSENTÉ PAR **CHRISTOPHE CHABERT**

EN VUE D'OBTENIR

LE DIPLÔME D'INGENIEUR C.N.A.M.

EN INFORMATIQUE

HYPERATLAS ET *HYPERADMIN* : DES OUTILS CARTOGRAPHIQUES POUR L'ANALYSE DE PHÉNOMÈNES SOCIAUX

SOUTENU LE 29 MARS 2007

JURY

PRÉSIDENTE : MME VÉRONIQUE DONZEAU-GOUGE

MEMBRES : M. ERIC GRESSIER
M. JEAN-PIERRE GIRAUDIN
M. ANDRÉ PLISSON

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

PRÉSENTÉ PAR **CHRISTOPHE CHABERT**

EN VUE D'OBTENIR

LE DIPLÔME D'INGENIEUR C.N.A.M.

EN INFORMATIQUE

HYPERATLAS ET *HYPERADMIN* : DES OUTILS CARTOGRAPHIQUES POUR L'ANALYSE DE PHÉNOMÈNES SOCIAUX

Soutenu le 29 mars 2007

Les travaux relatifs à ce mémoire ont été effectués au laboratoire LSR-IMAG, *sous la direction de M. Jérôme Gensel.*

REMERCIEMENTS

Avant tout, je tiens à remercier toutes les personnes qui ont rendu possible l'achèvement de ce travail.

Je veux donc présenter mes remerciements aux membres du jury pour avoir permis la réalisation de l'étape finale de ce Mémoire. Merci donc à Madame Véronique Donzeau-Gouge et Monsieur Eric Gressier, professeurs au CNAM Paris ; à Monsieur Jean-Pierre Giraudin, professeurs à l'UPMF de Grenoble ; ainsi qu'à Monsieur André Plisson, directeur du centre d'enseignement CNAM de Grenoble.

Je souhaite également remercier les membres de l'équipe SIGMA qui m'on accueilli dans leur équipe et particulièrement à Monsieur Hervé Martin, professeur à l'université Joseph Fourier de Grenoble
J'aimerais également particulièrement remercier mon tuteur, Monsieur Jérôme Gensel, maître de conférences à l'UPMF, pour tous ses conseils, toute l'aide qu'il m'a apportée et le temps qu'il m'a consacré.

Merci également aux doctorants de l'équipe pour leur dynamisme et leur accueil chaleureux. Merci donc à Angela, Aurélie, Bogdan, Céline, Manuele, Marius.

Je souhaite également présenter mes remerciements à toutes les personnes avec lesquelles j'ai pu collaborer sur le projet *HyperAtlas* : Hélène, Liliane, Isabelle et Claude de l'équipe PARIS du laboratoire Géographie-cités et Saïd du laboratoire ID IMAG.

Pour conclure je souhaite également remercier ma famille et mes amis pour leur support et leur compréhension. Et surtout un grand merci à ma future femme pour sa compréhension et toutes les attentions dont elle a pu faire preuve durant cette épreuve.

TABLE DES MATIÈRES

GLOSSAIRE

A.C.I. : « Association Cartographique Internationale », est l'organisme faisant mondialement autorité en matière de cartographie, qui est la discipline traitant de la conception, de la production, de la diffusion et de l'étude des cartes.

Analyse Spatiale : étude formalisée de la configuration et des propriétés de l'espace produit et vécu par les sociétés humaines.

Carroyage : méthode qui permet de s'affranchir de l'arbitraire et de l'irrégularité d'un découpage administratif, et de mettre en évidence les grandes tendances de répartition spatiale d'une donnée. La donnée y est répartie sur un quadrillage régulier apposé sur la carte.

Carte : selon la définition de l'A.C.I., représentation sous forme de symbole de la réalité géographique, représentant des particularités et des caractéristiques spécifiques, résultant d'un effort créatif de la part de son auteur dans ses choix d'exécution, et conçue pour être utilisée lorsque les relations spatiales sont de la plus grande importance.

Carte choroplèthe : le type le plus usuel de carte statistique. Il s'agit d'une représentation de quantités (plethos) relatives à des espaces, ou aires géographiques (khorê), par le moyen d'une échelle de tons gradués.

Centroïde : point fictif, situé à l'intérieur d'un polygone, dont les coordonnées correspondent approximativement à celles du centre de ce polygone [@MRNF].

ESPON : (European Spatial Planning Observation Network) : programme lancé sous l'égide de l'initiative communautaire INTERREG III. Ce programme vise à établir un système permanent d'observation du territoire européen et de systématiser la coopération et la complémentarité entre les Etats membres de l'Union Européenne, la Commission Européenne et les instituts de recherche liés aux administrations responsables de l'aménagement du territoire. L'objectif principal est de développer les connaissances sur les structures territoriales, les tendances en matière de développement spatial ainsi que les impacts des politiques territoriales dans l'Europe élargie. Le programme ESPON a débuté en 2002 et continue jusqu'en 2006 [@BFS].

***Framework* :** ensemble de classes qui collaborent pour réaliser une responsabilité qui dépasse celle de chacune des classes qui y participent [Booch et al., 99].

Information Géographique : information relative à un objet ou à un phénomène du monde terrestre, décrit plus ou moins complètement par sa nature, son aspect, ses caractéristiques diverses, et par son positionnement sur la surface terrestre.

Multiscalaire : qui peut se repérer à différentes échelles géographiques.

Open Source : dont le code source est, soit dans le domaine public, soit détenteur de droits de *copyright* exprimés par une licence *open-source* telle que la Licence Public Générale GNU.

N.U.T.S. : (Nomenclature des Unités Territoriales pour les Statistiques) : définition au niveau européen d'unités régionales emboîtées, permettant le recueil homogène sur l'UE à 15 de plusieurs informations statistiques (attention, il y a eu un changement dans les définitions en 1999).
Les régions ainsi définies sont différentes des régions administratives (les superpositions peuvent être exactes pour certains pays comme la France, ce n'est pas le cas pour tous les membres de l'UE). Pour plus d'informations, voir les pages suivantes :
http://europa.eu.int/comm/eurostat/ramon/nuts/mainchar_regions_fr.html
http://europa.eu.int/comm/eurostat/ramon/nuts/introduction_regions_fr.html
http://europa.eu.int/comm/eurostat/ramon/nuts/basicnuts_regions_fr.html
http://europa.eu.int/comm/eurostat/ramon/nuts/application_regions_fr.html

Phénomène continu : phénomène est qualifié de "continu dans l'espace" s'il recouvre sans discontinuités (vides) cet espace.

Phénomène discret : phénomène est qualifié de "discret dans l'espace" s'il ne recouvre pas de façon continue l'espace étudié.

S.I.G. : (Système d'Information Géographique) : est un ensemble d'outils de collecte, stockage, vérification, recherche, transformation, analyse et affichage de données sur le globe terrestre à référence spatiale. Le S.I.G. est généralement exploité dans un environnement informatique. Les données qu'il manipule sont des informations géographiques qui peuvent y être visualisées et analysées pour fournir un outil d'aide à la décision.

Transaction : en base de données, une transaction est un regroupement d'opérations prenant une base de données dans un état cohérent et la rendant dans un état également cohérent. Pour garantir cette cohérence une transaction doit vérifier plusieurs principes souvent nommés par l'acronyme ACID. Une transaction doit être :

- Atomique : toutes les opérations composant la transaction doivent être exécutées ou aucune ne doit l'être.
- Cohérente : Une transaction assure l'intégrité des données, elle transfère des données d'un état cohérent à un autre. Si la procédure de mise à jour se passe mal, elle restaure les données initiales.
- Isolée : Si plusieurs transactions sont exécutées en même temps, elles ne doivent pas avoir d'interférence. Pour cela, elles sont souvent exécutées séquentiellement.
- Durable : Les résultats obtenus après l'exécution d'une transaction doivent être stockés durablement pour ne pas être affectés par une panne éventuelle du système.

UML : (*Unified Modeling Language*) **:** est, comme son nom l'implique, un langage de modélisation et non une méthode ou un procédé. L'UML est constitué d'une notation très spécifique ainsi que les règles grammaticales s'y attachant pour élaborer des modèles de logiciels. C'est un formalisme standard de modélisation objet conçu par l'OMG.

W3C : World Wide Web Corsortium, Consortium dont le but est de développer des technologies interopérables (spécifications, recommandations, programmes et outils) destinées au Web.

XHTML : langage normalisé par le W3C servant à la publication de pages Web sur Internet. Le XHTML offre les mêmes possibilités que le HTML tout en étant conforme à la norme XML.

XML : (eXtensible Markup Language) : est un métalangage utilisant un système de balises pour représenter des données textuelles. Le terme extensible a été choisi car XML laisse la possibilité aux utilisateurs de définir leurs propres balises.

TABLE DES ILLUSTRATIONS

Figures

Codes

CHAPITRE 1

INTRODUCTION

1.1 Contexte

Depuis ces dernières années, les technologies Web et multimédias ont connu un prodigieux essor et offrent aujourd'hui l'accès à une information riche, diversifiée et facilement accessible. Les possibilités offertes par le Web n'ont donc pas échappé aux différents acteurs de la géographie et de la cartographie car ces technologies permettent de combler le manque d'interactivité d'une carte classique. Aussi, de nombreux sites Web cartographiques ont vu le jour, proposant des cartes dans des domaines très diversifiés : cartes routières interactives, aux cartes météorologiques, topographiques, etc. Ainsi la cartographie Web interactive offre de nouvelles possibilités aux Systèmes d'Information Géographique (S.I.G.) traditionnels, leur permettant de présenter de manière conviviale une grande quantité d'informations. De plus, aujourd'hui, il semble plus prometteur de visualiser des données socio-économiques par l'utilisation de cartes descriptives qu'en traitant ces dernières de manière classique. Cependant, un phénomène peut être décrit par de multiples représentations cartographiques ; il n'est donc pas concevable de générer plusieurs centaines de cartes par avance pour les proposer à l'utilisateur. La solution de ce problème est donc de pouvoir générer des cartes « à la demande », complètement paramétrables en fonction des besoins des utilisateurs. C'est pour répondre à cette demande que le projet *HyperCarte*, sur le lequel porte mon mémoire, a vu le jour. Nous présentons ci-dessous succinctement ce projet.

1.1.1 Le projet HyperCarte

Le contexte de ce mémoire est le projet *HyperCarte*, groupement de recherche fondé en 1996 et ayant à son actif de nombreux travaux et réalisations scientifiques ainsi qu'une reconnaissance nationale et internationale dans le domaine de la cartographie interactive. Il réunit trois équipes de recherche du CNRS, présentées par la suite, autour du thème de l'analyse territoriale multiscalaire et de l'analyse spatiale afin de mettre au point une méthodologie de cartographie interactive et facile d'accès, permettant d'analyser et de visualiser les multiples représentations d'un même phénomène. Le projet *HyperCarte* produit donc des outils ayant pour vocation de permettre l'analyse et la visualisation de données socio-économiques dans un espace d'étude quelconque (un quartier, une ville, un département, une région, un pays, un continent, le monde) par rapport à un espace de référence (par exemple, l'Europe des 15, Europe des 27…) auquel sont comparées les données de l'espace d'étude. Ces analyses peuvent être réalisées en prenant en compte le maillage administratif existant dans le cadre de l'analyse territoriale ou en faisant abstraction de celui-ci et en définissant une portée ou rayon de lissage pour l'analyse spatiale. Les données manipulées sont donc de natures très diverses : informations statistiques (Population, PNB, …), spatiales (localisation), géométriques (contour des divisions administratives)... Le vo-

lume de ces données est donc conséquent et s'accroît proportionnellement au périmètre de l'étude et à la finesse du maillage utilisé pour la collecte de ces informations. A titre d'exemple, nous pouvons noter qu'un fichier contenant un seul indicateur, tel que le recensement de la population, portant sur l'Europe des quinze au niveau communal, représente à lui seul environ 800 Mo. Ceci nous permet de mieux appréhender la taille des données manipulées lorsqu'elles comportent un nombre variable d'indicateurs. Ainsi pour mieux comprendre le projet *HyperCarte* et ses enjeux, nous présentons plus avant ses objectifs.

1.1.2 Les objectifs du projet

Le projet *HyperCarte* vise la mise au point d'une série d'outils interactifs permettant la production, l'interrogation et la représentation cartographique de phénomènes sociaux, en prenant en compte la grande diversité des demandes sociales et politiques adressées à la cartographie statistique. A l'heure actuelle, ni les cartes " papier ", n'offrant qu'une représentation statique d'un phénomène, ni les S.I.G., n'ayant que des capacités d'analyse spatiale limitées, ne répondent à cette problématique. Aussi les membres du projet ont défini le concept d'« *HyperAtlas* » qui repose sur l'hypothèse centrale que toute spatialisation d'un phénomène social peut faire l'objet d'un nombre infini de représentations. Celles-ci dépendent à la fois de la nature intrinsèque des phénomènes sociaux, des hypothèses du concepteur de la carte, des objectifs, des demandes et des pratiques des utilisateurs finaux de l'information cartographique. Le logiciel « *HyperAtlas »* vise la réalisation d'un double objectif, décisionnel et scientifique. Comme tous les S.I.G., le logiciel « *HyperAtlas »* a pour vocation d'être un outil d'aide à la décision qui permet de représenter de manière simple et visuelle une grande quantité d'informations en y adjoignant une localisation spatiale. De plus, il implémente une nouvelle forme de représentation spatiale imaginée par les géographes œuvrant sur ce projet. La combinaison de ces deux aspects fournit aux utilisateurs d'« *HyperAtlas »* des cartes paramétrables et personnalisables représentant les informations qu'ils souhaitent exploiter, permettant, par exemple, aux acteurs de l'aménagement du territoire de mieux appréhender les données socio-économiques de celui-ci. C'est pourquoi, *HyperAtlas* propose un module d'analyse territoriale multiscalaire propre à favoriser l'analyse simultanée, à des échelles[1] différentes, d'un même phénomène social.
Le projet prévoit également la création d'un module d'analyse spatiale multiscalaire qui permettra l'analyse des distributions de phénomènes socio-économiques dans un espace continu. Des procédures de lissage à différentes portées permettront de s'affranchir des maillages territoriaux initiaux et de ne plus se fonder sur l'hypothèse selon laquelle « deux individus localisés à l'intérieur d'une même maille territoriale auront plus de relation que deux individus localisés dans des mailles territoriales différentes ». Une étude des formes sociales fondée sur une fonction d'accessibilité spatio-temporelle permet, au contraire, d'éliminer la grille territoriale de collecte initiale de l'information et de repérer des formes spatiales variables selon la portée des interactions retenues.

Le projet *HyperCarte* se caractérise également par des objectifs scientifiques en termes d'apports de solutions aux diverses problématiques soulevées par la cartographie interactive, l'analyse territoriale multiscalaire et l'analyse spatiale multiscalaire. Trois types de problèmes apparaissent principalement : ceux liés aux données ainsi qu'à leurs traitements, ceux posés par les moyens à mettre en œuvre pour l'accès à ces informations et le problème d'hétérogénéité des utilisateurs potentiels de l'applicatif. Dans le premier cas, la taille des données et les traitements qui leur sont associés, nécessitent de telles ressources que la

[1] Ici la notion d'échelle désigne à la fois l'échelle cartographique et l'échelle géographique (département, région, …).

mise en place d'une architecture distribuée s'avère incontournable. Il faut effectivement dix heures de temps CPU à un PC doté d'un processeur cadencé a 1.2Ghz avec 1Go de mémoire pour calculer la densité de la population européenne avec une fonction d'interaction Gaussienne reposant sur des données communales. Dans le second cas, il est nécessaire de s'interroger sur la mise en place d'une architecture permettant l'accès pour une population hétéroclite, à un nombre d'informations virtuellement infini. Enfin, la diversité des profils des utilisateurs de l'application doit être prise en compte pour fournir à ceux-ci des représentations adaptées à leurs besoins. Ceci nécessite donc la mise œuvre d'un environnement cartographique apte à délivrer rapidement aux utilisateurs, des cartes adaptées à leur besoin, tout en veillant à ce que la restitution de l'information ne soit pas source d'ambiguïtés ou d'erreurs d'interprétation de la part de l'utilisateur. Une détection des profils de ces derniers est donc souhaitée, soit par une déclaration explicite, soit par une analyse des requêtes des utilisateurs en vue de l'établissement de leur profil. Dans ce dernier cas, la complexité réside dans la création de « profils-types » correspondant aux utilisateurs, à l'évolution et à l'adaptabilité de ceux-ci en fonction des besoins et préférences des utilisateurs. Tout ceci devant pouvoir fonctionner le plus indépendamment possible des configurations matérielles et des systèmes qu'utilisent les clients.

Un champ d'application aussi vaste fait donc du projet *HyperCarte* un projet pluridisciplinaire réunissant, à ce titre, plusieurs équipes scientifiques aux compétences variées que nous présentons dans la partie suivante.

1.1.3 Les acteurs du projet

Le projet *HyperCarte* est un projet scientifique réunissant plusieurs équipes de recherche du CNRS autour du thème de l'analyse territoriale et spatiale, multiscalaire :

- L'équipe P.A.R.I.S. du laboratoire U.M.R. Géographie-cités
- L'équipe R.I.A.T.E. du laboratoire U.M.S. RIATE
- Le projet APACHE de l'INRIA[2] Rhône-Alpes
- L'équipe S.I.G.M.A. du laboratoire U.M.R. LSR-IMAG

L'unité mixte de recherche U.M.R. Géographie-cités est composée de plusieurs équipes de recherche autour de différents thèmes d'étude en géographie, tels que " Les Modalités du géographique ", " Système territoriaux et contextes politiques et culturels en Europe "...
L'équipe P.A.R.I.S. (Pour l'Avancement des Recherches en Interaction Spatiale) élabore les concepts géographiques et statistiques mis en œuvre dans le projet *HyperCarte*, au travers de l'un de ses thèmes de recherche portant sur la modélisation et l'analyse spatio-temporelle en géographie.

L'unité mixte de service U.M.S. RIATE a pour vocation de fédérer les compétences scientifiques françaises en matière d'aménagement du territoire européen.
L'équipe R.I.A.T.E. (Réseau Interdisciplinaire pour l'Aménagement du Territoire Européen) assure, de manière plus spécifique, le rôle de référent français pour l'Observatoire en Réseau de l'Aménagement du Territoire Européen (O.R.A.T.E. ou E.S.P.O.N. -traduire European Spatial Planning Observation Network) participe au projet *HyperCarte* dans le cadre de ses objectifs en terme de mise en place d'un serveur cartographique offrant un accès aux bases de données statistiques et cartographiques existantes.

[2] Institut National de Recherche en Informatique et en Automatique

Les équipes travaillant sur le projet APACHE (Algorithmique, Programmation Parallèle et Partage de Charge) proposent une approche originale de la programmation des machines parallèles pour le calcul de haute performance. Performances nécessaires aux traitements et calculs rapides portant sur les nombreuses données manipulées dans le cadre du projet *HyperCarte*.

Enfin l'équipe S.I.G.M.A. (Système d'Information : Ingénierie et MultimédiA) du laboratoire LSR-IMAG, dont les recherches concernent l'ingénierie des systèmes d'information avec des thématiques variées telle que la réutilisation (*e.g.* patrons de conception, objets métiers), les systèmes d'information multimédia et les systèmes d'information géographique. L'axe « Multimédia Web » de l'équipe S.I.G.M.A. a, entre autres, la charge de concevoir et réaliser des outils de représentation et de consultation de données géographiques. C'est au sein de cet axe, bénéficiant d'une solide expérience dans le domaine des S.I.G. pour avoir participé à de nombreux projets de recherche dans le domaine de la cartographie interactive, que ce travail a été réalisé.

1.2 Problématique et contribution

La première version du logiciel *HyperAtlas* intègre les solutions proposées par les différents partenaires du projet. Cette version connaît déjà un certain succès auprès de la communauté scientifique ainsi qu'auprès des divers organismes exploitant des données socio-économiques. Elle a d'ailleurs été retenue et financée par l'Observatoire en Réseau de l'Aménagement du Territoire Européen (ESPON[3]) dans le cadre de son projet ESPON 3.1. Toutefois, *HyperAtlas* a été développé autour d'un jeu de données unique. Et à ce stade il n'était pas possible pour un utilisateur d'intégrer ses propres données ou même d'enrichir celles déjà contenues dans l'application.

Notre objectif sera donc, dans un premier temps, de faire évoluer l'application pour lui permettre de charger à tout moment d'autres jeux de données et d'y intégrer de nouvelles fonctionnalités. Ensuite notre objectif sera de concevoir et réaliser un nouvel outil permettant de créer et gérer simplement des jeux de données pour *HyperAtlas*.

1.3 Organisation du mémoire

Ce mémoire se décompose en cinq chapitres présentant respectivement un état de l'art, une présentation du logiciel *HyperAtlas* ; une description de l'application *HyperAdmin* avant de conclure.

Après notre introduction, le 2 présente au lecteur le concept de système d'information géographique puis décrit comment est représentée l'information dans ces systèmes. Enfin, il présente des outils dédiés à l'analyse spatiale de données socio-économiques.

Dans le troisième chapitre, nous présentons au lecteur la version précédente d'*HyperAtlas* (1.0.0.c). Une fois le lecteur familiarisé avec l'application, ses principes et son fonctionnement, nous décrivons les améliorations apportées à l'application.

[3] ESPON est l'abréviation de « European Spatial Planning Observation Network ».

Le quatrième chapitre s'articule autour d'*HyperAdmin* ; il fournit le cahier des charges utilisé pour sa conception. Puis, il décompose sa réalisation en trois étapes : l'analyse, la conception et sa réalisation.

Finalement, nous concluons sur le travail réalisé au sein de l'équipe SIGMA et présentons un bilan personnel sur celui-ci.

CHAPITRE 2

ETAT DE L'ART

Pour permettre une meilleure compréhension de ce travail, ce chapitre propose un état de l'art sur les systèmes d'information géographique (S.I.G.). Après un bref rappel historique, il présente les notions relatives aux systèmes d'information géographique en abordant les principales fonctionnalités d'un S.I.G., le mode de représentation de l'information géographique. Dans un second temps, il offre un comparatif entre divers outils cartographiques. Enfin, il étudie les diverses solutions intégrées de développement Open Source [@OSI].

2.1 Principes des systèmes d'information géographique

2.1.1 Introduction aux systèmes d'information géographique

Définition

Pour définir un S.I.G. il est nécessaire de définir les deux notions qui le compose : celle de système d'information et celle d'information géographique. L'organisation ISO (*International Standards Organisation*) définit un système d'information de la manière suivante : « système de communication permettant de communiquer et de traiter l'information ». De plus, dans le cas d'un S.I.G., les informations traitées sont des informations géographiques, c'est-à-dire, des informations sur des objets ou phénomènes du monde décrits plus ou moins complètement par leur nature, leur aspect, leurs caractéristiques, … et leur positionnement sur la surface terrestre. Selon B. Bordin [@SEIG], une information peut être qualifiée de géographique lorsqu'elle peut trouver une place sur une carte. Le but du S.I.G. est donc de permettre à un utilisateur d'obtenir diverses informations sur un territoire telles que la proximité de certains objets, la localisation d'un phénomène ou sa superposition avec un autre…

Au cours du temps, plusieurs définitions ont été proposées par les professionnels de la cartographie. Ces définitions mettent en avant deux aspects d'un S.I.G. : sa nature et son objectif. La définition américaine émanant du comité fédéral de coordination inter-agences pour la cartographie numérique (FICCDC, 1988) insiste sur les fonctions techniques d'un S.I.G. : un système d'information géographique est un " *système informatique de matériels, de logiciels, et de processus conçus pour permettre la collecte, la gestion, la manipulation, l'analyse, la modélisation et l'affichage de données à référence spatiale afin de résoudre des problèmes complexes d'aménagement et de gestion* ".

De son côté, l'économiste Michel Didier (1990) met en avant la finalité d'un S.I.G. en proposant la définition suivante : " *ensemble de données repérées dans l'espace, structuré de façon à pouvoir en extraire commodément des synthèses utiles à la décision.* ".

Ces deux définitions se complètent de manière opportune, la première insistant sur les aspects techniques que doit comporter un S.I.G. et la seconde mettant l'accent sur la finalité d'aide à la décision.
Deux fonctions se dégagent alors des S.I.G. :

- La capacité de traiter et gérer les relations spatiales entre objets ou phénomènes sur l'espace terrestre, ce qui implique des fonctions d'analyse spatiale et des facultés de synthèse permettant d'offrir une aide à la décision.
- La capacité de représenter ces informations sous la forme d'une carte, ce qui implique des fonctions de conception et production cartographique.

Ceci nous amène donc naturellement à nous intéresser aux fonctionnalités qu'offre un S.I.G. dans la partie suivante.

Fonctionnalités

Selon Jean Denègre et François Salgé [Denègre et al., 96], pour être complet un S.I.G. doit implémenter cinq fonctionnalités principales : Permettre la saisie (ou Acquisition) de données, le stockage (Archivage) de celles-ci, leur Analyse et leur visualisation (Affichage) en fonction d'un modèle dont le S.I.G. doit rendre compte (Abstraction). Ces principes sont communément appelés les « 5A » et sont représentés par la Figure 2-1 :

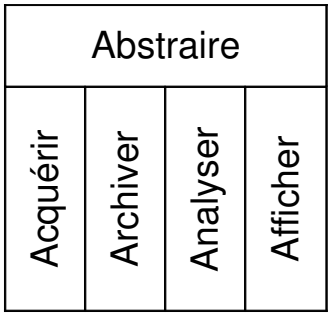


Figure 2-1 : Les « 5A » d'un SIG.

Abstraire : revient à concevoir un modèle qui organise les données par composants géométriques et par attributs descriptifs et aussi à établir des relations entre les objets.

Acquérir : revient à alimenter le S.I.G. en données. Les fonctions d'acquisition consistent à entrer d'une part la forme des objets géographiques et d'autre part leurs attributs et relations.

Archiver : consiste à transférer les données de l'espace de travail vers l'espace d'archivage (disque dur).

Analyser : permet de répondre aux questions posées.

Afficher : pour produire des cartes de façon automatique, pour percevoir les relations spatiales entre les objets, pour visualiser les données sur les écrans des ordinateurs.

2.1.2 Représentation de l'information dans les S.I.G.

L'information stockée (archivée) dans un S.I.G. est de deux natures distinctes : les données spatiales nécessaires à la représentation de l'espace (affichage de cartes) et les données thé-

matiques représentant des informations, souvent de nature statistique, relatives à un thème. Par exemple, " la population des départements français en 1999 " correspond à une information thématique alors que les contours des départements sont des informations spatiales. Le traitement de ces deux types d'informations par un S.I.G. permet, par exemple, la génération d'une carte choroplèthe rendant compte visuellement des écarts existants dans les différents départements.

Deux modes de représentation de l'information spatiale sont distingués : le mode point, ou « Raster », et le mode vectoriel.
En mode « raster », une surface est représentée par un ensemble de points, également appelé *pixels*, doté d'une couleur donnée. Les données vectorielles se présentent sous forme d'éléments géométriques (lignes, courbes, surfaces) repérés par leurs coordonnées comme le montre la Figure 2-2.

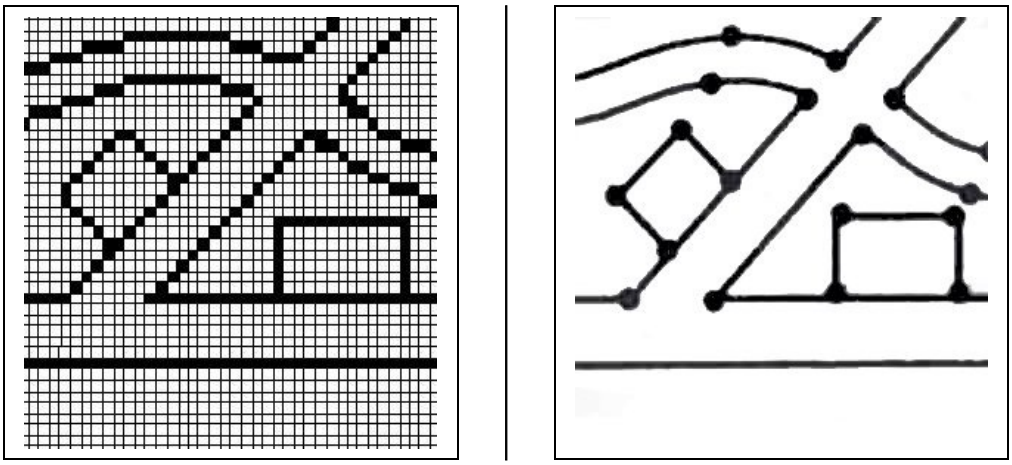


Figure 2-2 : Deux représentations différentes d'une même topologie, à gauche en mode « raster » et à droite en mode vectoriel.

Il est également intéressant de noter que ces deux modes peuvent être combinés pour obtenir une information plus précise et plus lisible comme le montre la Figure 2-3. Ces superpositions sont notamment utilisées par certains sites Web de calcul d'itinéraires comme par exemple www.mappy.com, pour superposer une photo aérienne (raster) à un plan vectoriel.

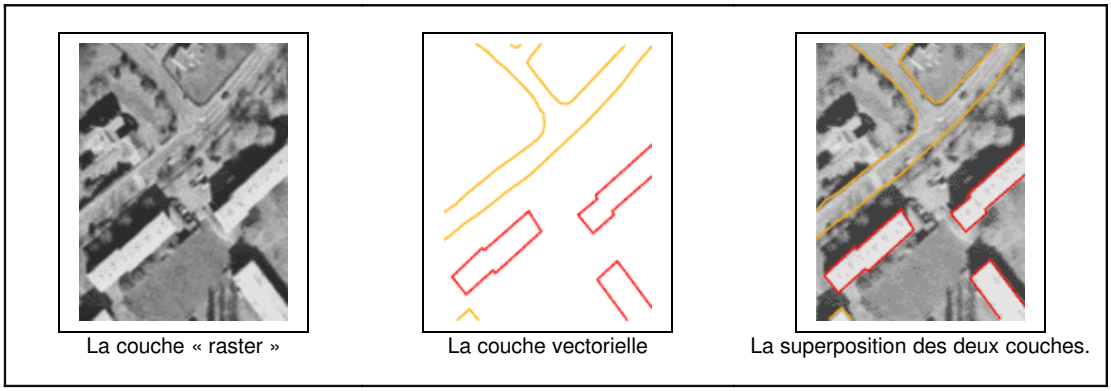


Figure 2-3 : Exemple de superposition de données vectorielles et de données « raster » permettant un enrichissement de la représentation.

2.1.3 Les normes de l'O.G.C.

Fondée en 1994, l' « Open Geospatial Consortium » (O.G.C.) est une organisation internationale à but non lucratif regroupant divers acteurs du domaine des technologies de l'-

information géographique (agences gouvernementales, éditeurs de logiciels S.I.G., universités …). L'O.G.C. a pour vocation de définir des standards autour de l'information géoréférencée et des services à base de localisation (Location Based Services). Les spécifications qu'il propose permettent donc de résoudre les problèmes d'incompatibilité des données constituant l'information géographique et les problèmes d'interopérabilité des systèmes, problèmes soulignés par l'ensemble de la communauté géomatique. Ainsi, l'objectif de l'O.G.C. est de rendre les systèmes d'information géographique inter-communicables et interopérables, au travers de prescriptions techniques traduites dans ses spécifications, accessibles gratuitement sur son site Web : http://www.opengeospatial.org/. Lorsque l'on étudie ces standards et les documents s'y rapportant nous retrouvons fréquemment le terme « OpenGIS ». Ce terme est un adjectif décrivant les spécifications et autres produits issus des travaux de l'O.G.C., supportant à ce titre, un accès transparent, via un réseau, vers des données géoréférencées hétérogènes ainsi qu'aux ressources de traitement de ces données. Notons cependant que l'on appelle souvent l'O.G.C., « l'OpenGIS Consortium », par abus de langage.

Parmi les spécifications mises à disposition par l'O.G.C., nous allons plus particulièrement nous intéresser aux spécifications définissant les données géoréférencées et leurs représentations dans un système compatible OpenGIS. Celles-ci sont donc présentées au travers de la spécification intitulée : « *OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1 : Common architecture* » [@OGC]. Cette spécification de l'O.G.C. définit, en s'appuyant sur le formalisme UML, divers types de données géoréférencées ainsi qu'un modèle d'implémentation de ces géométries. Dans ce modèle, chaque objet géométrique dispose des propriétés générales suivantes :

- Il est associé à un système de référence spatiale (*Spatial Reference System*), qui décrit l'origine des coordonnées de l'espace.
- Il appartient à une classe géométrique.

La Figure 2-4 extraite de cette spécification montre la hiérarchie des classes géométriques, nous remarquons que la classe de base est la classe « *Geometry* » qui sert de classe parente à toutes les autres géométries telles que les points, courbes, polygones… Ce modèle géométrique étendu définit également des classes de collection à 0, 1 ou 2 dimensions permettant la représentation de géométries composées d'autres géométries. C'est le cas notamment des classes « *MultiPoint* », « *MultiLineString* » et « *MultiPolygon* » qui sont respectivement des collections de « *Point* », « *LineString* » et « *Polygon* »

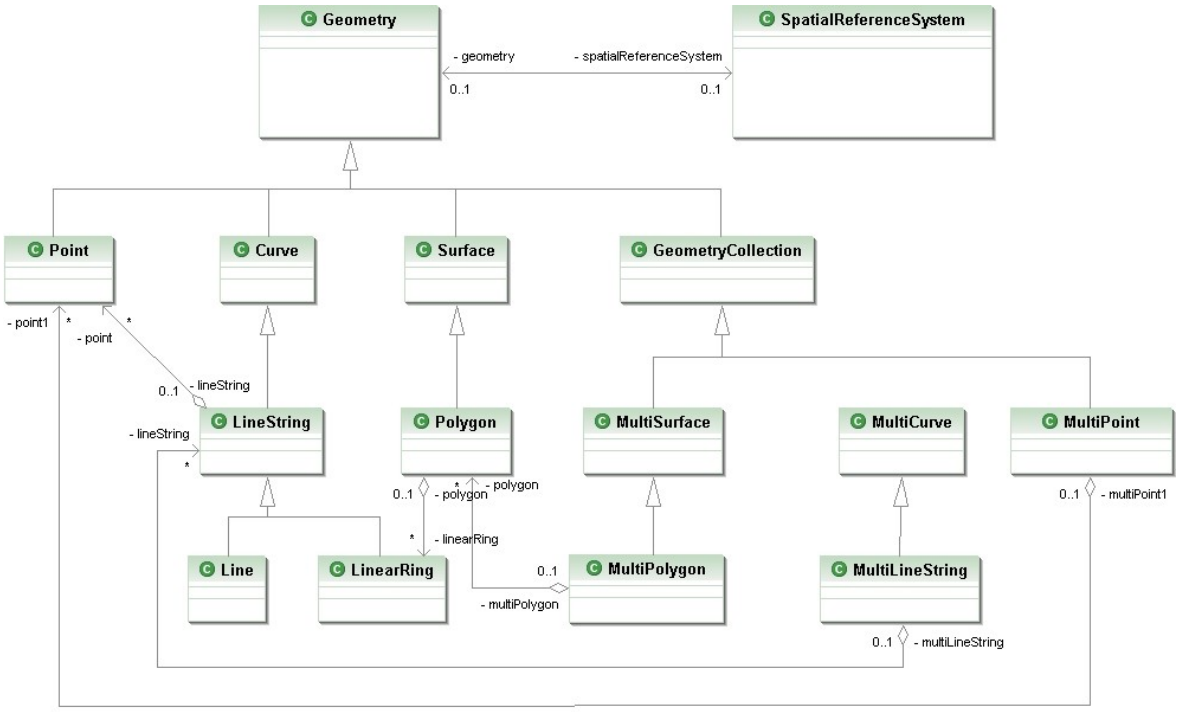
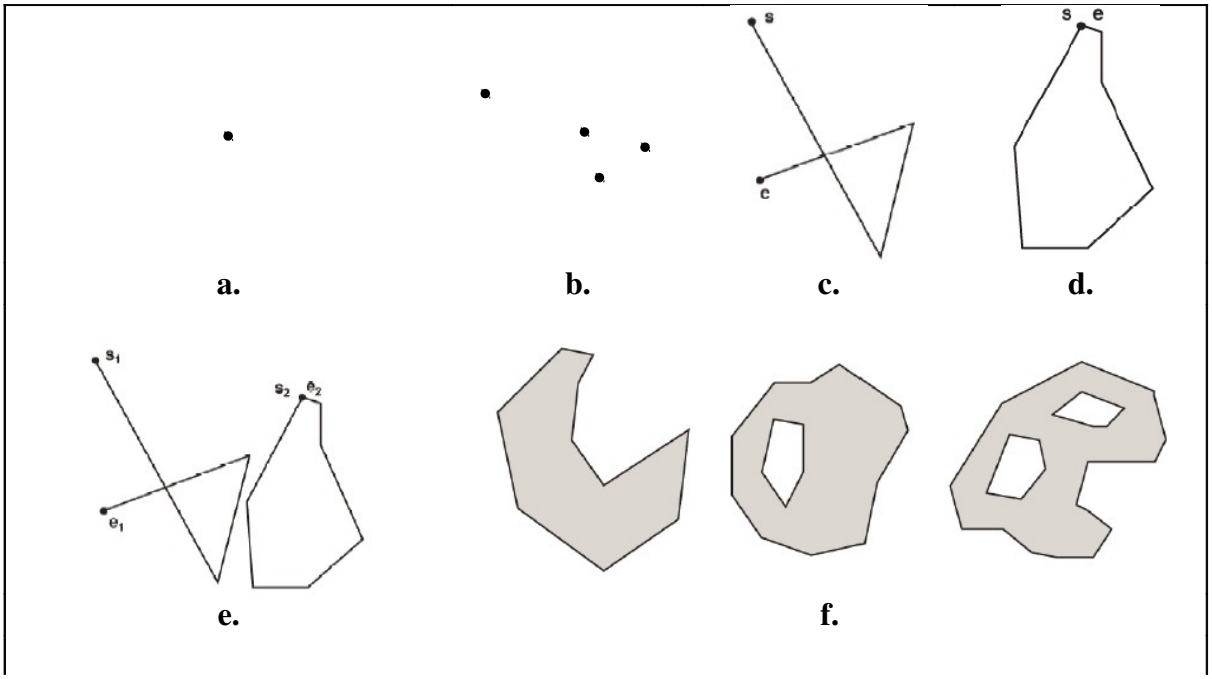
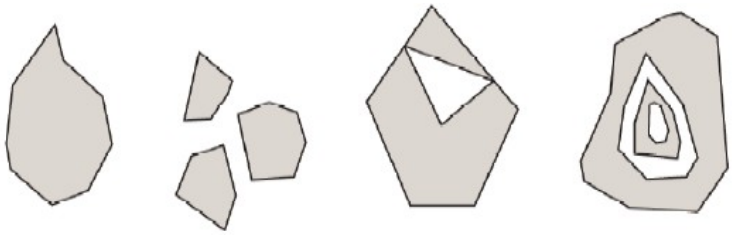


Figure 2-4 : Hiérarchie des classes géométriques.

Les classes « *MultiSurface* » et « *MultiCurve* » sont des classes abstraites généralisant les collections de courbes et surfaces. Nous proposons maintenant au lecteur un bref aperçu de ces géométries et de leurs principales méthodes, Une description en détail de celles-ci pouvant être trouvées sur le site de l'O.G.C. sous la rubrique « spécifications ». Nous allons commencer ce descriptif par la présentation des principales géométries supportées puis nous nous intéresserons à leurs principales méthodes.





g.

Légende :

a. Point
b. MultiPoint
c. LineString
d. **LinearRing (une LineString close, dont le dernier point est identique au point de départ)**
e. MultiLineString
f. Polygon
g. MultiPolygon

Figure 2-5 : Un exemple des diverses géométries définies dans la spécification de l'OGC.

Toutes ces géométries ont pour ancêtre la classe abstraite « *Geometry* » qui définit un ensemble de méthodes communes à toutes les géométries. Chaque sous-classe (type de géométrie) doit donc implémenter chacune des méthodes abstraites de manière à répondre à leurs spécifications. Deux types de méthodes peuvent être dénombrés : les méthodes génériques et les méthodes testant les relations spatiales entre géométries.

Les méthodes suivantes donnent des indications sur la géométrie en elle-même :

- **Dimension** ( ): Integer — Retourne la dimension de l'objet qui doit être inférieure ou égale aux dimensions de ses coordonnées. Notons que cette spécification de l'O.G.C. se limite aux géométries existantes dans un espace à 2 dimensions.

- **GeometryType** ( ): String — Retourne le nom du sous-type instanciable de la géométrie représenté par l'instance.

- **SRID** ( ): Integer — Retourne l'identifiant du système de référence spatial (projection géographique) dans lequel les coordonnées de l'objet sont exprimées.

- **Envelope**(): Geometry — Fournit le rectangle englobant minimal de la géométrie sous la forme d'un polygone dont les coordonnées sont les suivantes : [(x_{min}, y_{min}), (x_{max}, y_{min}), (x_{max}, y_{max}),(x_{min}, y_{max}), (x_{min}, y_{min})] où x_{min} est l'abscisse minimale de l'ensemble des coordonnées de la surface, y_{min} est l'ordonnée minimale de la surface, x_{max} est l'abscisse maximale et y_{max} son ordonnée maximale.

- **AsText**( ): String — Fournit une représentation textuelle de la géométrie selon le formalisme WKT[4].

[4] WKT : Well Known Text est un format de représentation textuel d'une géométrie, ce format est défini par l'OGC.

- **AsBinary**( ):Binary — Exporte la figure sous la forme d'une WKB[5].

- **IsEmpty**( ):Integer — Retourne 1 (TRUE) si l'objet géométrique représente une géométrie « vide ». C'est-à-dire si la géométrie est composée d'un ensemble vide de point, ∅, dans l'espace.

- **IsSimple**( ):Integer — Retourne 1 (Vrai) si cette (*this*) géométrie n'a pas de point ambigu tel que des intersections ou tangentes avec elle même. Chaque classe implémentant cette interface doit fournir les conditions spécifiques de cette simplicité.

- **Boundary**( ):Geometry — Retourne la bordure fermée de la géométrie, c'est-à-dire un ensemble de géométries pour lesquelles le premier et le dernier point sont identiques.

Outre ces méthodes de base, d'autres permettent de tester les diverses relations spatiales entre les géométries ; parmi celles-ci, nous pouvons noter les méthodes suivantes :

- **Equals(anotherGeometry:Geometry):Integer** — Retourne 1 (Vrai) si les deux géométries sont "spatialement égales".

- **Disjoint(anotherGeometry:Geometry):Integer** — Retourne 1 (Vrai) si les géométries sont disjointes.

- **Intersects(anotherGeometry:Geometry):Integer** — Retourne 1 (Vrai) si les géométries ont une ou plusieurs intersections. Nous pouvons également noter que pour deux géométries « a » et « b » nous avons la relation suivante : `a.Intersects(b)  ! a.Disjoint(b)`

- **Touches(anotherGeometry:Geometry):Integer** — Retourne 1 (Vrai) si les géométries se touchent.

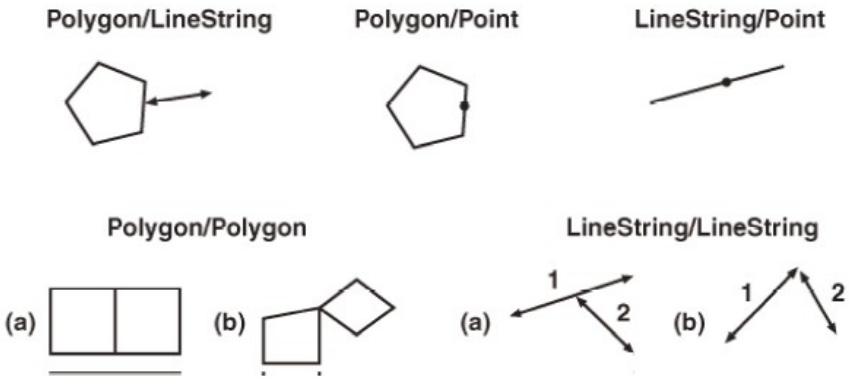


Figure 2-6 : Exemples de la relation « touch » entre diverses géométries.

- **Crosses(anotherGeometry:Geometry):Integer** — Retourne 1 (Vrai) si les géométries se croisent.

[5] WKB : Well Known Binary, format binaire standardisé de l'OGC pour la représentation d'une géométrie.

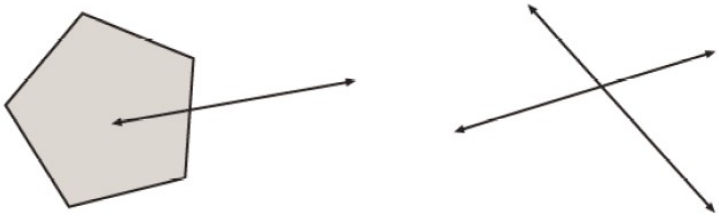


Figure 2-7 : Deux exemples de la relation de croisement entre deux géométries : à gauche entre un objet de type Polygon et une LineString et à droite entre deux objets de type LineString.

- **Within(anotherGeometry:Geometry):Integer** — Retourne 1 (Vrai) si la géométrie est contenue dans la géométrie passée en paramètre de la méthode.

Figure 2-8 : Deux exemples de la relation de contenance, entre deux polygones à gauche et deux LineString à droite.

- **Contains(anotherGeometry:Geometry):Integer** — Retourne 1 (Vrai) si la géométrie contient la géométrie passée en paramètre de la méthode. Nous pouvons également noter que : a.Contains(b) ⇔ b.Within(a)

- **Overlaps(anotherGeometry:Geometry):Integer** — Retourne 1 (Vrai) si la géométrie "chevauche spatialement" la géométrie passée en paramètre (anotherGeometry).

- **Relate(anotherGeom:Geometry, intersectionPatternMatrix:String) : Integer** — Retourne 1 (Vrai) si la géométrie est connexe à la géométrie passée en paramètre. Ce teste s'effectue en testant les intersections entre les bordures internes et externes des deux géométries comme spécifié par les valeurs contenues dans la matrice d'intersection fournie.

Les Formats d'échange

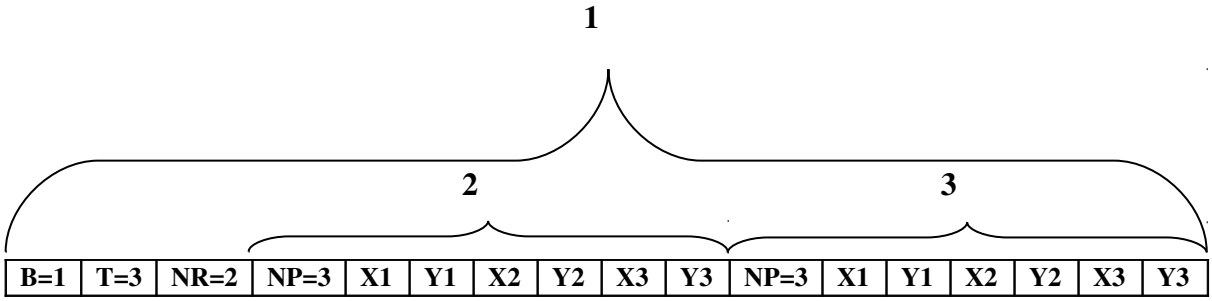
Rappelons que l'un des premiers objectifs de l'OGC est de définir des standards permettant l'interopérabilité au travers de systèmes hétérogènes. A ce titre, il a défini dans sa spécification deux formats d'échange de données. Le premier est un format binaire et le second un format textuel intelligible par un être humain. Ces deux formats sont les formats WKB[6] et WKT[7].

Le format WKB

Ce format a pour objectif de fournir un format d'échange binaire compact pour la représentation des objets géométriques décrits précédemment. Les flux d'octets obtenus peuvent, par exemple, être envoyés à destination d'une base de données ou d'un programme compatible avec la norme et être rassemblés en géométries.

[6] Abréviation de « *Well Known Binary* » traduisible par binaire bien connu.
[7] Abréviation de « *Well Known Text* » traduisible par texte bien connu.

Il se base sur des données numériques pour être le plus compact possible, lorsque l'on souhaite échanger ce type de données entre des systèmes hétérogènes il est nécessaire de préciser quel format est utilisé pour les encoder. Ainsi le premier octet du flux définit l'encodage utilisé (Big Indian ou Little Indian). Dés lors, il est possible d'envoyer les informations propres à la géométrie telles que son type et sa composition. La Figure 2-9 présente un exemple de géométrie encodé selon la grammaire WKB.



Légende :

1- polygone WKB
2- premier ring.
3- second ring

Figure 2-9 : Représentation WKB d'une géométrie codée en « Little Indian » (B=1), de type polygone (T=3), formé par deux LinearRing (NR=2) de 3 points chacun (NP=3)

Notons que la liste exhaustive des champs et valeurs nécessaires à la représentation WKB d'une géométrie peut être trouvée dans la spécification intitulée : « *OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1:Common Architecture* » [@OGC].

Le format WKT

Chaque type de géométrie a également une représentation textuelle utilisant une norme définie et peut être créé via une chaîne de caractères. Nous présentons le langage servant à la description des géométries. La notation {}* signifie qu'il y a zéro ou plusieurs répétitions de la sous chaîne contenue entre accolades. Les accolades n'apparaissent pas dans la chaîne résultante. Les chaînes de caractères respectant la grammaire WKT présentée par la Figure 2-10 permettent l'instanciation d'une géométrie.

```
<Geometry Tagged Text> :=
<Point Tagged Text>
           | <LineString Tagged Text>
           | <Polygon Tagged Text>
           | <MultiPoint Tagged Text>
           | <MultiLineString Tagged Text>
           | <MultiPolygon Tagged Text>
           | <GeometryCollection Tagged Text>
<Point Tagged Text> :=
           POINT <Point Text>
<LineString Tagged Text> :=
           LINESTRING <LineString Text>
<Polygon Tagged Text> :=
           POLYGON <Polygon Text>
<MultiPoint Tagged Text> :=
           MULTIPOINT <Multipoint Text>
<MultiLineString Tagged Text> :=
           MULTILINESTRING <MultiLineString Text>
<MultiPolygon Tagged Text> :=
           MULTIPOLYGON <MultiPolygon Text>
<GeometryCollection Tagged Text> :=
```

```
          GEOMETRYCOLLECTION <GeometryCollection Text>
<Point Text> := EMPTY | ( <Point> )
<Point> := <x> <y>
<x> := double precision literal
<y> := double precision literal
<LineString Text> := EMPTY
          | ( <Point > {, <Point > }* )
<Polygon Text> := EMPTY
          | ( <LineString Text > {, < LineString Text > }*)
<Multipoint Text> := EMPTY
          | ( <Point Text > {, <Point Text > }* )
<MultiLineString Text> := EMPTY
          | ( <LineString Text > {, < LineString Text > }* )
<MultiPolygon Text> := EMPTY
          | ( < Polygon Text > {, < Polygon Text > }* )
<GeometryCollection Text> := EMPTY
          | ( <Geometry Tagged Text> {, <Geometry Tagged Text> }* )
```

Figure 2-10 : Grammaire définissant la norme WKT.

Pour illustrer la grammaire présentée ci-dessus, le Tableau 2-1 offre des exemples de représentation textuelle utilisant cette grammaire et permettant au lecteur de constater que les chaînes résultantes sont parfaitement compréhensibles.

Geometry Type	Représentation littérale Textuelle	Commentaire
Point	'POINT (10 10)'	Un point
LineString	'LINESTRING (10 10, 20 20, 30 40)'	Une LineString de 3 points
Polygon	'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'	Un polygone sans trou.
Multipoint	'MULTIPOINT (10 10, 20 20)'	Un multipoint de 2 points
MultiLineString	'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'	Une MultiLineString composée de 2 LineString
MultiPolygon	'MULTIPOLYGON (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))'	Un MultiPolygon avec 2 polygones
GeomCollection	'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'	Une GeomCollection composée de 2 points et 1 LineString

Tableau 2-1 : Exemple de représentations textuelles de géométries suivant la norme WKT.

2.2 Outils Cartographiques d'analyse spatiale

Ce chapitre présente une liste non exhaustive d'outils d'analyse spatiale de phénomène socio-économique. Pour comparer ces outils, nous allons nous intéresser à plusieurs points pour guider notre comparaison. Ainsi, nous allons d'abord étudier les possibilités offertes pour l'analyse en décrivant les cartes proposées ainsi que les traitements qu'il est possible de réaliser sur les indicateurs statistiques. Ensuite, nous étudions les fonctionnalités de navigation dans la carte pour déterminer, par exemple, si des déplacements sont possibles ou quels sont les types de zoom proposés. Enfin, nous verrons quelles possibilités de personnalisations sont possibles.

2.2.1 Le logiciel JCT, Java Cartes Thématiques

JCT est un logiciel écrit en Java permettant de réaliser des cartes thématiques, initialement écrit par deux étudiants de l'EIVD, Ecole d'ingénieurs du canton de Vaud (Suisse), pour le compte du DFJ[8]. La version 1.0 est actuellement disponible en libre téléchargement, sous la licence GNU/GPL, à l'adresse suivante :

http://jct.sourceforge.net/

JCT permet l'affichage de cartes dont les données géométriques et statistiques sont contenues dans des fichiers portant l'extension « JCT ». Comme le montre la Figure 2-11, JCT permet de présenter une carte choroplèthe, c'est-à-dire, une carte présentant un ensemble d'unités territoriales colorisées en fonction d'une valeur donnée ; dans le cas présent cette valeur représente soit un indicateur ou le résultat du ratio entre deux indicateurs.

[8] DFJ : Département pour la Formation et la Jeunesse (Suisse).

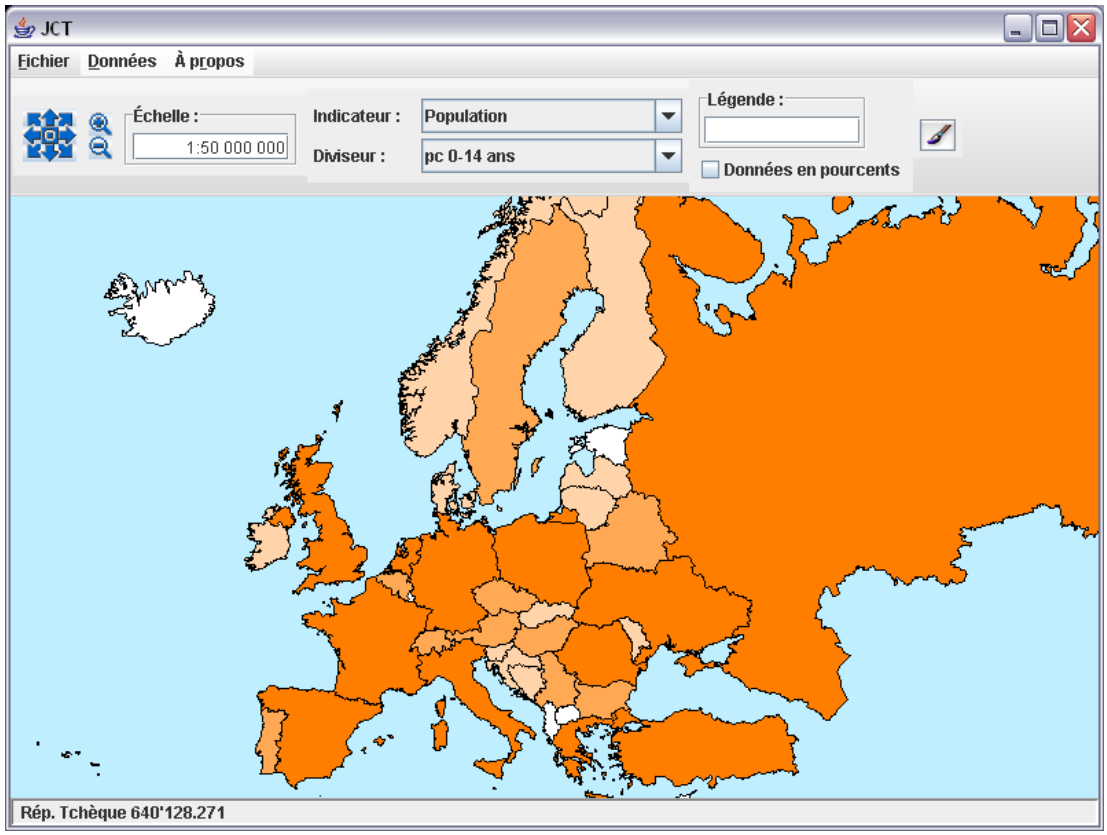


Figure 2-11: Copie d'écran de l'application JCT affichant une carte choroplèthe représentant le ratio entre deux indicateurs.

Possibilités offertes pour l'analyse spatiale

JCT, permet de générer une carte présentant un indicateur mais il permet aussi de calculer un taux permettant de faire le ratio entre deux indicateurs. Il permet donc de choisir un numérateur et un dénominateur pour présenter le résultat du ratio sur la carte choroplèthe.

Possibilités de navigation dans la carte

Il est possible de zoomer et de ce déplacer dans la carte grâce aux contrôles situés en haut à gauche de la fenêtre. Il est également possible de zoomer sur une sélection rectangulaire définie avec la souris.

Personnalisations possibles

Pour personnaliser la carte qu'il vient de produire, un utilisateur, peut choisir le nombre d'intervalles utilisé par la carte ainsi que la couleur à utiliser. De plus, il est possible de choisir la méthode de discrétisation utilisée pour le calcul des intervalles.
En revanche, il n'est pas possible pour un utilisateur d'ajouter ou de modifier les données chargées dans le logiciel. Notons que les jeux de données utilisés sont dans un format propriétaire et que certains sont disponibles sur le site Web dédié au logiciel.

Malheureusement certains bugs sont à déplorer comme une mauvaise gestion du nombre de couleurs (classes) lors d'une personnalisation de celles-ci, comme le montre la Figure 2-12.

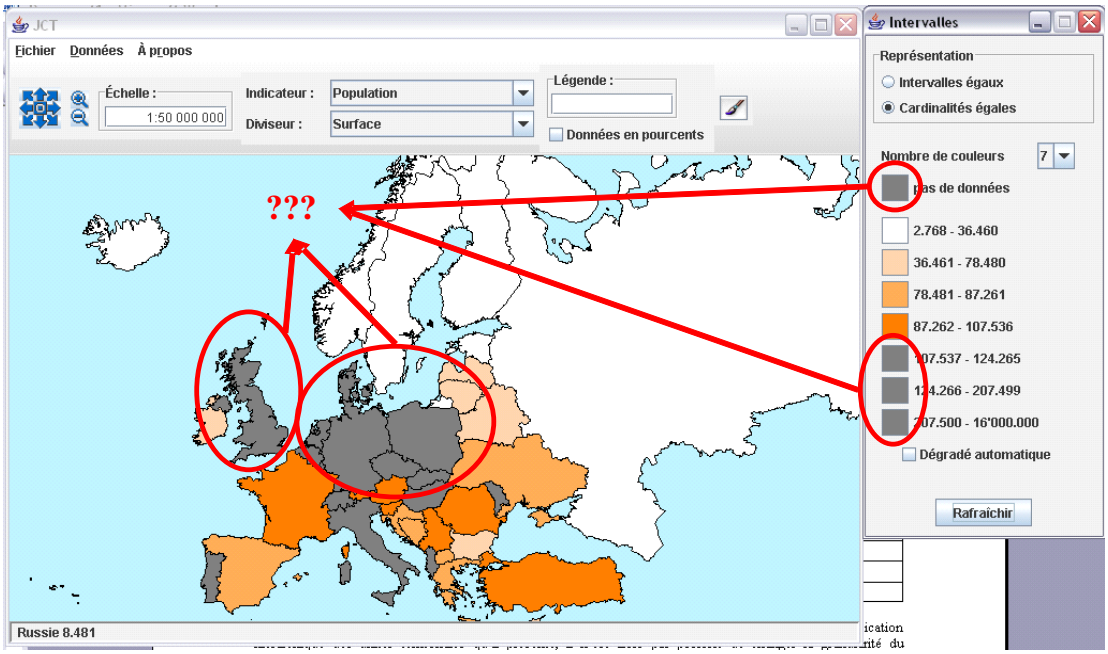


Figure 2-12 : Valeurs manquantes lors du changement du nombre de tons de la palettes dans l'application JCT.

Conclusion

Il est important de noter que le logiciel JCT n'a pas bénéficié du même nombre années d'existence que d'autres logiciels. En conclusion, bien que présentant moins de fonctionnalités que certains de ses concurrents, JCT reste prometteur surtout si l'on tient compte de la « jeunesse » de l'application, version 1.0 datant du 7 juillet 2004. Il a d'ailleurs été primé en septembre 2005 comme 3ème meilleur logiciel étudiant par le site MacGénération (http://trophee.macgeneration.com/).

2.2.2 GeoClip

GeoClip est un Système d'Information Géographique développé par la société « E=MC3 ». Accessible depuis le Web, il permet de réaliser des cartes affichables depuis n'importe quel site Web. La première chose que peut remarquer une personne qui essaie cet outil, est la qualité et l'ergonomie de son interface utilisateur (voir Figure 2-13). Ses principales fonctions sont accessibles via une barre d'outils placée en haut et à droite de la carte et via une zone de configuration placée à droite de la carte. Notons que les zooms et les déplacements se font via l'utilisation d'une mini carte représentant la zone étudiée.

Une version d'essai en ligne est disponible à l'adresse : http://www.*GeoClip*.fr/danseuse/carto.php?lang=fr

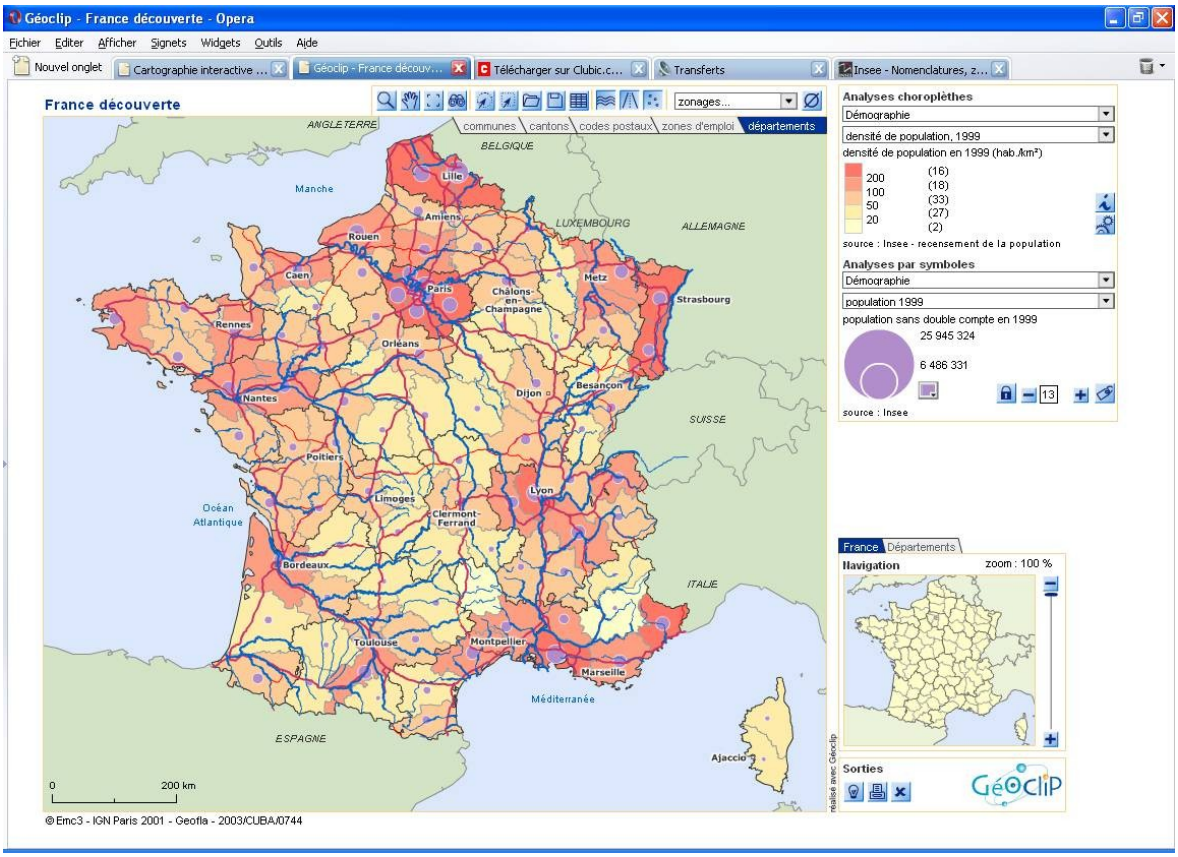


Figure 2-13 : copie d'écran de l'application *GeoClip*, disponible sur le site de la société E=MC3.

Cette version permet de créer des cartes choroplètes ainsi que des cartes à disques proportionnels sur les différents niveaux de maillages territoriaux de la France. Il est également possible de visualiser plusieurs couches d'informations pour faire apparaître le réseau routier, les fleuves et les villes.

Possibilités offertes pour l'analyse spatiale

Dans la version proposée en démonstration sur le site il est possible de créer trois type de carte : une carte choroplète, une carte à symboles proportionnels et le mélange des deux premier types.
Le choix du type de carte généré dépend simplement du renseignement des champs permettant le choix des indicateurs. Ceux-ci sont, par ailleurs, regroupés selon des thèmes ce qui permet de faciliter leur consultation.

Notons que dans le cas de *GeoClip* il n'est pas possible pour un utilisateur de créer ses propres ratios, un utilisateur ne peut que choisir parmi les indicateurs proposés. Même si cette solution réduit la souplesse de l'application, elle a pour avantage de réduire l'éventail des choix possibles à une sélection de ratios jugés pertinents.

Possibilités de navigation dans la carte

Parmi les fonctionnalités de navigation il est possible de se déplacer soit par l'utilisation de « cliquer-glisser » sur la carte, soit en déplaçant la zone affichée dans la mini-carte présente en bas à droite de la fenêtre. Le zoom se fait soit par l'établissement d'une sélection rectangulaire, soit par l'utilisation de boutons permettant le zoom avant ou arrière. Il est également possible de sélectionner une ou plusieurs unités pour afficher un tableau présentant les valeurs de l'indicateur pour les unités sélectionnées (*cf.* Figure 2-14).

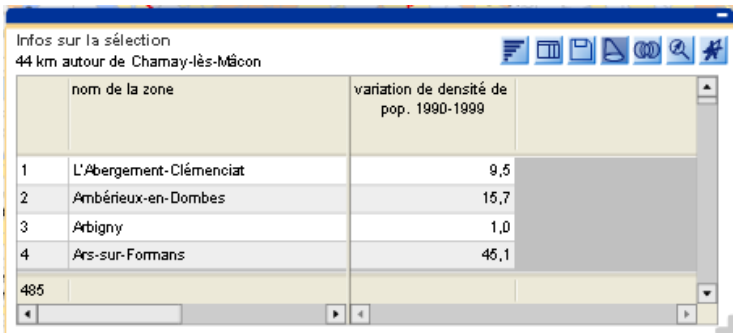


Figure 2-14 : copie d'écran du tableau généré par *GeoClip* pour lister les valeurs des unités sélectionnée.

Personnalisations possibles

La carte générée par l'application peut être personnalisée par le choix des couleurs de fond utilisées, ainsi que par le nombre d'intervalles permettant la discrétisation. De plus, l'utilisateur peut éditer les seuils délimitant les intervalles de discrétisation. Il peut également choisir de ne colorer que les unités territoriales correspondantes à une plage de valeur données.

Une fonctionnalité permet également d'intégrer et de traiter ses propres données statistiques. Cet import ce fait simplement par l'utilisation d'un « copier-coller » depuis un tableau Excel vers un dialogue de l'application. Une fonction d'export des données au format XML est également disponible.

Conclusion

Bien qu'assez complet et très ergonomique, *GeoClip* utilise une technologie employée souffrant d'une limitation quant au nombre d'objets qu'il est possible de d'afficher simultanément. Pour palier cette limitation lorsqu'il est nécessaire de travailler sur des cartes contenant plus de 6000 objets (limite annoncée par l'éditeur) la carte est représentée avec un niveau de zoom ne permettant pas de l'afficher dans son intégralité mais seulement par portions. Le déplacement en dehors de l'une de ces portions entraîne le rechargement des données et un nouveau tracé de la carte.

2.2.3 Le *Cartographeur*

Développé par la société Articque (http://www.articque.com), le logiciel « *Cartographeur* » sous ses différentes déclinaisons (gratuite, medium, premium et plus) offre un outil puissant pour la génération de cartes de statistiques, comme le montre les captures d'écrans présentées dans la Figure 2-15.

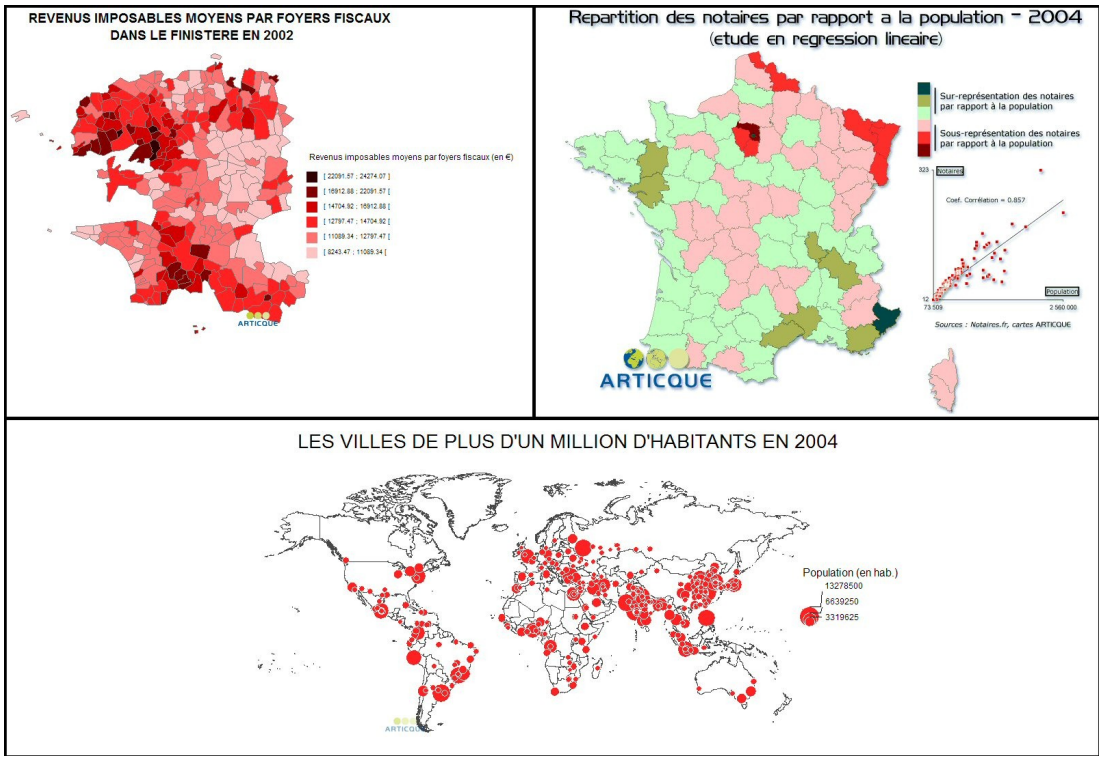


Figure 2-15 : Un extrait de la richesse de représentation cartographique possible dans le logiciel « *Cartographeur* ».

Une version gratuite est disponible à l'adresse :

http://www.articque.com/FR04/PROD/carto.htm

Possibilités offertes pour l'analyse spatiale

Cet outil permet de générer une grande diversité de cartes mais le *Cartographeur* à davantage vocation à créer des cartes imprimable plutôt que de servir d'outil de visualisation et de traitement de cartes. Bien qu'étant assez complet *Cartographeur* est difficilement exploitable pour la consultation de cartes qui impose à un utilisateur de franchir un nombre important d'étapes pour générer une carte. Un utilisateur souhaitant générer une carte doit en effet, réaliser les opérations suivantes : choisir un fond de carte, importer un ou plusieurs jeux de données, entrer manuellement les formules à appliquer sur ses données, choisir un mode de remplissage et lier tout ceci à un mode de visualisation…
La capture d'écran de la Figure 2-16, montre l'arborescence qu'il est nécessaire de réaliser pour créer une carte (partie gauche de l'image).

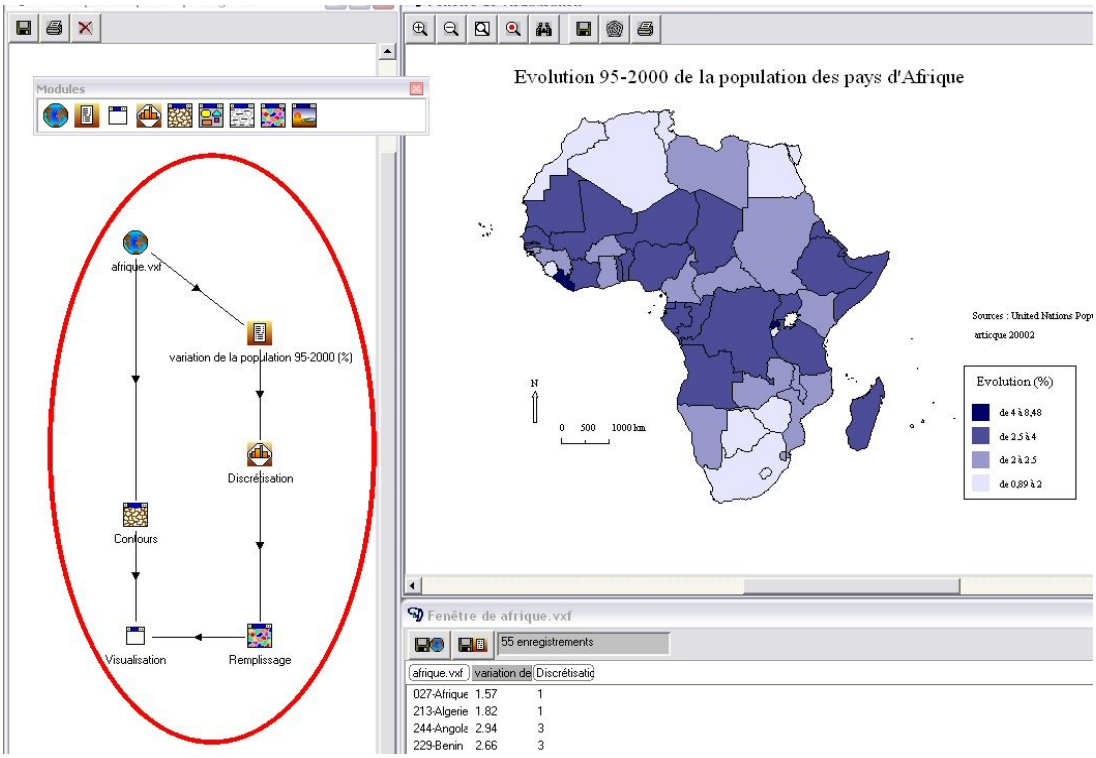
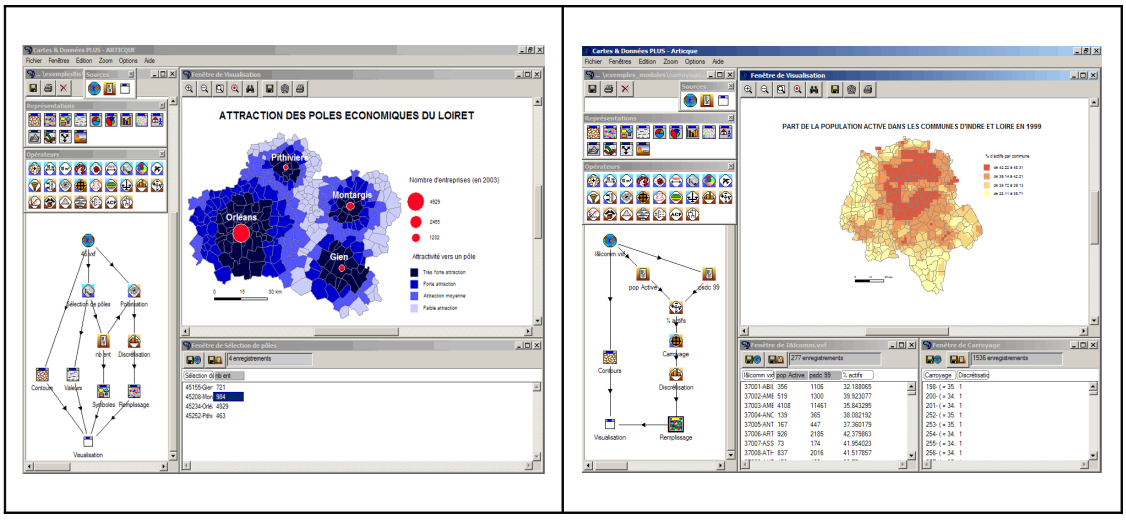


Figure 2-16 : Ici entouré, l'arbre des modules que doit composer l'utilisateur pour créer et visualiser une carte.

Dans ce contexte, générer à la volée une série de cartes devient vite fastidieux et peu convivial pour un utilisateur peu expérimenté. En revanche, ce système laisse une grande autonomie à un utilisateur confirmé, dans le paramétrage de ses cartes, lui permettant de réaliser les cartes qui lui conviennent le mieux.
De plus, il faut noter que des modules supplémentaires sont fournis dans les déclinaisons payantes du logiciel. C'est le cas, par exemple, pour les modules de polarisation, de visualisation 3D, de carroyage, ... Cependant la complexité de mise en œuvre semble croître proportionnellement à la puissance de représentation offerte par ces modules, comme le montrent la Figure 2-17 et la Figure 2-18.



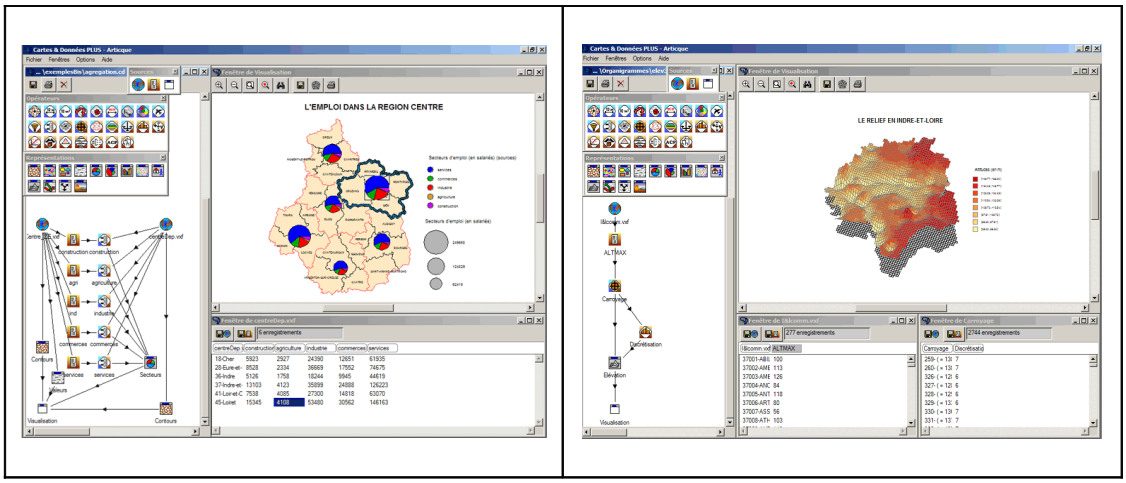


Figure 2-17 : copies d'écran représentant de gauche à droite et de haut en bas les modules de : polarisation, de carroyage, d'agrégation et de visualisation 3D.

La construction de certaines cartes nécessite l'imbrication de modules assez complexes et rend l'utilisation de ce logiciel assez peu intuitive, comme le montre la Figure 2-18 représentant l'imbrication de module à réaliser pour exploiter le module d'agrégation.

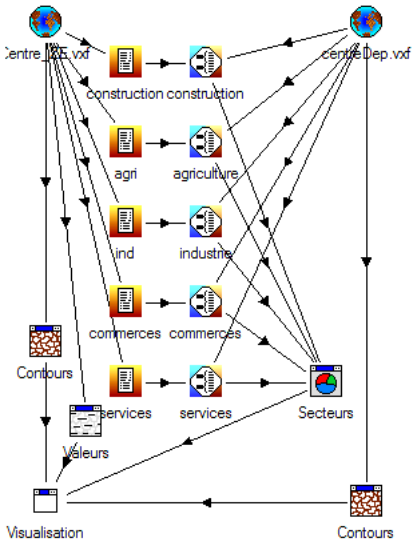


Figure 2-18 : l'imbrication de module nécessaire pour créer une carte exploitant le module d'agrégation.

Possibilités de navigation dans la carte

Comme nous l'avons vu, le *Cartographeur* à pour objectif de produire des cartes imprimables. Aussi, aucune possibilité de déplacement ou de zoom n'est permise par l'application.

Personnalisations possibles

Bien entendu, les possibilités de personnalisation varient en fonction de la carte produite, cependant l'outil laisse à l'utilisateur la possibilité de personnaliser chaque paramètre de représentation de la carte : couleurs de fond, couleurs et épaisseur des contours, choix et taille des formes pour les cartes à formes proportionnelles, etc.

Conclusion

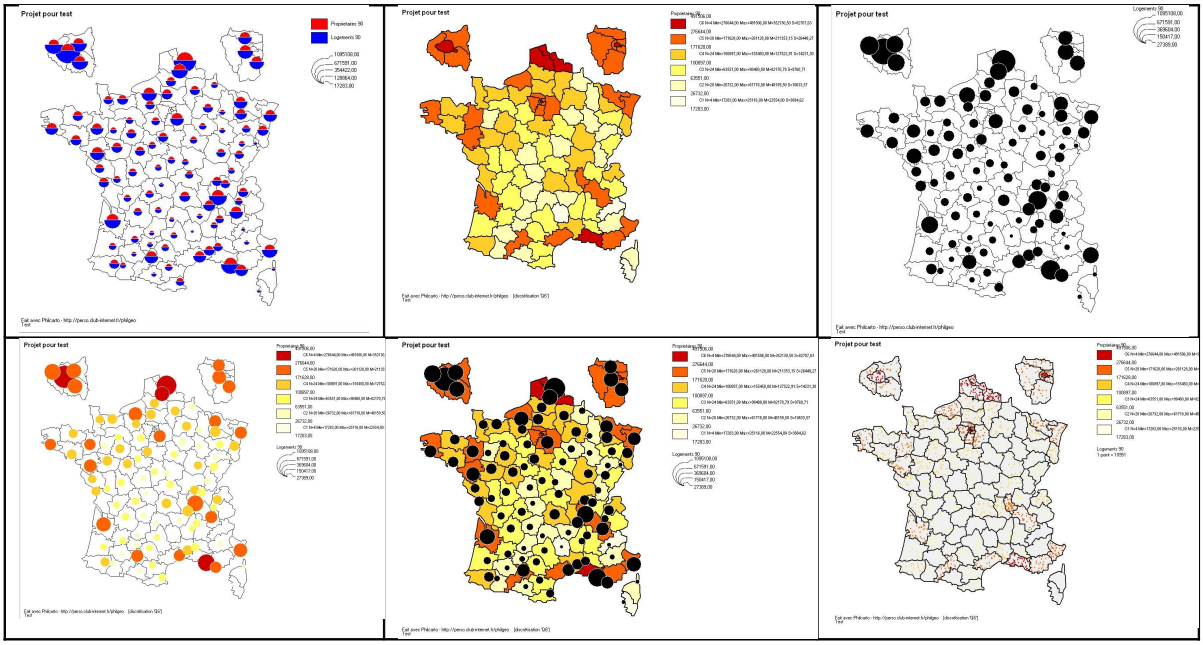
En conclusion, *Cartographeur* offre une solution assez complète pour la génération de cartes. Cependant, ce logiciel semble être adapté à des cartographes ou géographes professionnels pour lesquels il parait suffisamment paramétrable afin de répondre à leurs besoins. En revanche, il est très complexe à utiliser par des non-professionnels. De plus, *Cartographeur* s'inscrit dans une logique de production de cartes en vue de leur utilisation dans des documents et non pas de navigation dans des cartes comme c'est le cas pour *HyperAtlas*. Les deux logiciels sont donc différents de par leurs objectifs et de par le public visé.

2.2.4 PhilCarto

Issu de la recherche universitaire, l'outil *PhilCarto* a été développé par le géographe Philippe Waniez actuellement Directeur de Recherche à l'Institut de Recherche pour le Développement (Sciences sociales, géographie) en poste à l'Université Catholique de Rio de Janeiro, Brésil.
PhilCarto est disponible gratuitement à l'adresse [@PGEO] : http://philgeo.club.fr/

Possibilités offertes pour l'analyse spatiale

Il permet la génération de nombreux types de cartes tel que : cartes en plages, cartes en semis, cartes en points, cartes associant points et plages de couleurs, cartes en hérisson, cartes en points colorés… (*cf.* Figure 2-19).



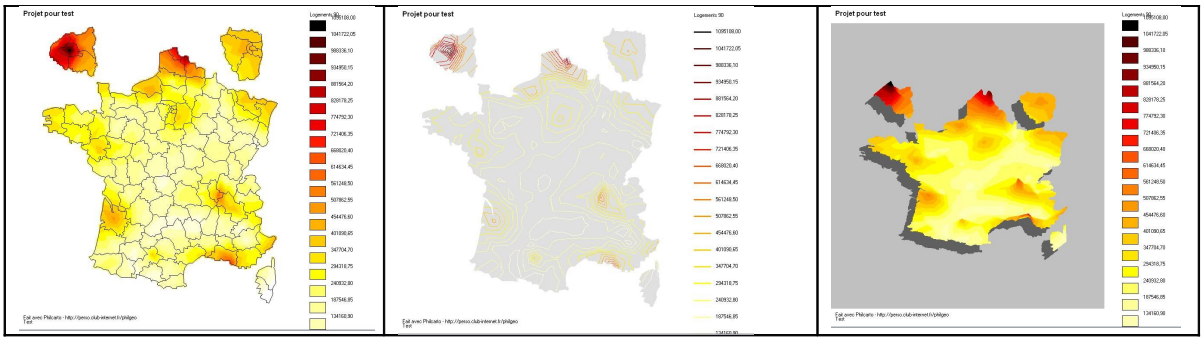


Figure 2-19 : Exemple de cartes générées par le logiciel *PhilCarto*.

De plus, cinq méthodes de discrétisation automatiques sont proposées et la délimitation des classes peut être affinée en agissant sur l'histogramme qui permet de déplacer les limites des classes.

PhilCarto permet également de baser les cartes sur des calculs d'indices selon neuf méthodes mathématiques possibles.

Possibilités de navigation dans la carte

Même s'il permet de zoomer sur certaines cartes, dans son esprit *PhilCarto* s'approche plus d'une logique de production de cartes imprimables que de cartes navigables. Aussi l'accent est-il plus mis sur la variété des cartes qu'il est possible de produire, que sur l'ergonomie de l'outil.

Conclusion

Le nombre et la qualité des cartes produites sont semblables à ceux des outils payants et *PhilCarto* est souvent cité comme outil pédagogique sur le site de nombreuses universités, car les jeux de données qu'il utilise sont gratuits et que leur format est ouvert. Toute personne peut intégrer ses propres données pour générer des cartes qui sont d'autant plus facilement intégrables à des documents qu'elles sont exportables sous forme d'images.
Cependant cet outil peut devenir rapidement complexe à utiliser et nécessite d'avoir des connaissances solides autour de la cartographie et des statistiques.

2.2.5 Récapitulatif des fonctionnalités des outils présentés

Fonctionnalités	**JCT**	***Cartographeur***	***GeoClip***	***PhilCarto***
Affichage d'une carte de contexte (Sans Coloration)	**X**	**X**	**X**	**X**
Affichage d'une carte présentant le numérateur	**X**	**X**	**X**	**X**
Affichage d'une carte présentant le dénominateur			**X**	**X**
Affichage de la carte des ratios	**X**	**X**	**X**	**X**
Affichage de la carte des déviations (locale, moyenne et Globale)				
Zoom sur la carte	**X**	**X**	**X**	**X**
Zoom sur une sélection souris	**X**		**X**	
Zoom sur une unité territoriale		**X**	**X**	
Déplacement sur la carte	**X**		**X**	**X**
Personnalisation des dégradés de couleur et du nombre de classes utilisées pour la génération des cartes choroplèthes	**X** (Mais fortement perfectible)	**X**	**X**	**X**
Gestion de l'imbrication hiérarchique (composition des unités territoriales)		**X**	**X**	
Changement dynamique de l'espace d'étude		**X**	**X**	
Ouverture de carte lors de l'utilisation	**X**	**X**		**X**
Génération un rapport		**X**		**X**
Ajout possible de nouveaux types de cartes		**X**		
Lancement possible via le Web			**X**	
Possibilité de choix de méthode mathématique pour le calcul d'indice				**X**
Type de logiciel	Gratuit	Commercial	Gratuit /Commercial (Selon la version choisie)	Gratuit
Disponibilité de sources	Disponibles	Non disponibles	Non disponibles	Non disponibles

2.3 Synthèse

Lors de notre étude comparative, nous avons pu distinguer deux types d'outils cartographiques : ceux permettant de réaliser des cartes « navigables », et ceux dont la finalité et de produire des cartes d'avantages destinées à l'impression. En général, les outils de productions de cartes statiques sont moins ergonomiques mais permettent de générer des cartes véritablement personnalisées. Cependant, leur utilisation s'adresse davantage à des professionnels de la cartographie, ou à des experts en sciences sociales habitués à travailler avec des SIG. A l'inverse les outils permettant de naviguer dans les cartes sont souvent beaucoup plus accessibles bien qu'ils soient souvent moins riches en termes de cartes proposées.

Ainsi, *GeoClip* est tout à fait adapté à une utilisation par des personnes néophytes grâce à sa bonne ergonomie et sa facilité de prise en main. *PhilCarto* et le *Cartographeur* se destinent plus à un usage par des professionnels de la cartographie et/ou des statistiques.
A ce titre, l'outil *Cartographeur* semble beaucoup plus évolutif de par la possibilité d'y ajouter des modules cartographiques. Cependant, rappelons qu'il s'agit d'une application commerciale utilisant elle-même des fonds de cartes réalisés dans un format propriétaire et dont l'utilisation est, elle-même, soumise à l'achat d'une licence.
PhilCarto, en revanche, s'il est moins aboutit d'un point de vue ergonomique, permet néanmoins de réaliser une grande variété de cartes. Il permet, de plus, de créer et charger ses propres jeux de données (fonds de carte et statistiques). C'est pourquoi, *PhilCarto* est très orienté vers une utilisation dans le domaine de la recherche et l'enseignement car il permet de générer gratuitement des cartes d'une qualité professionnelle.

Comme nous allons le voir dans le chapitre suivant, *HyperAtlas*, a pour ambition de se situer à la frontière entre ces outils en proposant une certaine variété de cartes et un bon niveau de personnalisation mais en restant le plus convivial possible afin d'être utilisable tant par des professionnel que par des particuliers.

CHAPITRE 3

RÉINGÉNIERIE DE L'APPLICATION HYPERATLAS

Nous présentons dans ce chapitre les travaux de réingénierie réalisés dans le cadre de ce mémoire, sur l'application *HyperAtlas*.
Ces travaux reprennent le cheminement du processus de conception représenté par la courbe du soleil. [Nanci et al., 92]

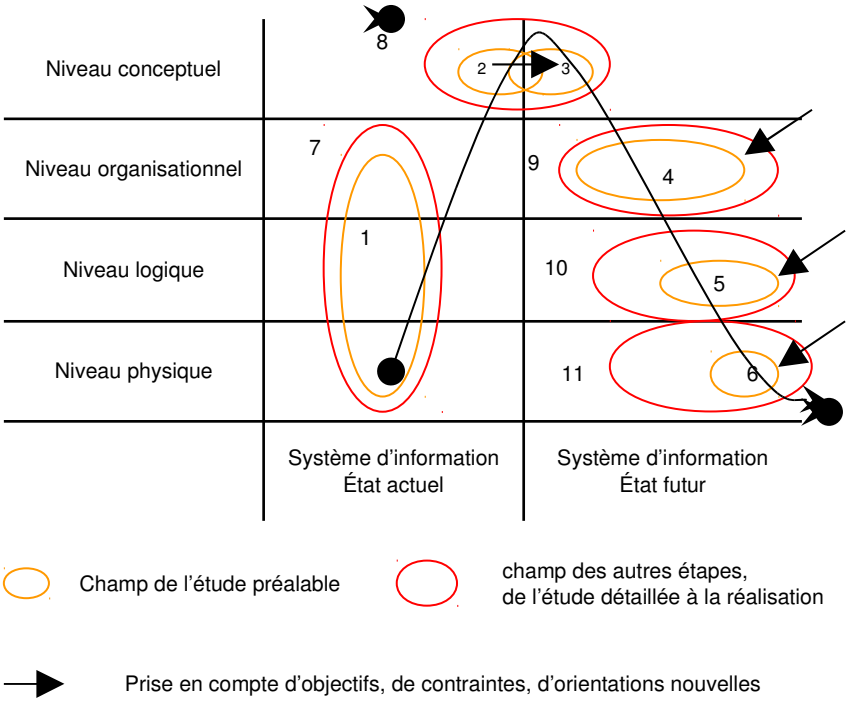


Figure 3-20 : Le cheminement de conception de Merise (courbe dite du soleil).

Dans un premier temps, nous faisons un rappel sur les principes mis en œuvre dans l'application. Ensuite, une analyse de la version 1.0 sera proposée avant d'aboutir aux modifications effectuées.
Ensuite nous présentons une analyse de la version précédente de l'application avant de présenter les travaux que nous avons effectués sur l'application.
Pour illustrer nos propos, l'exemple de l'espace Européen sera utilisé. Cet exemple repose sur des réalisations demandées par l'organisme Européen ESPON[9] [@ESPON].

[9] ESPON est l'abréviation de « European Spatial Planning Observation Network ».

3.1 Cahier des charges

Cette partie présente des éléments du cahier des charges du Module MTA[10] du logiciel *HyperAtlas*. Outre un rappel sur les objectifs du projet, les principaux concepts et terminologies utilisés par la suite sont détaillés pour permettre au lecteur d'avoir une meilleure compréhension de ceux-ci. Ensuite, les différentes cartes générées par l'application sont présentées.

3.1.1 Principes et concepts

Dans le cadre du Projet ESPON 3.1[11] [@ESPON], les membres du projet de recherche *HyperCarte* [@HYP] mettent au point des outils interactifs d'analyse territoriale. Le but étant d'exploiter des données provenant d'une base de données regroupant diverses informations statistiques sur les pays de la Communauté Européenne ainsi que sur les pays candidats à l'adhésion. L'objectif des membres du projet *HyperCarte* est de coupler ces données à une interface permettant de les exploiter pour proposer à un utilisateur de produire un ensemble de cartes rendant compte de la cohésion territoriale. Ces cartes présentent notamment la déviation d'un territoire vis-à-vis de la moyenne européenne, nationale ou par rapport aux régions voisines. Pour répondre à ces attentes, il est nécessaire de concevoir un outil capable de représenter sous forme de cartes les différentes unités territoriales. Cet outil doit être multiscalaire, c'est-à-dire, permettre à un utilisateur de pouvoir travailler à différentes échelles géographiques, des pays aux départements. L'utilisateur doit également pouvoir choisir la zone d'étude sur laquelle il souhaite travailler (Europe des 25, Europe des 29, Pays d'Europe Centrale et Orientale...).
Outre les fonctionnalités de base (tel que le déplacement ou le zoom sur les cartes), *HyperAtlas* permet de choisir : la zone d'étude (Europe des 15, Europe des 29...), le maillage administratif à étudier (NUTS1, NUTS2,…), les indicateurs statistiques (Population, PIB...) ainsi que les types de déviation à étudier. Il est intéressant de noter que la possibilité de travailler à plusieurs échelles administratives est à l'origine du qualificatif de " multiscalaire " dans le sigle MTA. Il permet également de sauvegarder un rapport au format XHTML reprenant l'ensemble des cartes produites ainsi qu'un tableau numérique représentant les résultats des calculs des cartes.
Il faut cependant signaler que les données sont incluses au programme *HyperAtlas* et ne permettent donc pas à un utilisateur de charger d'autres jeux de données, ou d'enrichir les données existantes. Cette lacune fait donc l'objet d'une demande de mise à jour, prévue pour fin 2006, de la part des membres du projet ESPON 3.1.

De ce qui précède, nous pouvons extraire les entités suivantes : les unités territoriales, les espaces d'études, les maillages territoriaux et les indicateurs statistiques. Nous allons maintenant présenter ces différentes entités et leurs relations. Pour cela, nous utilisons, outre une description textuelle, des diagrammes UML[12], qui offrent une schématisation claire et normalisée.

Les unités territoriales (TU ou Unit)

Ce sont toutes les unités géographiques, quelque soit leur niveau hiérarchique. Elles ne portent pas nécessairement de données attributaires mais elles ont une

[10] MTA est l'abréviation de « Multiscalar Territorial Analyse », analyse territoriale multiscalaire.
[11] Le volet 3.1 du projet ESPON concerne les outils intégrés d'aménagement du territoire européen.
[12] UML (*Unified Modeling Language*) : formalisme de modélisation objet.

description/existence graphique dans l'application. Cette classe d'objets regroupe donc aussi bien des pays que des communes, des espaces européens (UE15, UE25) ou mondiaux, suivant l'échelle d'étude de l'utilisateur. De plus, des unités territoriales peuvent être une agrégation d'unités territoriales de niveau de maillage inférieur, c'est le cas par exemple de la région Rhône-Alpes qui est une agrégation des départements qui la compose (Rhône, Isère, …). De la même manière, certaines unités n'ont pas d'unités parentes ou ne sont pas composées d'autres unités, c'est le cas des unités territoriales qui sont ajoutées à un projet pour des raisons de lisibilité mais qui n'entrent jamais dans la composition d'une aire d'étude (Ex. la Russie). La Figure 3-21 présente cette relation de composition, appelée hiérarchie dans la suite de ce document.

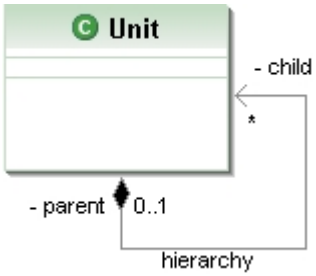


Figure 3-21 : Diagramme de classes : relation hiérarchique entre unités territoriales.

Les espaces d'études (Area)

Une aire d'étude représente un sous-ensemble nommé d'unités territoriales, associées à une hiérarchie de maillage dans un projet donné. Cet ensemble d'unités peut être utilisé soit comme un espace d'étude dans l'application MTA (choix du contexte d'étude), soit comme espace de référence pour le calcul des déviations globales (déviation d'une unité par rapport à un espace de référence : UE29, UE15…). Suivant l'échelle de l'étude, l'aire peut correspondre à des espaces européens, à des espaces plus spécifiques (Arc Atlantique, Nord Atlantique) mais aussi à des unités territoriales non élémentaires (pays, régions).
La description des aires d'étude par les unités territoriales doit se faire avec les unités de plus haut niveau possible, l'application fonctionnant grâce à un arbre hiérarchique. Par exemple, l'Europe des 25 doit être définie comme un ensemble de pays et non pas comme un ensemble de régions.

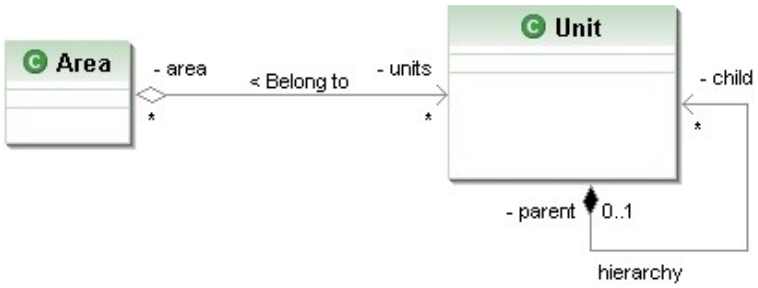


Figure 3-22 : Diagramme de classes : relation entre aire d'étude et unités territoriales

Maillages territoriaux (Zoning)

Un *zoning* est un découpage, une partition de l'espace. Elle est constituée par un ensemble d'unités territoriales. Elle découpe entièrement ou partiellement les espaces d'étude. La relation hiérarchique des zoning entre-elles n'est pas visible dans cette classe d'objets, mais se retrouve grâce aux relations Unit-Zoning et Unit-Unit.

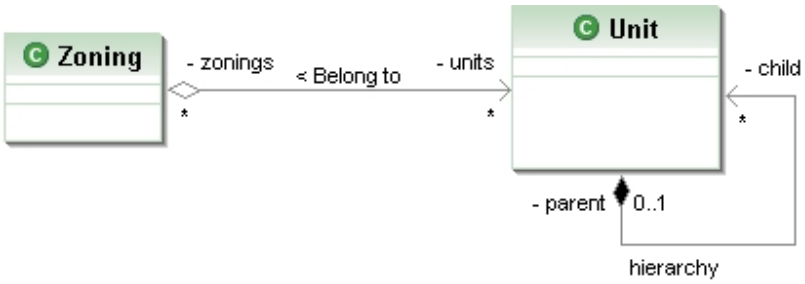


Figure 3-23 : Diagramme de classes : relation entre maillage et unités territoriales

Dans le cadre du projet ESPON, les maillages choisis pour découper les pays européens sont les maillages N.U.T.S. [@NUTS], Nomenclature des Unités Territoriales Statistiques. Ces maillages ont pour but de fournir des découpages territoriaux uniques et cohérents pour l'établissement des statistiques régionales de l'Union Européenne. La Figure 3-24 présente un exemple de la composition de la nomenclature N.U.T.S. et de la manière dont les unités de rang inférieur composent celles de rang supérieur. Dans cet exemple, nous pouvons voir que l'unité « Centre-Est » appartenant au maillage NUTS 1 est composée des unités « Rhône-Alpes » et « Auvergne » appartenant au maillage NUTS 2, et ainsi de suite...

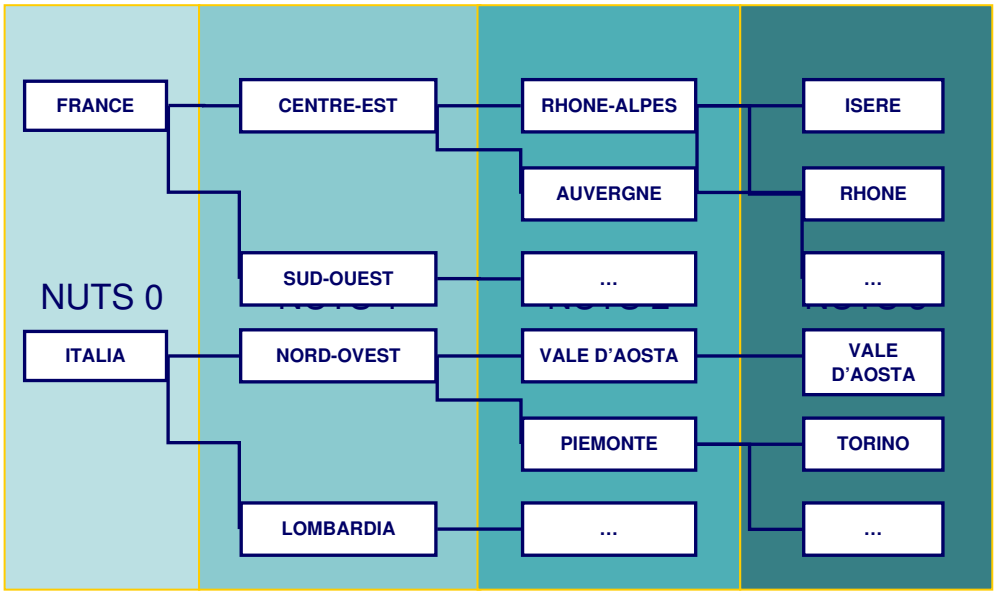


Figure 3-24 : Extrait du maillage N.U.T.S.

Il est important de noter que la relation hiérarchique des *zoning* entre eux n'est pas visible dans cette classe d'objets ; mais se retrouve grâce aux relations *Unit-Zoning* et *Unit-Unit*.

Les données statistiques (Stocks)

Les données statistiques sont des données numériques caractérisant des unités territoriales et permettant de les comparer entre elles. Elles portent sur la géographie, la démographie, l'économie... La Figure 3-21 montre la relation entre unités territoriales et stocks.

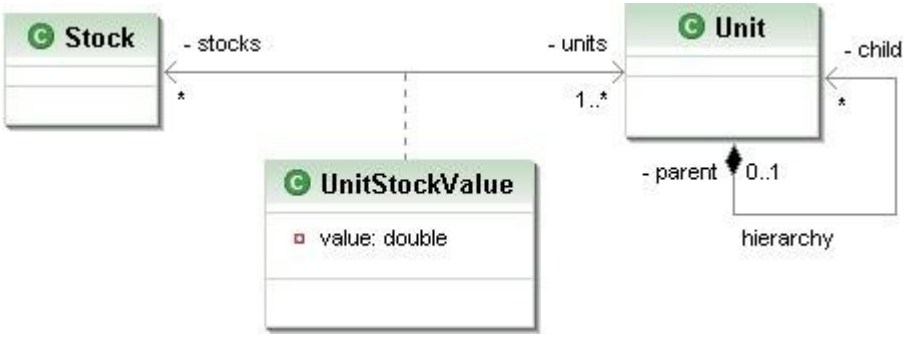


Figure 3-25 : Diagramme de classes : relation entre stocks et unités territoriales

Les relations de composition entre unités territoriales permettent de déduire des stocks ; ainsi la " Population en 1999 " pour la région Rhône-Alpes est égale à la somme de la population en 1999 des départements qui la composent.

L'utilisation des entités présentées

Les entités définies précédemment sont à la base de l'application *HyperAtlas* et de son paramétrage. Nous introduisons certains concepts et terminologies liés à ces entités. Trois concepts essentiels apparaîtront dans la suite de ce document : le contexte de l'étude, les indicateurs, et le contexte des déviations. Le contexte d'étude regroupe deux entités : l'espace d'étude et le maillage territorial. Il est paramétrable par un utilisateur lui permettant de choisir son aire d'étude (Europe des 29, Europe des 15...) et le maillage (régions, départements...) sur lequel porte l'étude. La partie " indicateurs " composée de deux statistiques permet d'en faire le ratio pour obtenir des rapports ou des taux tels que la richesse par habitant ou le taux de natalité. Enfin le contexte des déviations permet de définir une moyenne à laquelle sera comparé un taux. Ce contexte se compose de deux entités : une aire d'étude et un maillage. Par exemple, il permet de comparer le taux de mortalité d'un département à la moyenne régionale (déviation moyenne) ou à la moyenne d'une aire d'étude (déviation globale).
NB : si l'aire d'étude utilisée dans le contexte de déviation peut être identique à celle du contexte d'étude, cela n'est pas nécessairement le cas : il est donc tout à fait possible de travailler sur l'Europe des 27 et de comparer les ratios obtenus avec ceux de l'Europe des 15.

3.1.2 Cartes générées

Pour représenter les diverses informations traitées par *HyperAtlas*, deux types de cartes ont été choisis et implémentés : les cartes à disques proportionnels et les cartes choroplèthes.

Les cartes à disques proportionnels

Le site Web de *GeoClip* [*@GEOCLIP*] offre une définition et une description claires et explicites de ce type de carte et de son intérêt majeur. Voici cette définition : « *La symbolisation par cercles ou disques proportionnels est la représentation la plus intuitive, la plus facile à comprendre. Elle est adaptée aux variables brutes absolues, comme une population, une production, dès lors par exemple que les valeurs s'additionnent. On peut sommer une population, on ne peut pas sommer un taux de chômage. L'œil humain appréhende les surfaces plus volontiers que les rayons des cercles, c'est donc la surface qu'on rendra proportionnelle à la variable étudiée (ou le rayon à la racine carrée de la variable).*

On a le choix entre dessiner des cercles simples ou des disques pleins. Les disques pleins sont généralement plus faciles à lire, car ils donnent à voir leur surface plus facilement. » La Figure 3-26 et la Figure 3-27 montrent un exemple de légende et de carte à disques proportionnels.

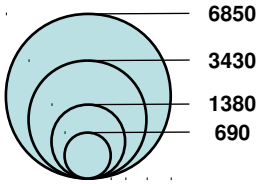


Figure 3-26 : Exemple de légende d'une carte à disques proportionnels.

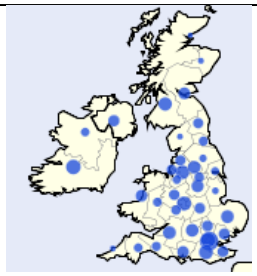


Figure 3-27 : Exemple de carte à disques proportionnels

Les cartes choroplèthes

Les cartes choroplèthes sont des cartes souvent utilisées en statistique, elles permettent de représenter des valeurs relatives telles que des taux. « *Les cartes choroplèthes comportent des données statistiques étant donné qu'elles représentent des régions situées à l'intérieur des limites d'unités géographiques prédéfinies (comme les divisions de recensement et les secteurs de recensement)* » [@STCAN]. Ces cartes représentent donc des unités territoriales remplies avec une couleur en accord avec une échelle discrétisée comme le montre la Figure 3-28.

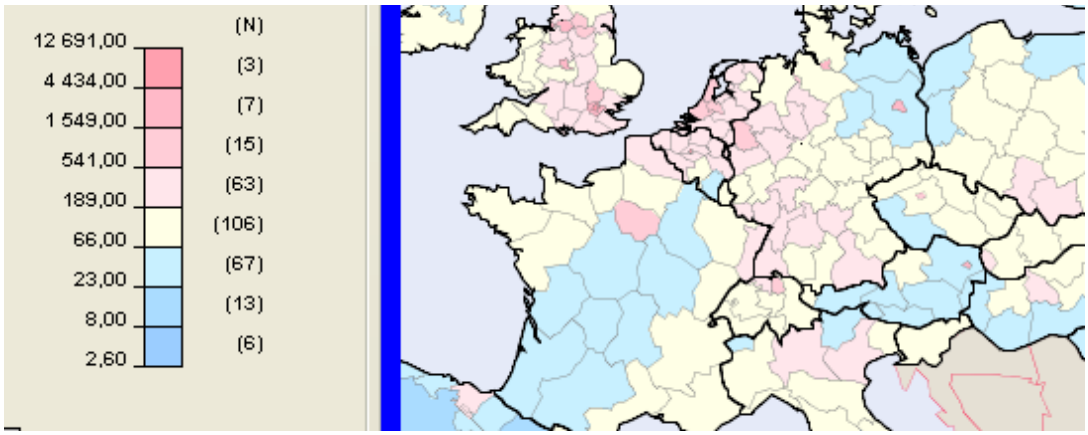


Figure 3-28 : Exemple de carte choroplèthe, les régions présentées sont coloriées à l'aide des couleurs décrites dans la légende et en fonction de leur valeur.

Les cartes produites par *HyperAtlas*

Après avoir décrit, dans la partie précédente, les deux principaux types de cartes choisis pour représenter les données dans *HyperAtlas*, nous nous intéressons à leur implémentation. Pour cela nous allons dans un premier temps répertorier les fonctionnalités communes à toutes les cartes, pour ensuite étudier plus en détail les cartes produites.

Parmi les fonctions proposées à l'utilisateur, pour tous les types de cartes, nous pouvons citer le déplacement, le zoom, l'affichage des détails d'une Unité (*cf.* Figure 3-29).

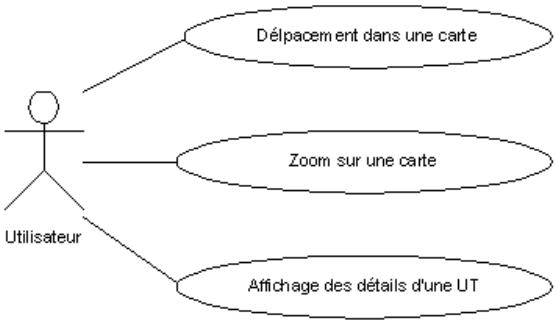


Figure 3-29 : Diagramme des cas d'utilisation de l'affichage de carte dans *HyperAtlas*.

Après avoir présenté les fonctions communes à chacune des cartes de l'application, nous les détaillons. La première carte produite par l'application est la carte de contexte.

Carte de contexte

Cette carte permet de visualiser le contexte d'étude sur lequel porte l'analyse ainsi que les unités territoriales qui le composent. Pour cela, l'application propose une carte représentant les unités territoriales étudiées d'une couleur, et celles qui ne font pas partie de l'aire d'étude, d'une autre, comme le montre la Figure 3-30.

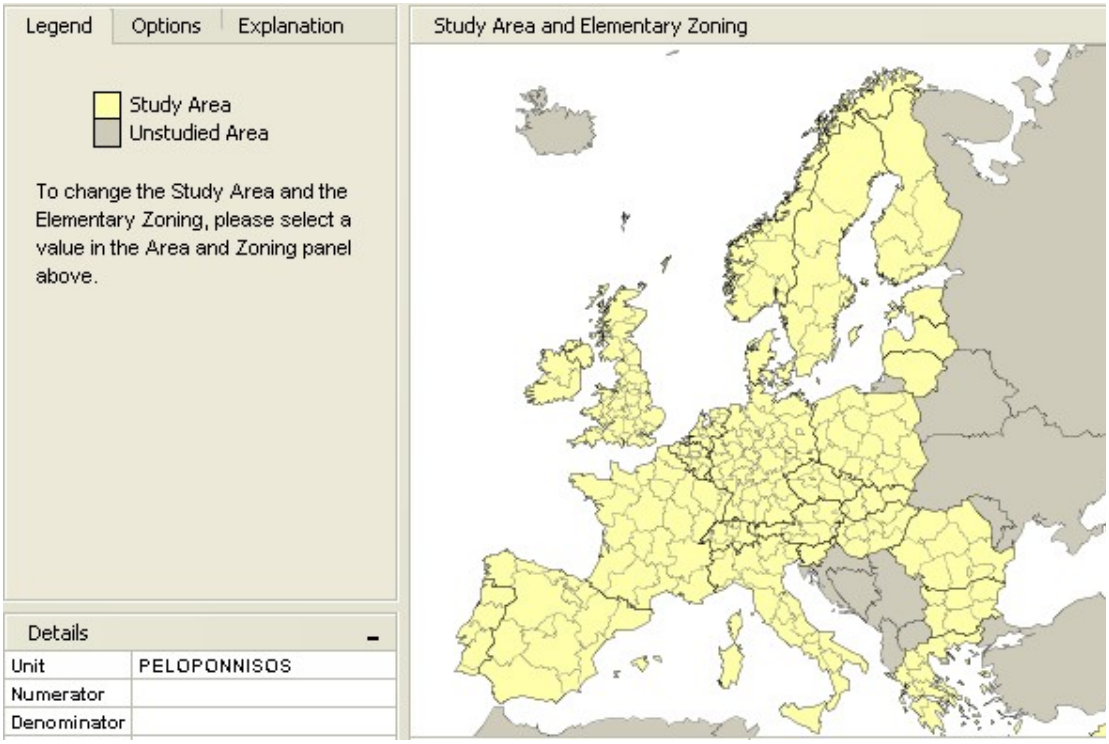


Figure 3-30 : Exemple de carte de contexte, les unités concernées par l'étude sont coloriées en jaune alors que celle hors de l'étude sont en gris.

Après avoir défini la zone qu'il souhaite étudier, l'utilisateur peut alors visualiser des cartes représentant les indicateurs choisis à l'aide des cartes de stocks présentées dans la partie suivante.

Cartes de stocks

L'application permet de générer deux cartes à disques proportionnels comme celle présentée par la Figure 3-31. Outre les fonctionnalités basiques offertes par toutes les cartes, l'utilisateur peut paramétrer la couleur et la taille des disques. Comme nous l'avons dit précédemment, la surface des disques est proportionnelle à la valeur du stock qu'elle représente. La construction d'une telle carte compte trois étapes. Tout d'abord, on trie les stocks par ordre décroissant de valeur. Ensuite on associe le disque de taille maximale (taille arbitrairement fixée) à la plus forte valeur. Puis, on détermine la taille des disques par proportionnalité entre la valeur du stock et celle du stock maximal. Enfin, il ne reste qu'à dessiner les unités territoriales puis les cercles par ordre décroissant de taille pour que les plus petits ne soient pas recouverts par ceux de diamètre supérieur. Pour des raisons de lisibilité, on trace également les contours des disques avec une autre couleur que la couleur de remplissage afin de les distinguer s'ils se superposent. La Figure 3-31 présente un exemple de carte à disques proportionnels : nous pouvons remarquer qu'un liséré blanc permet de distinguer les disques qui se superposent.

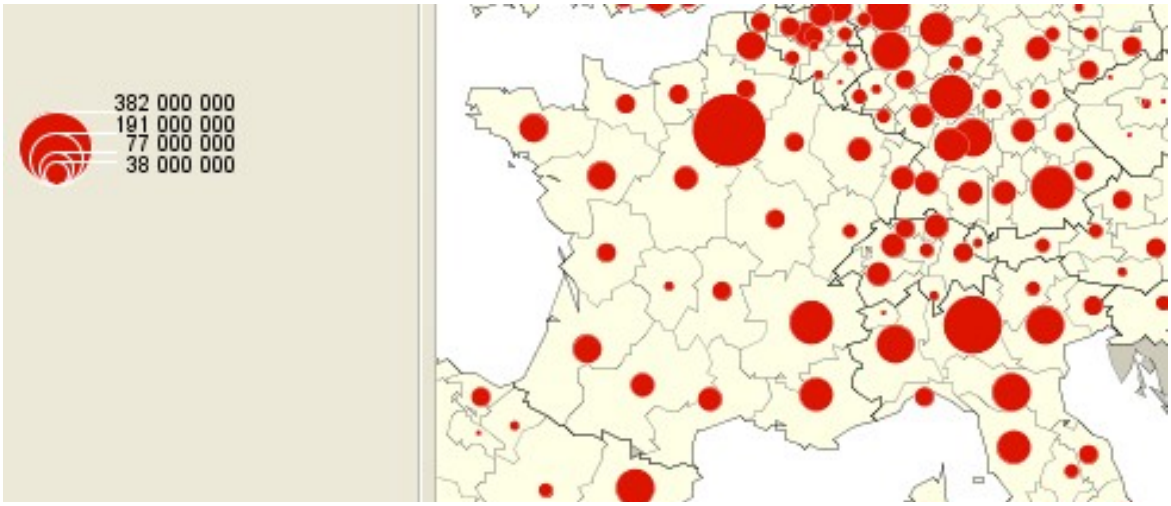


Figure 3-31 : Carte de stocks.

Carte de ratio

L'utilisateur peut également visualiser une carte choroplèthe présentant le rapport entre les stocks choisis (numérateur et dénominateur), cette carte utilise une palette de couleurs à ton unique comme le montre la Figure 3-32. Le rapport entre numérateur et dénominateur est appelé indicateur. Sa valeur est représentée par un aplat dont la couleur est en correspondance avec celle attribuée à la plage de valeur correspondante.

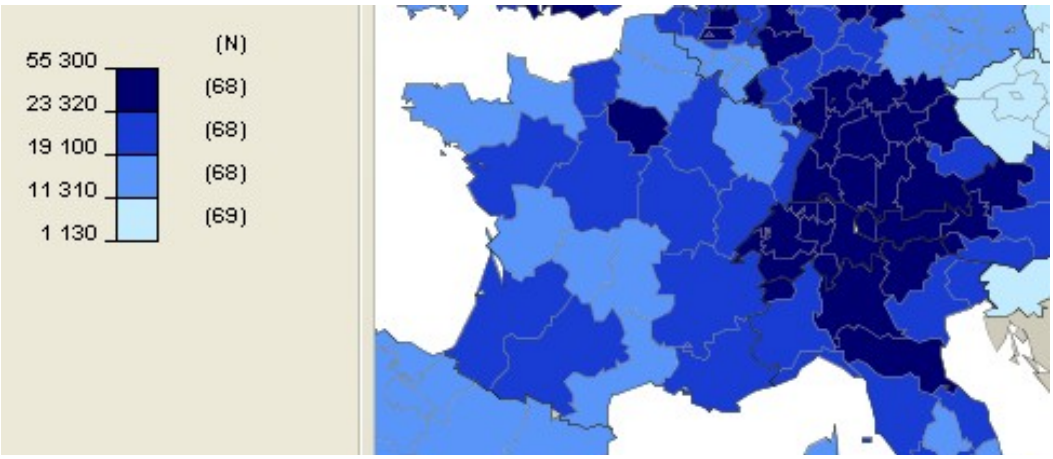


Figure 3-32 : Carte de rapport, choroplèthe avec une palette à ton unique.

Un utilisateur peut également personnaliser cette carte en choisissant la couleur à utiliser pour la palette ainsi que le nombre de classes utilisées pour la discrétisation.

Cartes de déviations

Ces cartes permettent de rendre compte des disparités des indicateurs vis-à-vis d'une référence. Cette référence peut être l'intégralité d'une aire d'étude ou une région contenant l'unité, ou bien encore les unités voisines. *HyperAtlas* s'intéresse donc à trois déviations possibles : la déviation globale (par rapport à un espace d'étude), la déviation moyenne (par rapport à une région englobante), et enfin la déviation locale qui utilise les unités contiguës.

Ces cartes permettent donc de situer un indicateur par rapport à une référence. Par exemple dans le cas de la richesse par habitant en Europe, elle permet de savoir si une unité territoriale est plus ou moins riche que la moyenne européenne, plus ou moins riche que son unité englobante (région, pays…), et enfin si elle est plus ou moins riche que son voisinage. Pour représenter les déviations, *HyperAtlas* utilise une carte choroplèthe utilisant une palette de couleurs formée à partir d'un dégradé entre deux couleurs comme le montre la Figure 3-33.

Comme pour la carte de ratio, l'utilisateur peut personnaliser les dégradés de couleur ainsi que le nombre de classes usitées pour la discrétisation. Mais il peut également choisir l'échelle utilisée pour la discrétisation : celle-ci peut être arithmétique ou géométrique.

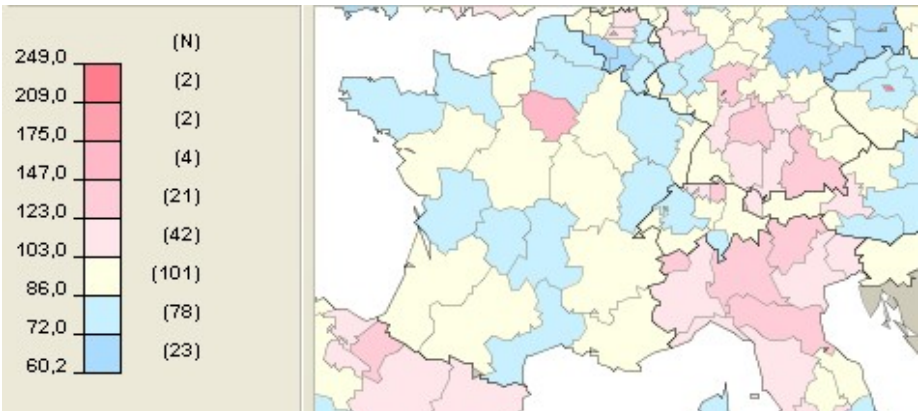


Figure 3-33 : Carte de déviation, choroplèthe avec une palette de couleurs à deux tons.

Carte de synthèse

Enfin *HyperAtlas* propose une carte de synthèse des trois déviations qui permet de classer les unités en fonction de leurs déviations. Les unités ont alors pour couleur de fond l'une des huit couleurs possibles, décrites dans la légende de la carte. La Figure 3-34 montre un exemple de carte de synthèse avec, sur la gauche, le tableau fourni en légende qui permet de comprendre la carte. Ce tableau propose une couleur en fonction du dépassement ou non d'un seuil.

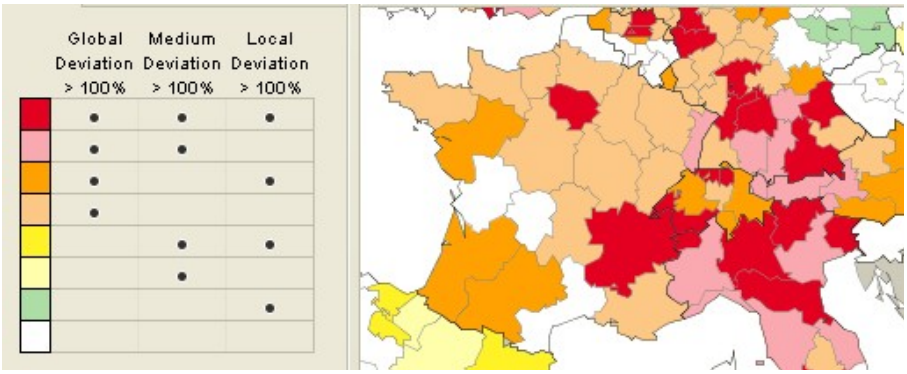


Figure 3-34 : Carte de synthèse.

Dans cet exemple, les unités sont coloriées en rouge si et seulement si chacune de leur trois déviations est supérieure à 100%, en blanc si, au contraire, aucune de leur déviation ne dépasse 100%... Pour avoir une vision encore plus précise des valeurs prises par les déviations, *HyperAtlas* permet d'afficher un histogramme récapitulatif pour une unité choisie (voir Figure 3-35). L'utilisateur peut, s'il le souhaite, ouvrir simultanément plusieurs fenêtres de graphiques pour comparer les déviations des unités territoriales qu'elles représentent.

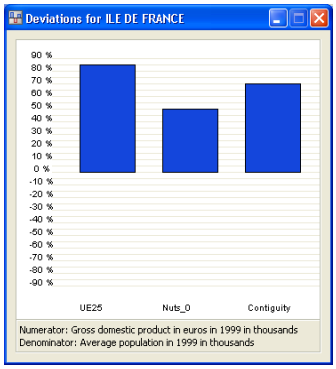


Figure 3-35 : Histogramme récapitulant les déviations d'une unité.

Cette carte peut également être personnalisée car il est possible de choisir le seuil à utiliser pour les déviations (100% par défaut) ainsi que l'opérateur de comparaison (supérieur par défaut). Il est donc possible d'afficher en rouge les unités qui, par exemple, ont toutes leurs déviations inférieures à 80% et en blanc celles pour lesquelles l'ensemble des déviations est supérieur à 80%. Il faut cependant noter que les couleurs affectées dans la légende et la carte, ne sont pas personnalisables.

3.1.3 Sauvegarde de l'étude et génération d'un rapport

Après avoir défini le contexte de son étude ainsi que les stocks à utiliser, un utilisateur peut sauvegarder les paramètres de son espace de travail. Il est alors possible de restaurer cet espace et de retrouver exactement la session sauvegardée, tant en termes de contexte d'étude et de paramètres, que de positionnement et zoom dans les cartes.

De même, *HyperAtlas* permet de générer facilement un rapport sur les résultats de l'étude. Celui-ci comporte un descriptif du contexte de l'étude, les cartes générées ainsi qu'un tableau récapitulatif des stocks et résultats obtenus (ratio et déviations). Pour des raisons de portabilité, le format choisi est le format XHTML[13]. Ce format permet de créer des docu-

[13] XHTML : Extensible HyperText Markup Language. Langage normalisé par le W3C servant à la publication de pages Web sur Internet.

ments accessibles via le Web. Il devient alors possible d'intégrer un ou plusieurs rapports à un site Web pour que ceux-ci soient directement consultables en ligne. Les copies d'écran présentées dans la Figure 3-36 et la Figure 3-37.

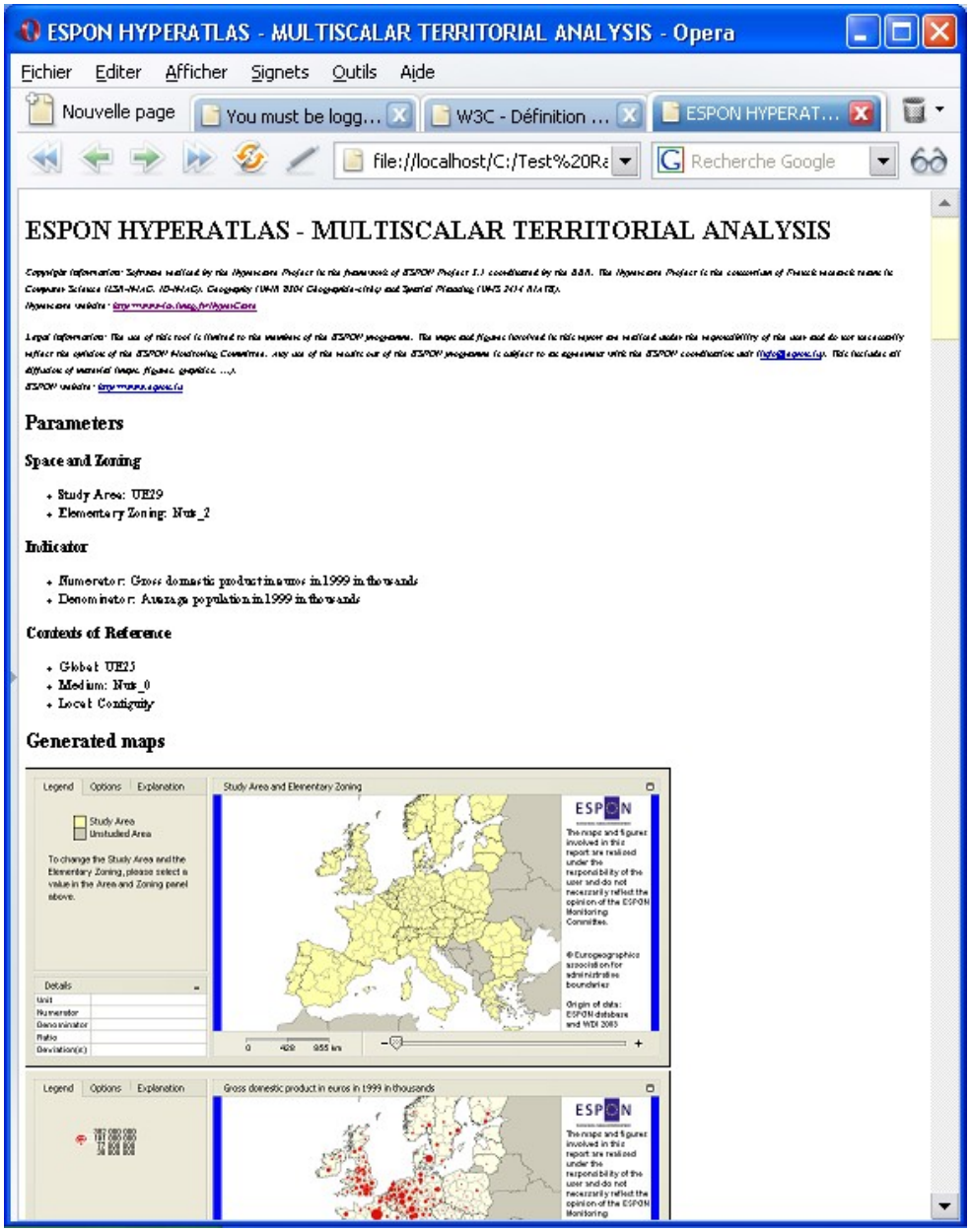


Figure 3-36 : Copie d'écran de l'entête d'un rapport affiché dans un navigateur Web.

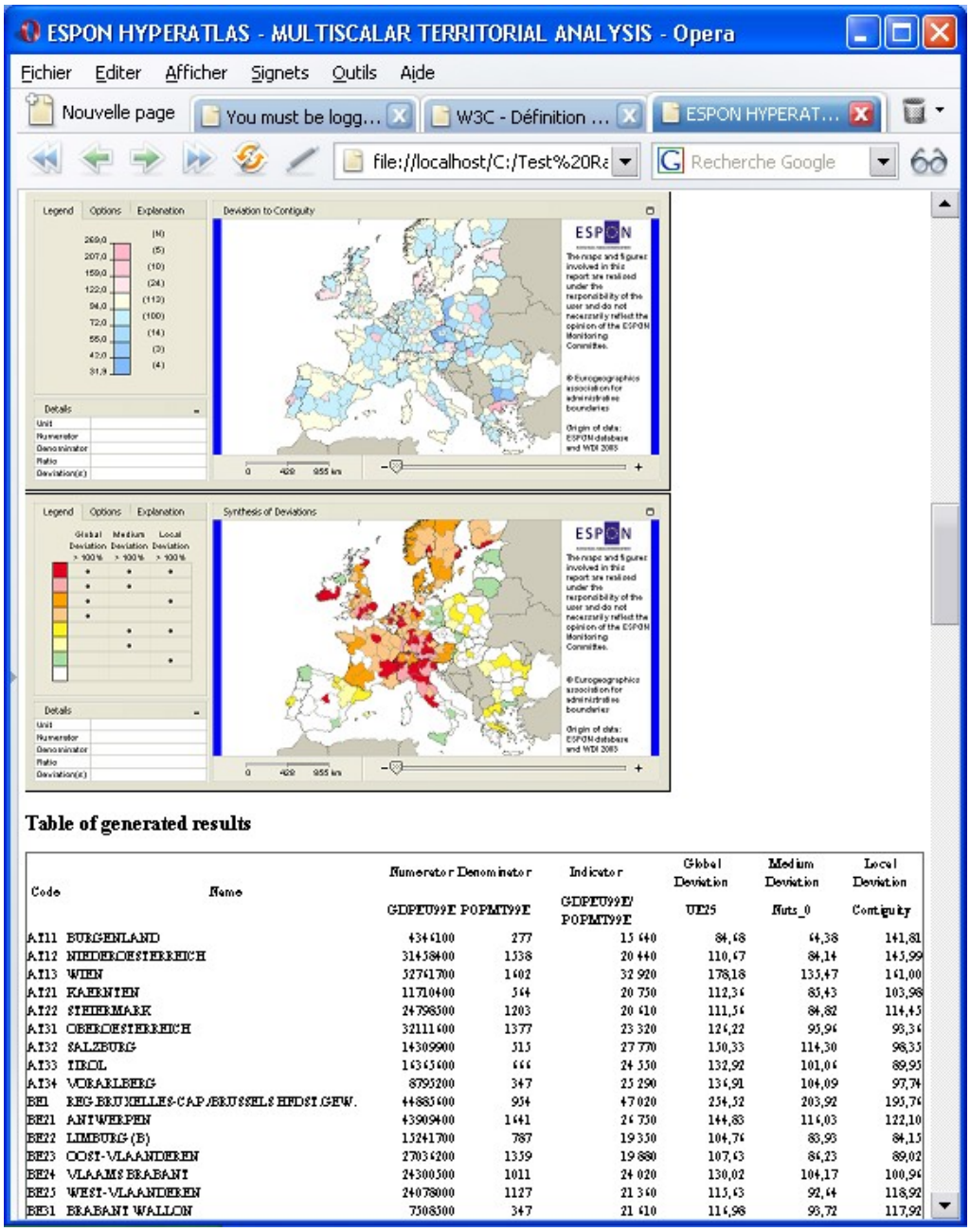


Table of generated results

Code	Name	Numerator GDPEU99E	Denominator POPMT99E	Indicator GDPEU99E/ POPMT99E	Global Deviation UE25	Medium Deviation Nuts_0	Local Deviation Contiguity
AT11	BURGENLAND	4341100	277	15 660	84,68	66,38	141,81
AT12	NIEDEROESTERREICH	31458400	1538	20 440	110,67	84,14	145,99
AT13	WIEN	52761700	1602	32 920	178,18	135,47	141,00
AT21	KAERNTEN	11710400	564	20 750	112,36	85,43	103,98
AT22	STEIERMARK	24798500	1203	20 610	111,56	84,82	114,45
AT31	OBEROESTERREICH	32111400	1377	23 320	126,22	95,96	98,36
AT32	SALZBURG	14309900	515	27 770	150,33	114,30	98,35
AT33	TIROL	14365400	666	24 550	132,92	101,06	89,95
AT34	VORARLBERG	8795200	347	25 290	136,91	104,09	97,74
BE1	REG.BRUXELLES-CAP./BRUSSELS HFDST.GEW.	44885400	954	47 020	254,52	203,92	195,76
BE21	ANTWERPEN	43909400	1661	26 750	144,83	116,03	122,10
BE22	LIMBURG (B)	15241700	787	19 350	104,76	83,93	84,15
BE23	OOST-VLAANDEREN	27034200	1359	19 880	107,63	84,23	89,02
BE24	VLAAMS BRABANT	24300500	1011	24 020	130,02	104,17	100,96
BE25	WEST-VLAANDEREN	24078000	1127	21 360	115,63	92,64	118,92
BE31	BRABANT WALLON	7508500	347	21 610	116,98	93,72	117,92

Figure 3-37 : La carte de synthèse ainsi que le tableau de résultat de l'étude en fin du rapport.

3.2 Analyse de la version 1.0.0.c

Nous allons maintenant nous intéresser à la version 1.0.0.c d'*HyperAtlas*. Celle-ci a été proposée et fournie à ESPON dans le cadre de son projet 3.1. Elle est, par ailleurs, à la base de ce travail.
Pour commencer cette analyse, nous décrivons d'abord ces fonctionnalités et l'interface utilisateur. Puis, nous étudions plus en détail les techniques employées dans cette version, pour finalement en dégager des axes d'amélioration.

3.2.1 Fonctionnalités et interface utilisateur

De nombreux efforts ont été fournis pour la réalisation d'une interface la plus ergonomique et esthétique possible. En effet, *HyperAtlas* manipule des concepts parfois complexes : il est donc nécessaire de proposer une interface utilisateur qui n'augmente pas davantage cette complexité. La Figure 3-38 propose une copie d'écran de l'interface utilisateur d'*HyperAtlas* v1.0. Dans cette version, des travaux ont été réalisés pour rendre l'interface plus conforme [Cuenot, 05] aux recommandations pour le développement d'interfaces utilisateurs, décrites par des spécialistes en ergonomie [BAST, 93], des éditeurs de système d'exploitation [APPL, 04] ou par le CNRS [CNRS, 00].

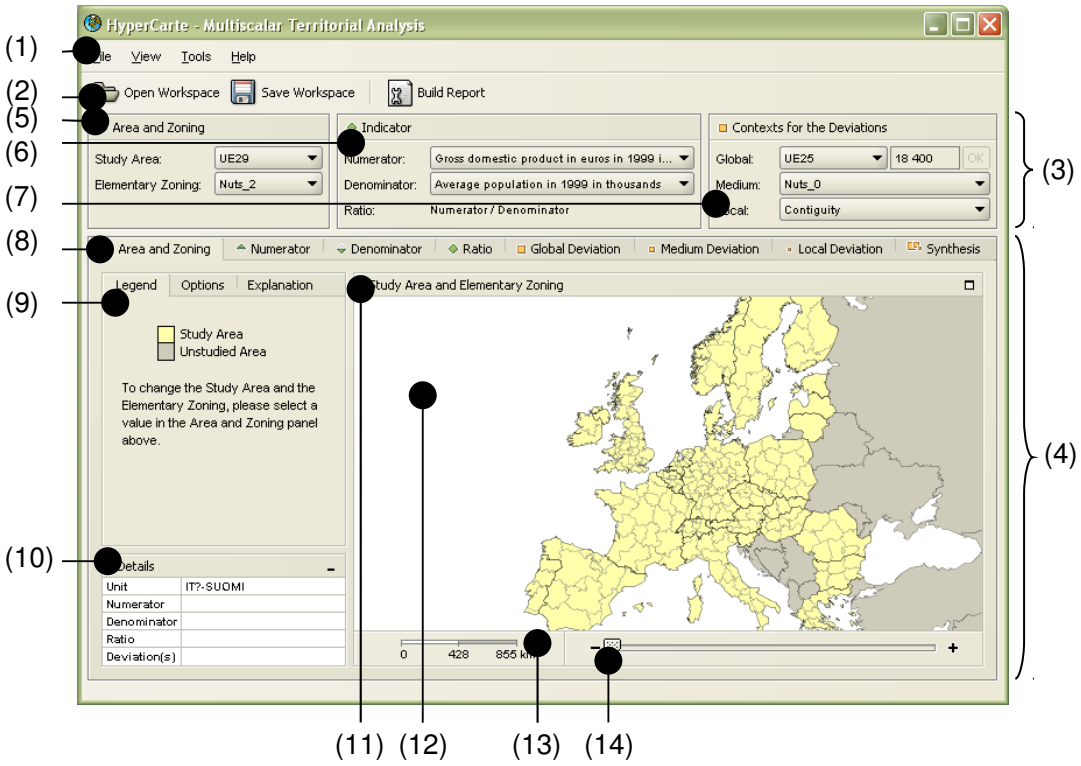


Figure 3-38 : Interface de la version 1.0 d'*HyperAtlas*

(1) Barre de menu *(2) Barre d'outils* *(3) Panneau des paramètres d'étude* *(4) Panneau à onglet contenant les cartes générées* *(5) Panneau de choix de contextes* *(6) Panneau des Indicateurs* *(7) Panneau du contexte des déviations*	*(8) Onglet permettant le choix de la carte à afficher* *(9) Panneau à onglets regroupant la légende, les options et la description de la carte affichée* *(10) Zone d'affichage des détails d'une unité territoriale* *(11) Titre de la carte* *(12) Carte* *(13) Echelle*

	(14) Curseur de zoom.

Cette interface (Figure 3-38) comporte 4 parties principales : un menu *(1)*, une barre d'outils *(2)*, un panneau de paramétrage *(3)* et enfin la partie où les cartes sont affichées *(4)*. Le panneau des paramètres est lui divisé en trois sous-parties correspondant : au contexte de l'étude *(5)*, à l'indicateur *(6)* et au contexte des déviations *(7)*. Chacun de ces panneaux de paramétrage reprend en titre le nom du concept qu'il représente. Le paramétrage du contexte d'étude, réalisé dans le panneau du même nom, se fait grâce à deux listes déroulantes (combo box) offrant la possibilité de choisir, pour l'une, l'aire d'études, et pour l'autre, le maillage. Il en est de même pour les indicateurs, une liste déroulante permet le choix d'un numérateur, et l'autre, celui d'un dénominateur. De même pour le paramétrage du contexte des déviations, avec toutefois, l'ajout d'une zone de texte et d'un bouton servant à la définition manuelle d'une moyenne pour la déviation globale, comme le montre la Figure 3-39.

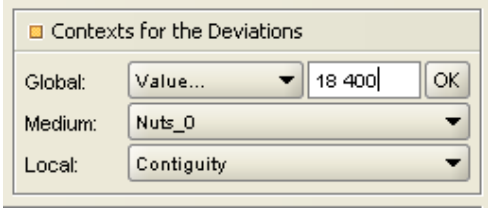


Figure 3-39 : Détail du panneau de paramétrage des déviations.

Après avoir défini les paramètres sur lesquels porte son étude, l'utilisateur peut visualiser directement les cartes résultantes dans la zone de la fenêtre destinée à cet effet. Les fonctionnalités de l'outil pourraient s'arrêter là, mais l'utilisateur peut également personnaliser les cartes produites. Ces personnalisations dépendent, bien entendu, de la carte qu'il souhaite personnaliser : il est évident que les cartes à disques proportionnels offrent des personnalisations que ne peuvent offrir les cartes choroplèthe et vice versa. Nous allons donc étudier plus en détail les personnalisations possibles en fonction des cartes. La plus basique de toutes - la carte de contexte - n'offre pas dans cette version, de possibilités de personnalisation. Ce n'est pas le cas des autres cartes, telles que les cartes du numérateur et du dénominateur. Ces cartes permettent à l'utilisateur de spécifier la taille des disques ainsi que leur couleur de fond (Figure 3-40 et Figure 3-41). Ces personnalisations sont réalisées à l'aide de l'onglet « Options » du panneau de description *(9)* des cartes présentées par la .

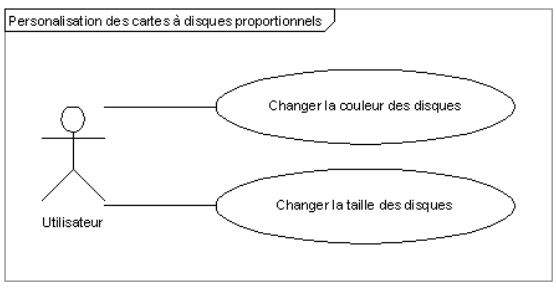


Figure 3-40 : Diagramme de cas d'utilisation pour la personnalisation des cartes à disques proportionnels.

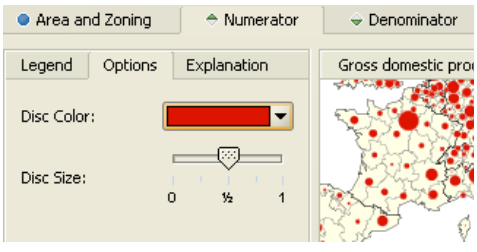


Figure 3-41 : Interface de personnalisation des cartes à disques.

Dans le cas de la carte de ratio, il est possible de choisir le nombre de classes, c'est-à-dire le nombre de plages de valeurs utilisées pour la discrétisation, ainsi que la couleur de la palette et le sens du dégradé. Par défaut, la discrétisation nécessaire à l'établissement de cette carte choroplèthe est réalisée grâce à un quantile basé sur le nombre d'unités territoriales. Cependant, l'utilisateur peut choisir de baser la discrétisation sur le numérateur ou sur le dénominateur. La Figure 3-42 montre les cas d'utilisation associés à la carte de ratio. La Figure 3-43 présente l'interface de personnalisation de la carte de ratio.

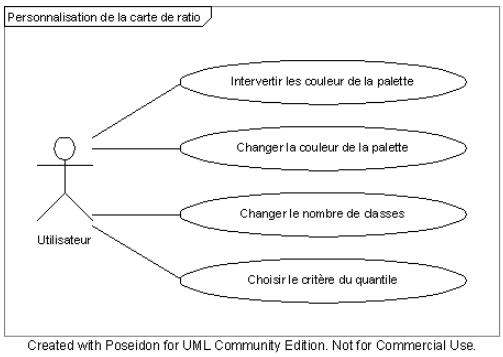


Figure 3-42 : Diagramme de cas d'utilisation pour la personnalisation de la carte de ratio.

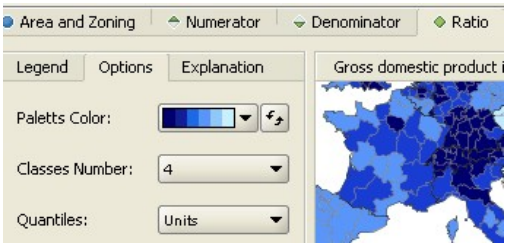


Figure 3-43 : Interface de personnalisation de la carte de ratio.

Les cartes de déviation offrent quasiment les mêmes possibilités de personnalisation que celles offertes par la carte de ratio, offrant en plus un choix sur la méthode de discrétisation utilisée dans les cartes. Le découpage des plages de valeurs des déviations peut être soit arithmétique, soit géométrique.

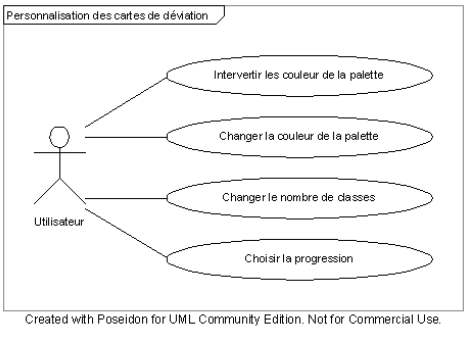


Figure 3-44 : Diagramme de cas d'utilisations pour la personnalisation des cartes de déviation.

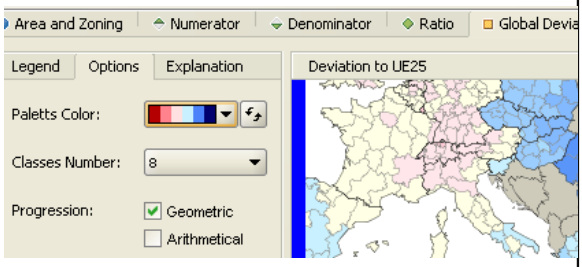


Figure 3-45 : Interface de personnalisation des cartes de déviation.

La carte de synthèse offre des personnalisations différentes par rapport aux autres cartes choroplèthes de l'application car son rôle est différent. L'utilisateur ne choisit pas les couleurs de la palette, mais la manière dont elles sont affectées aux unités, en modifiant la valeur du seuil et l'opérateur utilisés pour la discrétisation. Les figures Figure 3-46 et Figure 3-47, présentent pour l'une les cas d'utilisation, et pour l'autre l'interface proposée pour les réaliser.

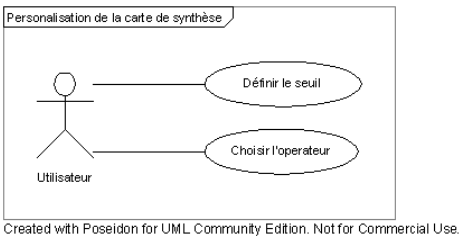


Figure 3-46 : Diagramme de cas d'utilisation pour la personnalisation de la carte de synthèse.

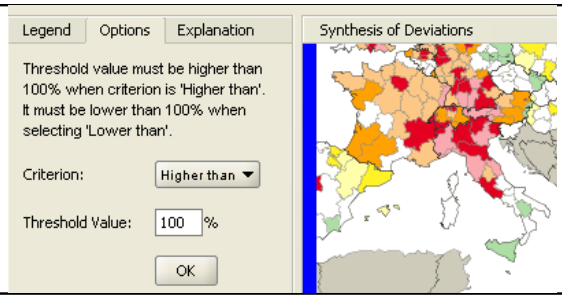


Figure 3-47 : Interface de personnalisation de la carte de synthèse.

Maintenant que nous avons présenté l'application dans son ensemble, nous pouvons désormais nous intéresser aux techniques mises en œuvre dans celle-ci. La partie suivante a donc pour objectif de présenter au lecteur ces techniques et les technologies mises en œuvre.

3.2.2 Techniques employées

Nous étudions plus en détail la conception et la réalisation de cette version d'*HyperAtlas*. Pour cela, et après avoir offert au lecteur un aperçu des technologies, cette partie présente l'architecture générale de l'application, pour finalement approfondir certains points de cette architecture tels que le mécanisme évènementiel de l'application, les objets à instance unique…

Choix des technologies

Pour réaliser une application cartographique répondant aux spécifications énoncées précédemment, une étude comparative entre les diverses techniques de rendu cartographique a été réalisée. Les solutions comparées étaient d'un côté SVG, les librairies graphiques JAVA standard (Java2D) de l'autre. Trois scénarios furent envisagés : l'utilisation de SVG en natif, une solution pure Java, et enfin une utilisation hybride de Java et SVG.
Le manque de maturité de SVG au moment de l'étude orienta la décision en faveur d'une solution 100% Java. L'interface présentée précédemment est donc réalisée à partir des librairies Swing, JGoodies et Java2D.
HyperAtlas n'exploite cependant pas un mode client-serveur et n'est pas accessible via le Web, *HyperAtlas* s'exécute donc comme une application locale à un poste ce qui implique que les données ne sont donc pas fournies par un serveur mais elles sont directement placées dans le répertoire d'installation de l'application. Ces données se matérialisent sous la forme d'un fichier d'objets sérialisés, celui-ci est lu au démarrage de l'application pour permettre le chargement en mémoire des données.

Architecture générale

L'architecture de *HyperAtlas* suit un découpage en couches. Pour ce faire, les classes sont réparties en 3 catégories :

- Les classes graphiques correspondent aux composants graphiques visibles dans l'interface utilisateur.
- Les classes techniques regroupent l'accès aux données et les paramètres, ainsi que les échanges entre composants.
- Les classes métiers regroupent la logique métier et la logique applicative. Celles-ci effectuent les calculs nécessaires au rafraîchissement d'une carte lorsque un paramètre est modifié.

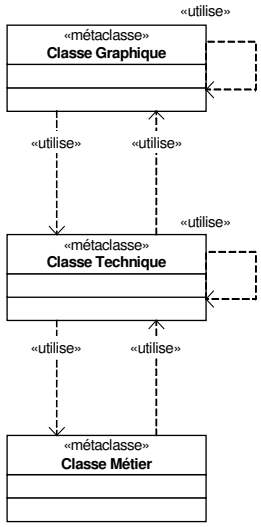


Figure 3-48 : Diagramme de classes de l'architecture logicielle

La Figure 3-48 [Cuenot, 05] représente cette architecture en couches et montre également leurs interactions.
Cette architecture présente de nombreux avantages en termes de développement, tests, maintenance et évolutivité. L'utilisation de couches applicatives indépendantes permet, notamment, un développement autonome de chacune d'elles. Il en va de même pour les tests : chaque partie peut être testée de façon indépendante et en parallèle. Il n'est pas nécessaire d'attendre la fin de toutes les programmations pour débuter les tests. Enfin, cela simplifie la maintenance et la mise à jour de l'application car une intervention sur l'une de ces couches n'entraîne pas de répercussions majeures sur les autres.
Un découpage d'une couche en sous-couches est également possible offrant les mêmes avantages que ceux cités précédemment, mais avec une granularité plus fine.

Principes généraux

Dans cette partie, nous présentons les principes généraux de l'architecture logicielle, ceux-ci sont mis en œuvre dans l'implémentation de la version 1.0.0.c.

Indices

Dans sa version 1.0.0.c, *HyperAtlas* implémente un mécanisme d'indexation, permettant la gestion des diverses cartes à afficher. Ainsi les classes directement nécessaires à l'affichage d'une carte telle que la classe de la carte, sa légende, son panneau d'option, etc. possèdent un attribut nommé `mapIndex`. Lors de la sélection d'un onglet (et donc d'une carte), seules les instances dont l'attribut `mapIndex` correspond à la carte sélectionnée, sont affichées. Ce mécanisme d'indexation est également utilisé pour tous les paramètres spécifiques à une carte tels que la taille des disques pour la carte du numérateur ou la palette de couleurs de la carte de déviation locale.

Les paramètres

Deux types de paramètres existent dans l'application, ils diffèrent par leur portée qui est :

- soit globale à l'application, comme la valeur du zoom, les stocks sélectionnés, les vecteurs de déplacements, etc. Ces paramètres communs concernent toutes les cartes de l'étude.
- soit limitée à une carte, comme la taille ou la couleur des disques, le nombre d'éléments d'une palette ou le type de progression, etc.

L'utilisation du patron Singleton

Un patron de conception décrit une solution générale et standardisée à un problème récurrent. La solution proposée est adaptable à un contexte particulier [Gamma et al., 99].
Le patron Singleton, proposé par Gamma, a pour objectif de garantir qu'une classe a une instance unique. Ce patron offre plusieurs avantages :

- un accès contrôlé à une instance unique,
- la diminution des espaces de noms en regroupant des variables globales,
- la possibilité de raffinement par l'utilisation de sous-classes,
- le choix du nombre d'instances possibles,
- une flexibilité accrue par rapport aux méthodes statiques.

L'utilisation du patron Singleton est appropriée car, comme nous l'avons vu précédemment des classes graphiques de l'application partagent des données, des comportements, ou encore des événements, par l'intermédiaire de classes dédiées (les classes techniques et les classes métiers).
Ces classes spécifiques ne doivent être instanciées qu'une seule fois et être persistantes. Ces propriétés sont garanties par l'utilisation de ce patron. Pour implémenter ce patron, les classes concernées possèdent leur unique instance en attribut privé et implémentent une méthode `getInstance()` qui retourne cette instance unique si elle existe et qui la crée dans le cas contraire (*cf.* Code 3-1).

Ce principe permet :

- de stocker des valeurs dans les attributs de classes dédiées,
- de ne pas avoir à vérifier si ces classes ont déjà été instanciées,
- d'éviter de ré-instancier ces classes très utilisées dans l'application.

```
public class Logic implements IGlobalEventListener,
IIndexedEventListener {
    /**
     * <code>logic</code> is the single Instance used for the
     * singleton pattern.
     */
    private static Logic logic = null;

    /**
     * Return the logic instance if it exist else create the
     * single instance.
     * @return the instance of logic singleton
     */
    public static Logic getInstance() {

        if (logic == null) {
            logic = new Logic();
        }
        return logic;
```

```
    }

    // ...
}
```

Code 3-1 : Extrait de la classe *hypercarte.Logic* utilisant le principe d'instance unique.

La classe *hypercarte.Logic* partiellement présentée par le Code 3-1 montre un extrait du code Java de cette classe. La classe *Logic* regroupe l'ensemble des méthodes métiers utiles à l'application, comme le calcul des déviations on encore le calcul des discrétisations… Comme nous pouvons le constater, cette classe n'implémente aucun constructeur accessible depuis une classe tiers ; le seul moyen d'accéder à une instance est donc d'utiliser la méthode `getInstance()` qui crée une instance, si elle n'existe pas déjà, et qui retourne finalement l'instance unique.

Ce patron est implémenté dans trois classes distinctes : La classe Logic contenant la logique métier (couche métier), la classe *hypercarte.config.Settings* qui regroupe les paramètres de l'application (couche technique), le répartiteur ou *dispatcher* d'événements (couche technique). Nous allons étudier plus en détail le dispatcher d'événements dans la partie suivante.

La communication inter-composants

Un dialogue inter-composants est défini comme un ensemble d'échanges entre des instances de classes d'*HyperAtlas.* Les *inner-classes*, c'est-à-dire les classes définies à l'intérieur d'autres classes, sont exclues [Cuenot, 05]. La communication entre les composants est dite événementielle car elle établit une méthode de propagation d'événements entre un émetteur et un ou plusieurs écouteurs. Dans le cadre de *HyperAtlas*, on parle même de « communication inter-composants événementielle anonyme » (CICEA) car la propagation des événements est anonyme, un émetteur ne connaissant pas la liste des récepteurs de l'événement et inversement, un récepteur à l'écoute d'un événement n'en connaît pas la provenance.

Avant d'étudier plus en détail les mécanismes de répartition des événements, il est nécessaire de comprendre quels sont les divers événements mis en oeuvre. *HyperAtlas* exploite trois types d'événements :

- Les événements globaux : concernent la globalité de l'application. Il s'agit par exemple, d'un événement annonçant un zoom. Effectivement un zoom concerne l'ensemble des cartes de l'application, mais également d'autres composants comme la barre d'échelle ou la tirette du zoom (zoomSlider)...

- Les événements indicés : concernent une carte donnée. C'est, par exemple, le cas des événements liés à une palette, comme le changement du nombre de classes ou bien le changement de couleur. Ce type d'événement ne concerne donc qu'une carte dans l'application, ainsi que les éléments qui lui sont directement liés, comme sa légende ou ses options.

- Les événements de type messages : permettent d'offrir une remontée d'information à l'utilisateur par le biais de l'affichage d'un message. Par exemple, pour la déviation moyenne -lorsqu'un utilisateur tente de définir un maillage inférieur ou égal au

maillage en cours d'étude- un événement message est envoyé pour que l'application notifie l'utilisateur que le paramétrage demandé est impossible.

Si ces trois types d'événements sont bien distincts, il n'en demeure pas moins qu'ils sont des événements. Pour cette raison, une classe abstraite nommée `AbstractEvent` existe dans *HyperAtlas*, cette classe implémente toutes les méthodes et attributs communs aux événements, après quoi, chaque événement spécialise cette classe abstraite en lui rajoutant ses méthodes ou attributs propres, comme par exemple la chaîne de caractères contenant le message pour les événement de ce type. La Figure 3-49 présente un diagramme de classes de la partie événementielle de *HyperAtlas* obtenue grâce à une application de rétro ingénierie.

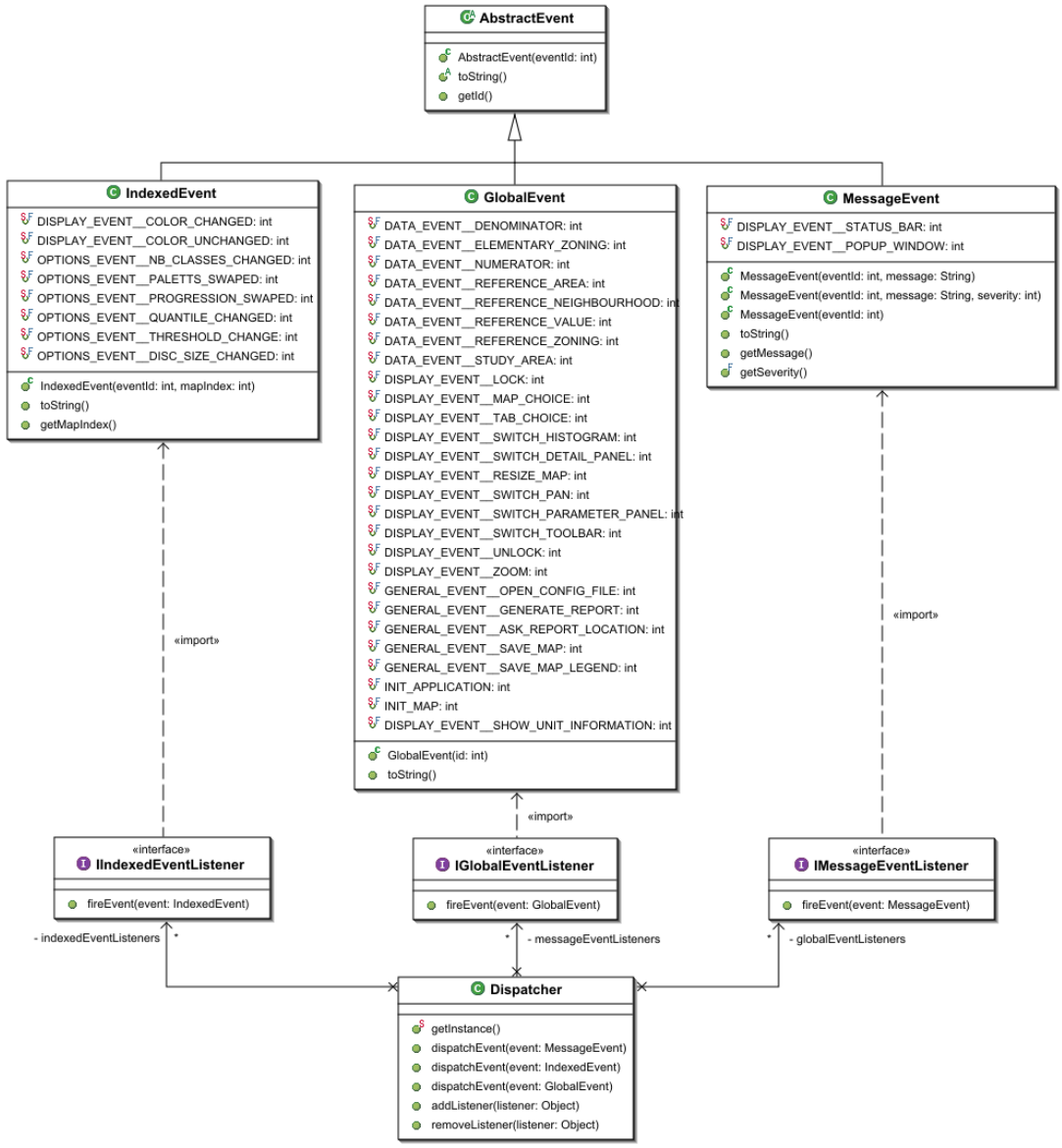


Figure 3-49 : Diagramme de classe de la partie événementielle de *HyperAtlas*.

La Figure 3-49 montre également que chaque événement possède, par le biais de sa super classe, un attribut numérique `id` (identifiant) qui peut prendre pour valeur une constante identifiant un événement.

L'élément essentiel pour réaliser cette architecture est le répartiteur d'événements (`hypercarte.event.Dispatcher`), une instance unique de cette classe collecte et redistribue les événements de l'application [BISS, 04]. Celui-ci gère des listes d'écouteurs, une par interface (par type d'événement). Le listing Code 3-2 présente les trois listes utilisées par le *Dispatcher*.

```
    private HashSet eventListeners = null;
    private HashSet messageEventListeners = null;
    private HashSet indexedEventListeners = null;
```

Code 3-2 : Listes d'écouteurs de la classe `hypercarte.Event.Dispacter`.

Plusieurs étapes sont nécessaires à la mise en place de cette architecture :

1- L'application crée une instance du *Dispatcher*.

2- Les classes souhaitant être notifiées des événements implémentent les interfaces correspondant aux écouteurs des événements qui les concernent.

3- Ces classes s'enregistrent auprès du *Dispatcher* grâce à sa méthode `addListener()` (*cf.* Code 3-3) .

```
    /**
     * Registers an object as listener of events.
     *
     * @param listener
     *  An EventListChangeListener that will be warned each time
     *  the list of events corresponding to the various criteria
     *  changes.
     */
    public void addListener(Object listener) {
        if (listener instanceof IGlobalEventListener) {
            this.eventListeners.add(listener);
        }
        if (listener instanceof IIndexedEventListener) {
            this.indexedEventListeners.add(listener);
        }
        if (listener instanceof IMessageEventListener) {
            this.messageEventListeners.add(listener);
        }
    }
```

Code 3-3 : La méthode `addListener()` de la classe `hypercarte.Event.Dispacter`.

4- Lorsqu'un objet souhaite envoyer un événement, il invoque l'une des méthodes `dispatchEvent()` de l'instance unique du *Dispatcher*. Le *Dispatcher* parcourt alors la liste d'écouteurs concernés pour les notifier, (*cf.* Code 3-4).

5- Les écouteurs sont notifiés par l'appel de leur méthode `fireEvent()` et réagissent en conséquence.

```
/**
 * An action occured, warn all the registered listeners.
 *
 * @param event
 *            An event constant number
 */
public void dispatchEvent(MessageEvent event) {
    Iterator listenerIterator = this.messageEventListeners.iterator();
    IMessageEventListener eventListener;

    while (listenerIterator.hasNext()) {
        eventListener = (IMessageEventListener) listenerIterator.next();
        eventListener.fireEvent(event);
    }
}
```

Code 3-4 : La méthode `dispatchEvent` de la classe `hypercarte.Event.Dispacter`.

La Figure 3-50 résume les étapes de la propagation d'un événement dans l'application.

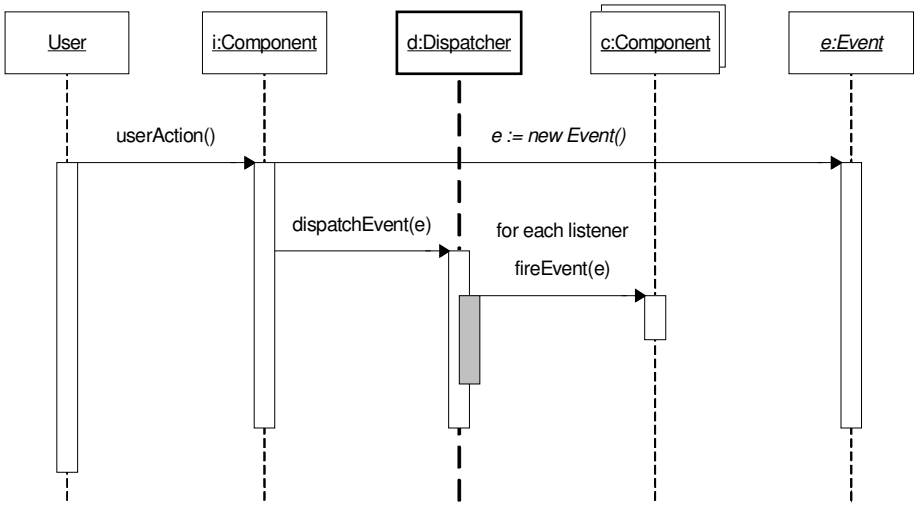


Figure 3-50 : Diagramme de séquences illustrant la propagation d'un événement.

Les cartes dans l'application

Dans les versions antérieures de l'application, le composant graphique permettant l'affichage des cartes était un « *Java Bean* » utilisant des tests conditionnels pour permettre le choix du type de carte et donc des opérations graphiques à réaliser. Dans la version 1.0.0.c, une solution à base d'héritage a été retenue pour des raisons de performance [Cuenot, 05]. Dans cette version, les cartes et les éléments qui leur sont liés sont regroupés à l'aide d'un indice. Lors de leur création, un branchement conditionnel est effectué pour déterminer quel type de carte instancier. Le diagramme de classes (Figure 3-51), montre l'arbre d'héritage des objets représentant les cartes de l'application. Au sommet de cet arbre d'héritage, on retrouve un composant graphique `IndexedPanel` qui spécialise un panneau graphique en lui rajoutant la notion d'index présentée précédemment. Ce composant est ensuite sous-classé en fonction des besoins : cartes, légendes… Ce composant est d'ailleurs spécialisé par la classe abstraite `AbstractMap` possédant toutes les méthodes et tous les attributs nécessaires au dessin d'une carte. Cette classe est cependant abstraite, car certaines méthodes, telles que la gestion des événements souris, sont à définir spécifiquement pour les cartes. Par exemple, pour ouvrir une fenêtre d'histogramme lors du clic sur une unité de la carte de synthèse. Ce comportement est propre à la carte de synthèse et ne se retrouve pas dans les autres types de cartes. L'abstraction de ces méthodes n'est pas strictement nécessaire mais elle offre un nommage normalisé des méthodes à implémenter, et fournit donc au développeur un canevas le guidant dans son implémentation. La classe `Map` spécialise et concrétise la classe `AbstractMap` fournissant ainsi toute les méthodes nécessaires au dessin d'une carte de l'application. Cette classe est elle-même spécialisée selon les types de cartes de l'application, ainsi la classe `ContextMap` dérive de la classe `Map` et surcharge la méthode `paintMap()` ce qui permet à la carte de contexte de dessiner les frontières des unités territoriales de plus haut niveau hiérarchique, comme par exemple, les frontières des états pour l'union européenne. La classe `DiscMap` (carte à disques) spécialise la carte basique (`Map`) en ajoutant le tracé et le remplissage des disques. Les classes `IndicatorMap` et `DeviationMap` permettent le remplissage des unités territoriales avec une couleur résultant de la fonction de discrétisation.

Les classes `NumeratorMap` et `DenominatorMap` spécialisent la classe `DiscMap` et surchargent sa méthode `paintMap()` pour permettre le tracé des disques proportionnels, au numérateur pour la première, et au dénominateur pour la seconde.

Les classes `GlobalDeviationMap`, `MediumDeviationMap` et `LocalDeviationMap` spécialisent la classe `DeviationMap` en surchargeant la méthode `fireEvent()`. Elles implémentent ainsi la gestion d'événements appropriée au type de carte.

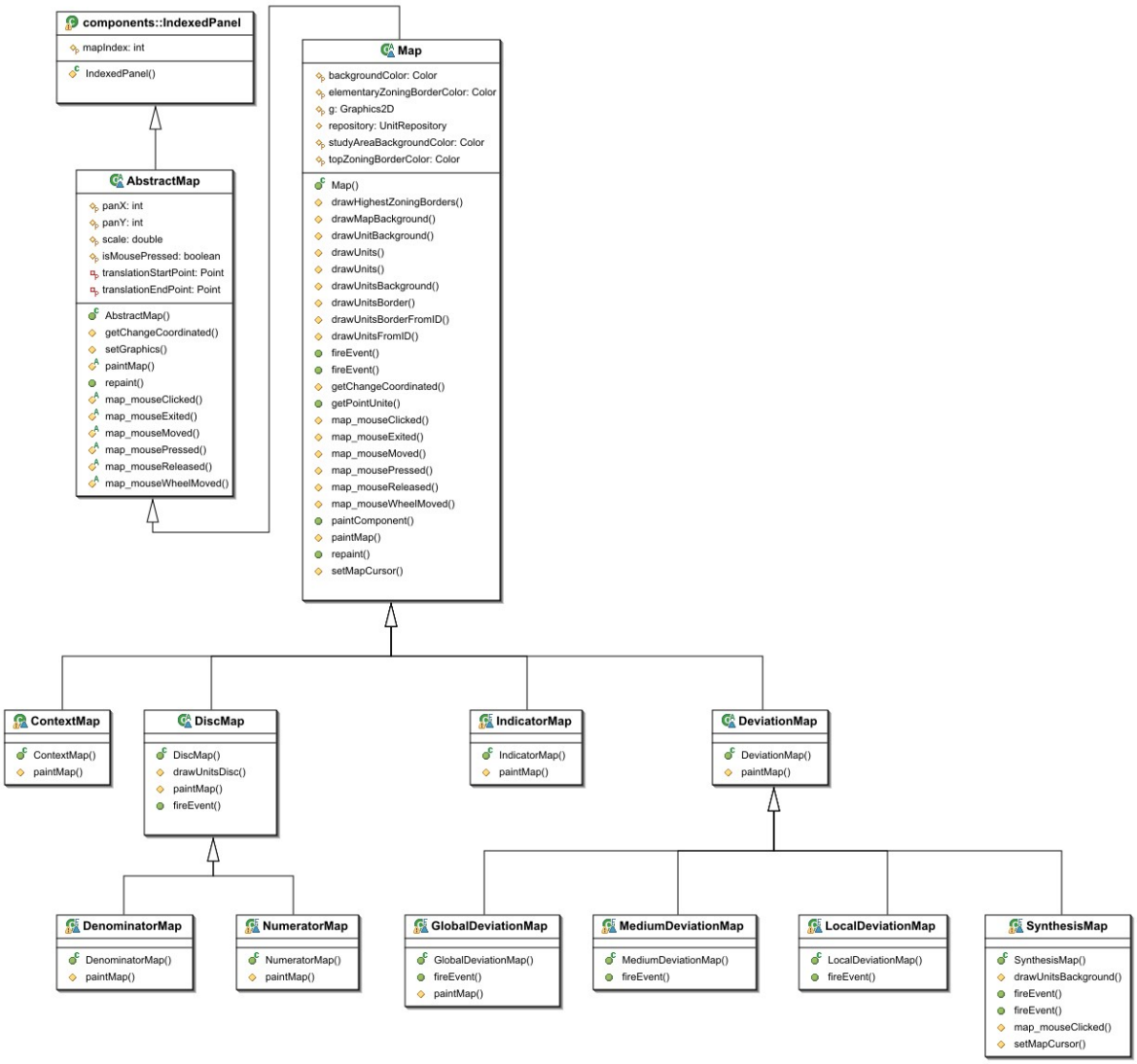


Figure 3-51 : Diagramme de classes des cartes implémentées dans *HyperAtlas*.

Les légendes liées aux cartes utilisent les mêmes principes, la Figure 3-52 détaille l'organisation des classes implémentant ces légendes. On retrouve sur cette figure la notion de spécialisation permettant un meilleur raffinement des objets mis en œuvre.

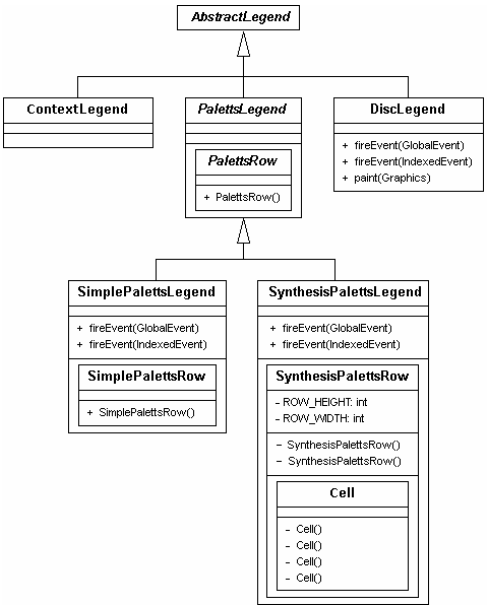


Figure 3-52 : Diagramme de classes des légendes implémentées dans *HyperAtlas*.

La structure de données

Nous nous intéressons maintenant à la manière dont les données sont traitées par l'application. Pour cela, nous étudions les structures mises en oeuvre pour leur traitement. Les structures présentées sont notamment celles qui sont sérialisées puis stockées sous la forme d'un fichier. Celui-ci est lu lors du lancement de l'application, les objets sont alors chargés en mémoire ce qui autorise le lancement de l'application.

Les classes utilisées pour représenter les données sont regroupées dans le package `hypercarte.data`. Il contient les classes permettant le stockage et l'agrégation des données géographiques et attributaires, celles permettant la sérialisation et la désérialisation, et enfin, celles pour l'accès aux fichiers textuels et au fichier sérialisé.

Exemple : détail du lancement de l'application

Lors du lancement de l'application, la première étape réalisée par celle-ci est la lecture du fichier d'objets sérialisés. Si le fichier existe et qu'il est valide, l'application poursuit alors son chargement ; dans le cas contraire une exception est générée et un message d'erreur est renvoyé à l'utilisateur. Après avoir lu le fichier, *HyperAtlas* commence l'initialisation des paramètres globaux par l'appel à la méthode `getInstance()` de la classe `Settings` présentée précédemment. Le paramétrage du gestionnaire d'apparence (`UIManager`) peut alors commencer, le gestionnaire d'apparence est une classe standard de l'API[14] *Swing*[15], qui permet de définir les caractéristiques par défaut de l'interface graphique. Ce paramétrage englobe le choix de la couleur de fond des fenêtres, la police, la taille et la couleur des textes utilisés dans l'interface, etc.
Cette étape terminée, l'interface graphique est alors créée. Pour cela, l'application commence par créer le panneau central de la fenêtre. Celui-ci est divisé en deux parties : le panneau de paramètres (élément 3.2.1 de la Figure 3-38), et le panneau à onglet contenant les cartes et leur légendes (élément 3.2.1 de la Figure 3-38). Un panneau indicé, nommé `Frameset` est alors créé, prenant en indice le numéro de l'onglet auquel il appartient. Ce panneau contient notamment les panneaux 3.2.1, 3.2.1 et 3.2.1 présentés dans la Figure 3-

[14] API est l'abréviation de *Applications Programming Interface* ou Interface de Programmation d'Applications.
[15] *Swing* est un paquetage graphique standard propre au langage Java.

38. C'est lors de la construction de la classe `Frameset` que survient la spécialisation des classes de carte. Cette spécialisation est réalisée à l'aide d'un branchement conditionnel sur l'index de la carte (*cf. Code 3-5*).

```
switch (this.mapIndex) {
            case Settings.MAP_CONTEXT:
                map = new ContextMap(this.mapIndex);
                break;
            case Settings.MAP_NUMERATOR:
                map = new NumeratorMap(this.mapIndex);
                break;
            …

            case Settings.MAP_SYNTHESIS:
                map = new SynthesisMap(this.mapIndex);
                break;
            default:
                map = new ContextMap(this.mapIndex);
                break;
            }
```

Code 3-5 : Utilisation de branchements conditionnels pour l'instanciation des cartes appropriées.

Notons que les composants créés s'enregistrent lors de leur création au dispatcher d'événements. De ce fait, l'application est fonctionnelle après la phase de création de son interface.

3.2.3 Champs d'exploration

Après avoir étudié la version 1.0.0.c, des axes d'amélioration ont été identifiés, ils concernent l'ergonomie de l'application, son évolutivité ainsi que l'ajout de nouvelles fonctionnalités.
Ces axes d'amélioration sont le fruit d'une analyse approfondie de l'application, de sa comparaison à d'autres S.I.G. et de la concertation avec les différents acteurs du projet.

L'ergonomie

L'un des premiers points remarqués lors de l'utilisation de *HyperAtlas* est le manque de contrôle du déplacement dans les cartes. Par exemple, rien n'empêche un utilisateur de se déplacer en dehors des limites de la carte. Il devient donc assez difficile de revenir sur cette carte car aucune fonction de centrage n'est proposée. La Figure 3-53 illustre un déplacement hors des limites de la carte.

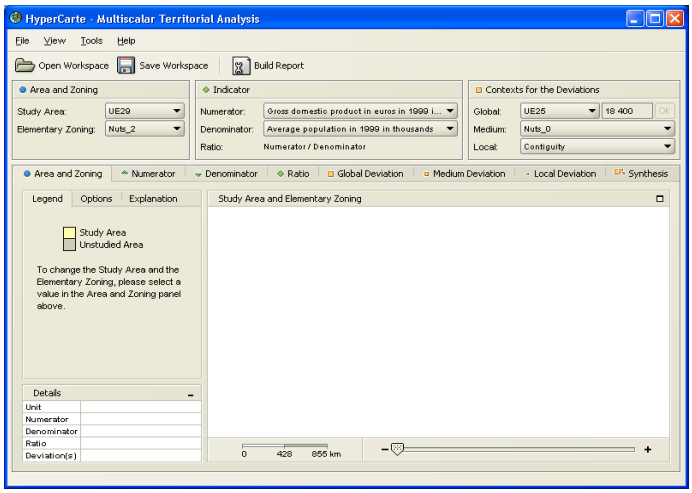
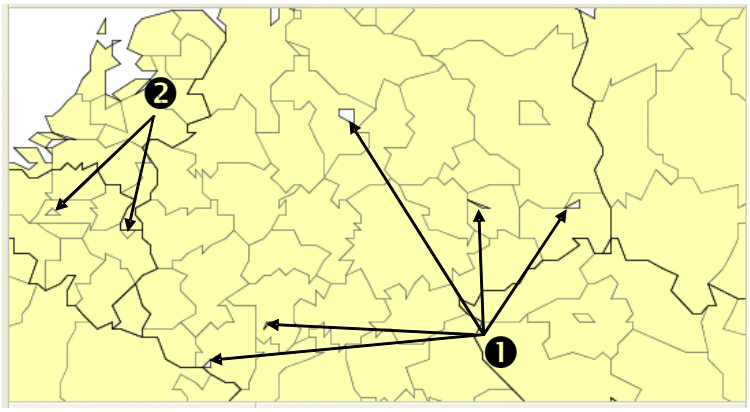


Figure 3-53 : Exemple de déplacement hors des limites de la carte.

Deux solutions peuvent alors être envisagées pour répondre à cette gêne : proposer une fonction de recentrage sur la carte ou l'interdiction des déplacements en dehors des limites de la carte.

Un autre point soulevé par les utilisateurs est le manque de précision des contours. Effectivement, comme le montre la Figure 3-54 , des trous et biffures apparaissent sur les cartes et le contour de certaines unités est réduit à un triangle.



❶ des trous ou biffures apparaissent sur la carte
❷ le contour de certaines unités est réduit à un triangle

Figure 3-54 : Exemples de problèmes de précision des contours.

De plus, lors du survol par la souris d'une unité territoriale, cette dernière n'est pas mise en valeur. Il devient donc difficile de savoir quelle unité est pointée par la souris, c'est notamment le cas pour les unités incluses dans d'autres. Cette fonctionnalité est par ailleurs implémentée par la plupart des outils S.I.G., tels que *GeoClip* (*cf.* chapitre 2.2).

Un point d'amélioration porte également sur l'obtention d'informations sur la composition d'unités territoriales. Par exemple, il est parfois intéressant de savoir rapidement à quel pays appartient une île, ou quels sont les départements qui composent une région …
Pour permettre à un utilisateur de voir aisément les détails d'une unité territoriale, l'idée d'utiliser un menu déroulant semble la plus adaptée, ainsi l'utilisateur peut sélectionner une unité avec le bouton droit de la souris pour faire apparaître un menu déroulant et obtenir des informations sur l'unité et sa hiérarchie de composition.

L'évolutivité

Dans sa version 1.0.0.c, *HyperAtlas* reste difficilement diffusable car cette version n'exploite qu'un seul jeu de données (des données de l'Union Européenne). En effet, cette version a été développée autour d'un jeu de données unique ce qui a eu pour conséquence de lier certains traitements aux données. Prenons un exemple pour mieux appréhender cette dépendance : les unité territoriales sont représentées par un ou plusieurs polygones. Les coordonnées de ceux-ci sont exprimées dans un repère géographique et non dans un repère « écran ». De ce fait, il est nécessaire d'établir une conversion permettant le changement de repère entre le repère géographique et le repère écran. Avant de détailler les transformations géométriques nécessaires notons que les deux repères source et cible sont tous deux orthonormés. La première transformation apportée entre les coordonnées géographiques et les coordonnées écran est une symétrie d'axe D d'équation « $y = f(x) = x$ ». Cette transformation intervertit donc les abscisses et les ordonnées des points. Cette transformation n'intervient qu'a la sérialisation des données. En effet, c'est lors de l'écriture des coordon-

nées que celles-ci sont permutées. Le fichier d'objets sérialisés ne contient donc pas les coordonnées géographiques au sens strict du terme, mais il ne contient pas non plus les coordonnées écran car d'autres étapes sont nécessaires pour le passage des coordonnées géographiques aux coordonnées écran.
Après cette première étape, une translation est appliquée, celle-ci a pour objectif le centrage de la carte à l'écran. A ce stade, une rotation d'angle $-\pi/2$ est encore nécessaire pour afficher la carte dans le sens de lecture conventionnel. Enfin, une homothétie est utilisée pour la mise à l'échelle afin que la taille des contours des unités territoriales ne soit pas disproportionnée par rapport à la taille de l'écran. L'extrait du code de la fonction `setGraphics` (Code 3-6), présente l'implémentation de la translation, de la rotation et de l'homothétie. Nous pouvons constater que certaines valeurs (en rouge dans l'extrait) ne sont pas calculées mais incluses directement au corps de la méthode.

```
protected Graphics2D setGraphics(Graphics graphics) {

        Graphics2D g = (Graphics2D) graphics;
                                        …

        g.translate(240 + this.panX, 220 + this.panY);  // translation
        g.rotate(-Math.PI / 2);                         // rotation
        g.scale(this.scale * 0.92, this.scale * 0.92);  // homothétie
        g.setStroke(new BasicStroke((float)this.settings.getThickness()));

        return g;
    }
```

Code 3-6 : Extrait du code de la méthode `setGraphics` permettant le changement de repère entre le repère géographique et le repère écran.

La conséquence directe de l'utilisation de ces valeurs statiques est le mauvais affichage d'autres cartes. Effectivement, supposons que nous souhaitons afficher la carte de la Tunisie avec le même système de coordonnées que celui utilisé pour l'Europe. Cette carte ne serait pas centrée à l'écran mais la Tunisie se trouverait alors beaucoup trop basse par rapport à l'écran. Ce problème de centrage étant dû au caractère statique des valeurs utilisées lors de la translation (Code 3-6). De plus, la Tunisie apparaîtrait avec une taille relativement modeste à cause des valeurs fixes utilisées pour l'homothétie. Imaginons maintenant que les coordonnées de la Tunisie ne soient pas exprimées dans le même repère que celui utilisé pour l'Europe : il devient alors impossible de prédire à quoi pourra ressembler la carte !
Dès lors, il devient impératif de calculer ces valeurs en fonction de la carte à représenter. Pour ce faire, l'une des solutions consiste à utiliser le rectangle englobant de la carte. En comparant celui-ci avec le rectangle défini par les coordonnées écran, il est alors possible de déduire la translation et le rapport permettant l'affichage correct de la carte (*cf.* Figure 3-55).

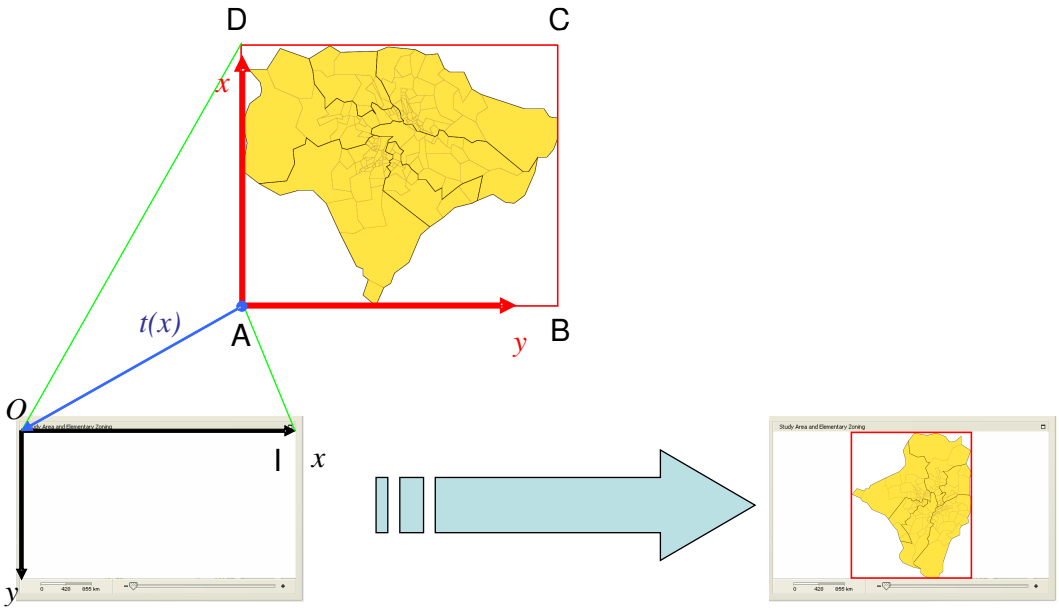


Figure 3-55 : Détermination des transformations nécessaires à l'affichage d'une carte.

L'autre point préjudiciable à l'évolutivité de l'application réside dans la structure de données utilisée. En effet, comme nous l'avons vu précédemment, l'application lit ces données à partir d'un fichier d'objets sérialisé permettant le chargement en mémoire de l'intégralité des données. Cette solution offre certes, un accès rapide aux données, mais trouve également rapidement ces limites, faute d'espace suffisant en mémoire. Typiquement, Java dans sa configuration par défaut n'alloue que 64Mo de mémoire par programme. En général, cet espace est suffisant pour la plupart des applications de gestion, mais dans un contexte de cartographie interactive celui-ci est rapidement saturé par l'ensemble des données à traiter. De plus, la taille des données croît proportionnellement à leur complexité. Ainsi des contours géographiques occupent d'autant plus d'espace qu'ils sont précis. Dans un tel contexte il est naturel de s'interroger sur la nécessité de charger tous les éléments en mémoire, y compris ceux qui n'entrent pas dans le contexte d'étude. Par exemple, si un utilisateur souhaite réaliser une étude sur l'Europe des 15 au niveau de maillage NUTS 1, est-il intéressant de garder en mémoire l'ensemble de contours et stocks de toutes les unités de niveau NUTS 3 ainsi que toutes les informations contenues dans le jeu de données ? La réponse à cette question n'est pas simple car elle dépend du jeu de données et de la capacité mémoire allouée à Java. Une étude est donc nécessaire pour déterminer quelles sont les limites rencontrées par *HyperAtlas* en termes de charge mémoire et de temps de traitement. Rappelons que l'un des objectifs du projet est de pouvoir traiter, dans un premier temps, des données statistiques communales pour la France, pour finalement arriver à traiter des données portant sur l'espace européen au niveau communal (NUTS 5). La première étape de cette étude consiste à quantifier le volume des données actuellement traitées. Pour cela, nous sommes repartis de la version d'*HyperAtlas* en exploitation pour créer une version présentant des statistiques sur les données. Cette version, même si elle n'a pas pour but d'être exploitée, nous apporte des renseignements précieux sur les données lues en entrée. Ce programme nous a permis d'obtenir les chiffres suivants :

	Europe	France
Nombre total d'unités	1662	40700
Nombre de stocks	11	11
Nombre de polygones	5457	42311

Nombre de points	70405	4046772
Moyenne du nombre de points par polygone	12,9	99,43
Médiane du nombre de points par polygone	11	63
Taille des données textuelles en entrée	840Ko	67,4Mo
Taille du fichier d'objet sérialisés	3,4Mo	161Mo

Figure 3-56 : tableau comparatif des données actuellement gérées grâce à *HyperAtlas* (Europe) et les données que l'on souhaite traiter (France).

La lecture du tableau précédent permet de constater l'écart de volume entre les données gérées dans la version actuelle et celles que l'on prévoit de traiter. Nous nous intéressons donc maintenant aux capacités de traitement de l'application. Pour cela, nous modifions cette dernière afin de créer autant d'unités territoriales que possible avec des contours de 12 points et sans indicateurs : avec les 64 Mo alloués par défaut à la JVM, l'application peut traiter 39216 polygones.
Au vu de ces résultats, la première idée des membres du projet fut d'augmenter la mémoire allouée par défaut à la JVM. Nous avons donc modifié le paramétrage mémoire de la machine virtuelle Java en lui permettant d'accéder jusqu'à 512Mo de mémoire.
Cependant, les résultats obtenus ne furent pas satisfaisants, ni en termes de qualité des cartes produites (*cf.* Figure 3-57), ni en matière de temps de traitement (plusieurs heures).

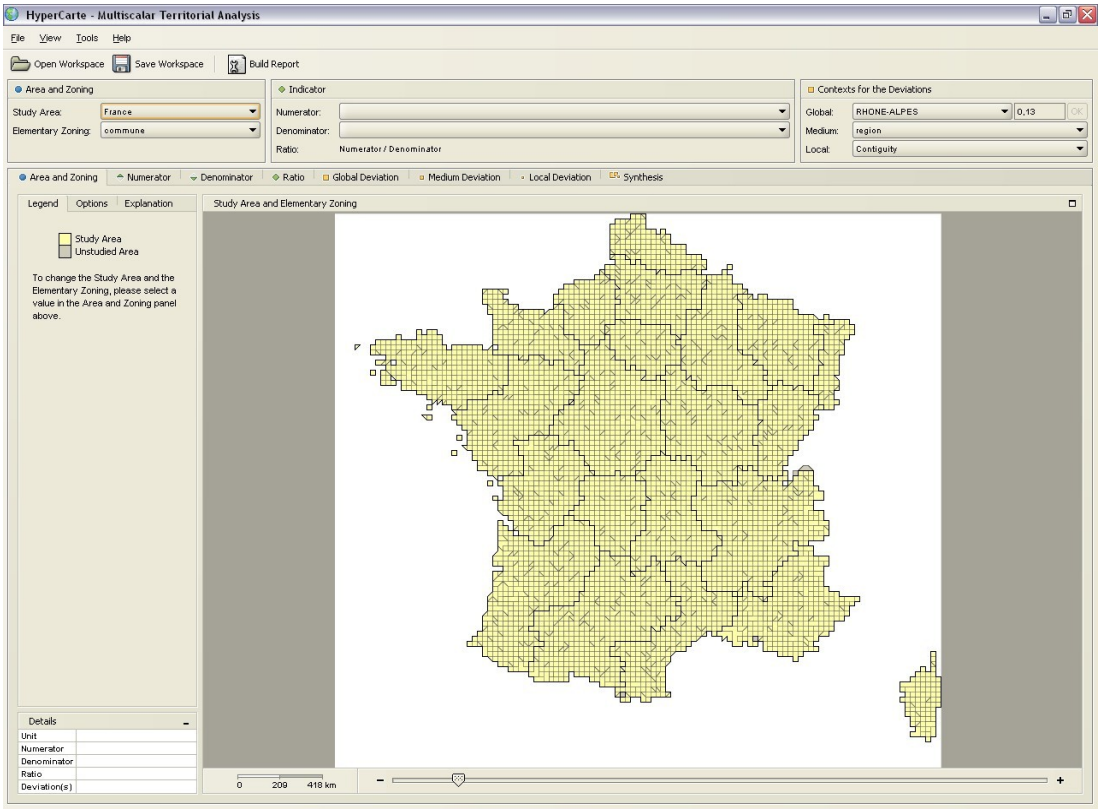


Figure 3-57 : Problèmes de qualité des contours lors de l'affichage de la carte de France.

De plus, les limites posées par le mécanisme de sérialisation d'objets ne s'arrêtent pas aux problèmes de mémoire, la sérialisation devient rapidement contraignante pour l'évolutivité d'un logiciel [Bloch, 02]. En effet, rappelons que le but de la sérialisation est de transcrire un objet en un flux d'octets et que la désérialisation consiste à retrouver un objet à partir

d'un flux d'octets. Lors de la désérialisation, il devient nécessaire de pouvoir connaître le type correspondant au flux de données traité. Pour réaliser l'association entre des données binaires, représentant l'état d'un objet et la classe ou type de cet objet, Java implémente la notion d'identifiant unique de version ou *serial version UID*[16] . Cet Identifiant représente une classe sérialisable, il est calculé dynamiquement en fonction : du nom de la classe, de ses méthodes publiques et protégées et des interfaces implémentées. Lors de la sérialisation le *serial version UID* est ajouté automatiquement au flux de données. Ainsi, lorsque l'on souhaite typer un objet qui vient d'être lu à partir d'un flux, la JVM compare l'*UID* associé à l'objet avec celui de la classe cible. Si les deux correspondent, le typage s'effectue, dans le cas contraire une exception est levée. Ce fonctionnement n'est pas anodin car il implique que toute modification de l'un des paramètres utilisé pour le calcul du *serial version UID*, entraîne une incompatibilité avec les objets sérialisés avec une version de classe antérieure.

A partir de ces constats, la décision fut prise d'apporter des modifications sur la gestion des données dans l'application. L'objectif de ces dernières étant double :

- conserver la compatibilité avec les anciens jeux de données,
- permettre à l'application d'exploiter de nouvelles sources de données (bases de données, réseaux...)

Les fonctionnalités attendues

Lors des diverses présentations de l'application *HyperAtlas* aux différents acteurs concernés par l'analyse spatiale de phénomènes sociaux, une question était récurrente : " *Comment charge-t-on nos propres données ?* ".
Si cette question semble anodine, elle soulève néanmoins deux problèmes importants :

- L'application peut-elle gérer d'autres données ?
- Comment charger ces données de manière simple et conviviale ?

La réponse à la première question nous est donnée dans la partie précédente, à savoir que la prise en charge de nouvelles données ne se fera pas sans modifications préalables.
Pour répondre au second point, il est essentiel de comprendre la complexité des données et le besoin de garantir la cohérence de celles-ci. Par exemple, les contours des unités territoriales doivent respecter certaines contraintes telles que la fermeture : une ligne ne peut représenter une unité territoriale. De plus, le contrôle de cohérence des données lues en entrée ne se limite pas aux contours, il inclut également la structure des données. A titre d'exemple, rappelons qu'une unité territoriale ne peut avoir qu'un seul parent.

Pour répondre à ces besoins d'ajout de données, il a été décidé de fournir aux utilisateurs d'*HyperAtlas* une interface permettant l'ajout ou la modification de stocks au jeu de données en cours d'utilisation. Le besoin de création de nouveaux jeux de données pour l'application a donné naissance à un nouveau projet : le projet *HyperAdmin*, présenté plus loin dans ce mémoire.
A ce stade de la définition des nouvelles fonctionnalités, le redémarrage de l'application est encore nécessaire au chargement d'un nouveau jeu de données. Il est alors apparu opportun aux membres du projet d'inclure la possibilité de charger dynamiquement des jeux de donnée dans l'application.

L'hétérogénéité des profils utilisateurs a également conduit à des besoins de personnalisation des cartes produites. Si *HyperAtlas* a pour vocation d'être accessible et utilisable par le plus grand nombre, il garde néanmoins pour ambition d'être exploitable par des experts tel

[16] Serial Version UID : identifiant unique de version.

que chercheurs, géographes, statisticiens… Des requêtes concernant les possibilités de personnalisation des cartes ont donc été formulées. Ces requêtes portent notamment sur des éléments de représentation ou d'ergonomie (personnalisation des contours, zoom sur une unité territoriale, …), ainsi que sur des besoins portant sur la méthode de génération des cartes. Parmi ceux-ci, nous pouvons citer le besoin d'édition manuelle des seuils utilisés par les palettes des cartes de déviation. A l'heure actuelle, ces seuils sont calculés à l'aide de quantiles (*cf.* paragraphe 3.2.1). Or une édition manuelle de ces seuils peut permettre de mettre en valeur certains phénomènes ou d'harmoniser les seuils entres les diverses cartes de déviation. La possibilité de définir son propre voisinage entre unités territoriales a également été évoquée, l'idée sous jacente étant qu'un utilisateur pourrait fournir à l'application une matrice de voisinage à utiliser pour le calcul des déviations locales. Par exemple, au lieu d'étudier la déviation d'une unité par rapport aux unités qui lui sont contiguës, il serait alors possible d'étudier la déviation de cette même unité par rapport aux unités accessibles en moins d'une heure de temps.

En plus des fonctionnalités présentées, la possibilité de "recomposer" les unités territoriales a été envisagée. Il s'agit d'offrir la possibilité à l'utilisateur d'éditer l'arbre de composition des unités territoriales, lui permettant ainsi de faire des simulations. Un utilisateur pourrait ainsi rattacher le Val d'Aoste à la région Rhône-Alpes pour ensuite visualiser les impacts de ce changement. Un organisme s'occupant de l'aménagement du territoire, tel que ESPON, trouverait alors dans l'outil une aide à la décision.

La Figure 3-58 offre un récapitulatif des fonctions attendues dans les futures versions de *HyperAtlas,* sous la forme d'un diagramme UML de cas d'utilisation.

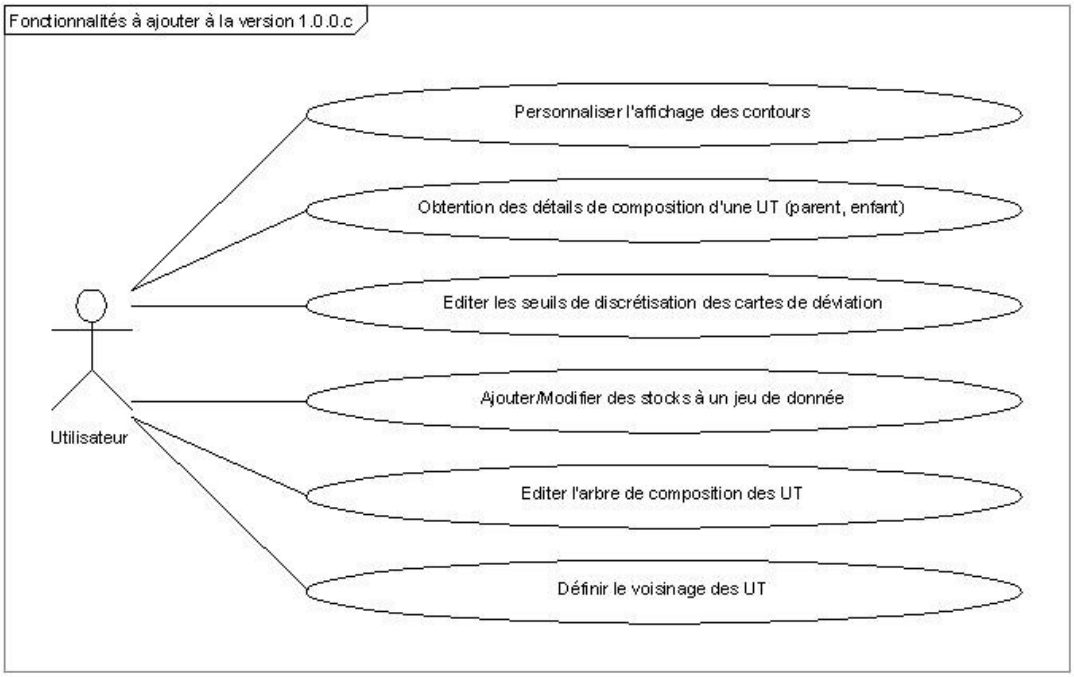


Figure 3-58 : Diagramme de cas d'utilisation décrivant les fonctionnalités à apporter à la version 1.0.0.c

Dans les parties suivantes nous nous intéressons aux diverses améliorations et fonctionnalités apportées à l'application.

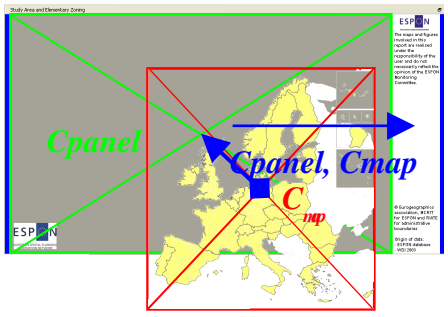
3.3 Améliorations ergonomiques

Comme nous l'avons vu dans le chapitre précédant (chapitre 3.2), l'analyse de la version 1.0.0.c nous a permis d'identifier des axes d'amélioration en termes d'ergonomie. Parmi ces améliorations nous avons choisi d'en présenter trois dans ce chapitre.

3.3.1 Contrôle des déplacements

Dans la version 1.0.0.c du logiciel *HyperAtlas,* aucun mécanisme ne garantit qu'une carte ne soit tracée en dehors des limites du panneau d'affichage ou même de l'écran (*cf.* Figure 3-53). Ce comportement s'avère rapidement gênant lors de l'utilisation du logiciel, d'autant plus qu'aucune fonctionnalité de recentrage de la carte n'est disponible.
Pour palier cette gêne, il a été décidé, dans un premier temps, d'implémenter une fonctionnalité de recentrage sur la carte, similaire à celle présente dans de nombreux outils cartographiques. Puis, dans un second temps, nous avons réfléchi à un moyen d'empêcher les déplacements en dehors des limites de la carte.
Dans les deux cas ces solutions nécessitent la création de limites sur une carte, pour cela nous avons naturellement choisi d'utiliser le rectangle englobant la carte. Ce rectangle est obtenu à partir de des points de coordonnées ($x_{\min}$, $y_{\min}$) et ($x_{\max}$, $y_{\max}$) qui représentent respectivement : le point ayant l'abscisse et l'ordonnée minimale et le point dont l'abscisse et l'ordonnée sont maximale.
Pour permettre le recentrage de la carte :

- on calcule les coordonnées du centre C_{map} de la carte (intersection des diagonales du rectangle englobant),
- on calcule les coordonnées du centre C_{panel} du panneau représentant la carte,
- on effectue une translation de vecteur $\overrightarrow{C_{map}C_{panel}}$



Même si la fonction de recentrage sur la carte permet à un utilisateur de "se repérer", l'idéal est bien entendu de lui éviter de se perdre... Dans cette optique nous avons souhaité empêcher les déplacements conduisant à faire sortir la carte de l'écran ou de son panneau affichage.
Nous décrivons dans la partie suivante la description de la méthode que nous avons mise en place pour contrôler les déplacements. Pour cela, nous définissons un déplacement comme une translation quelconque dans le référentiel écran. Rappelons qu'un écran constitue un repère orthonormé. A ce titre, nous pouvons exprimer notre déplacement comme la combinaison de deux translations : la première (t_x) selon l'axe des abscisses, la seconde (t_y) selon l'axe des ordonnées.
Pour des raisons de simplicité, nous ne nous intéresserons qu'à la limitation des déplacements horizontaux, la limitation des déplacements selon l'axe des ordonnées étant réalisé de manière analogue.
Pour limiter les déplacements horizontaux, il convient de distinguer deux cas :

- la largeur de la carte est inférieure ou égale à la largeur de sa zone d'affichage,
- la largeur de la carte est supérieure à la largeur de sa zone d'affichage.

Dans le premier cas, il ne faut pas autoriser de déplacements car la carte doit être centrée horizontalement par rapport à son conteneur (*JPanel*).
Dans le cas contraire, nous devons veiller à ce que le point de la carte ayant l'abscisse maximale n'ait jamais une valeur d'abscisse inférieure à celle du point d'abscisse maximale de son conteneur. Il en est de même pour les points d'abscisses minimaux, celui de la carte doit toujours être inférieur à celui du conteneur de la carte. La Figure 3-59 illustre le cas où une carte est de taille supérieure à celle du panneau d'affichage.

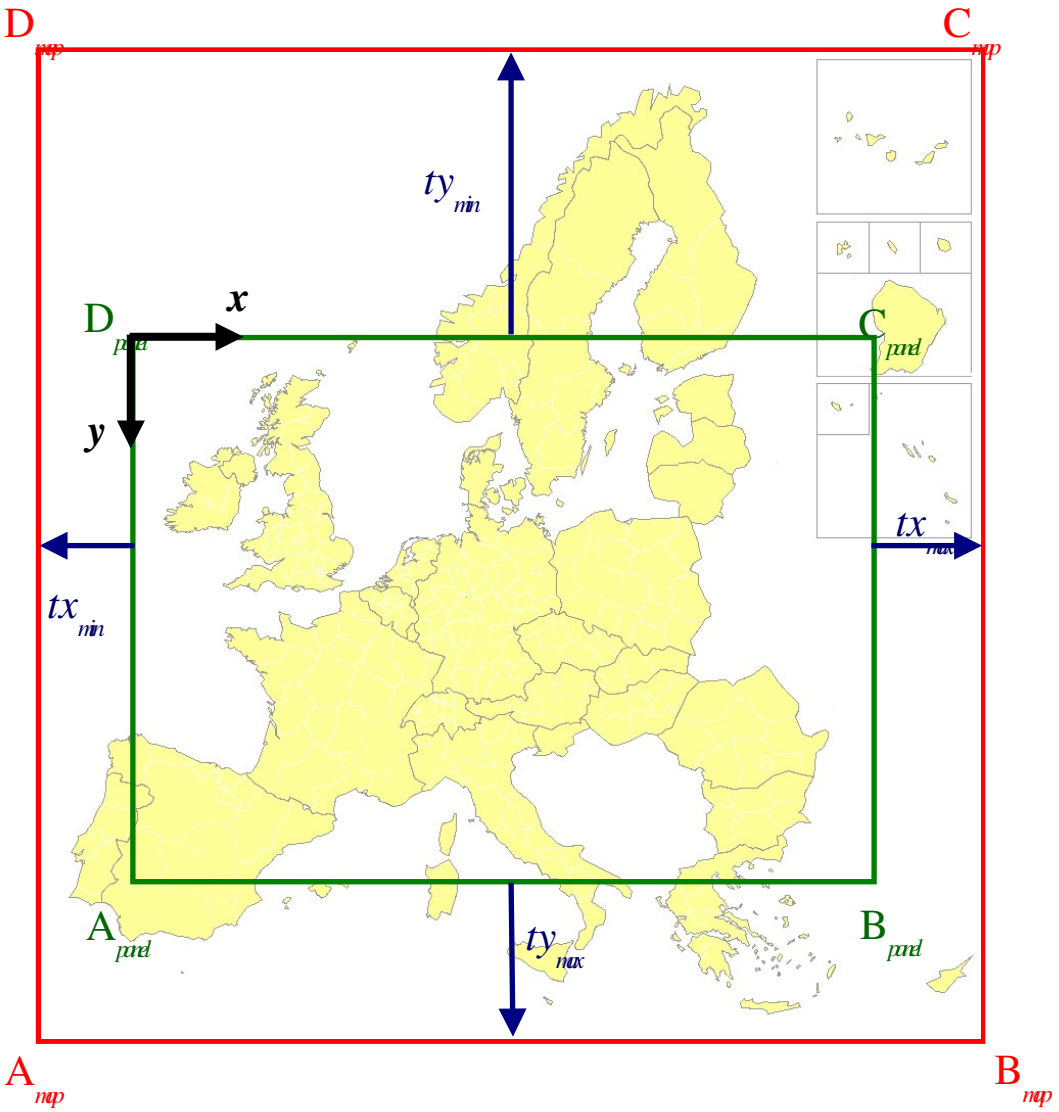


Figure 3-59 : illustration des translations possibles lorsque la carte (rectangle rouge) a une taille supérieure à celle de son conteneur (rectangle vert)

Nous pouvons constater qu'une translation t définie tel que $t = tx \circ ty$ doit respecter les contraintes suivantes :

- $tx_{\min} \leq tx \leq tx_{\max}$
- $ty_{\min} \leq ty \leq ty_{\max}$

3.3.2 Travaux sur les contours des unités

Cette partie présente les améliorations apportées à *HyperAtlas* en termes d'affichage de contours. Ces modifications portent principalement sur trois points : l'affichage du contour des unités utilisées pour le calcul de la déviation moyenne, la personnalisation de la représentation des contours et la mise en valeur de l'unité survolée par la souris.

Affichage des contours des unités territoriales utilisées par la déviation moyenne

Rappelons que le principe de la déviation moyenne est de comparer une unité à une autre qui lui est hiérarchiquement supérieure. Elle consiste, par exemple, à comparer une commune à son département ou à sa région.

Dans ce contexte, il est intéressant de pouvoir visualiser les contours des unités utilisées pour le calcul de cette déviation, ce qui n'est pas le cas dans la version précédente de *HyperAtlas*.

Pour cela, il est nécessaire de récupérer la liste de ces unités lors du tracé de la carte pour pouvoir en dessiner les contours. Le listing présenté par le Code 3-7 détaille la méthode de tracé des contours de ces unités. Dans un premier temps, elle modifie les paramètres des tracés en désactivant l'*antialiasing* grâce à la méthode « `setRenderingHint` » à laquelle l'utilisateur fournit une clé et une valeur, dans ce cas, la clé « `RenderingHints.KEY_ANTIALIASING` » accompagnée de la valeur « `RenderingHints.VALUE_ANTIALIAS_OFF` ». Puis, il invoque la méthode « `drawUnitsBorder` » en lui fournissant en paramètre l'objet « `Graphics2D` » à utiliser pour le dessin des contours, la collection des unités concernées ainsi que la couleur à utiliser. Cette méthode est la méthode générique de tracé des contours d'une collection d'unités territoriales, le détail du code de cette méthode peut être trouvé dans l'2. Enfin, la méthode réactive l'antialiasing. A ce stade nous pouvons noter que la liste des unités territoriales est fournie par le biais d'un objet « `HCUnitRepository` » qui implémente un cache d'unités territoriales. Une description plus précise est fournie dans le chapitre : « 3.4»

```
/**
 * Draw the border of zonings used for medium deviation calculation
 *
 * @param g
 *            the graphics where the border is draw.
 * @param borderColor
 *            color used to draw the border
 */
protected void drawMediumZoningBorders(Graphics2D g, Color borderColor) {

    g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_OFF);

    this.drawUnitsBorder(g, HCUnitRepository.getInstance()
            .getMediumZoningUnits(), borderColor);

    g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
}
```

Code 3-7 : Extrait de la méthode permettant de tracer les contours des unités utilisées comme références pour le calcul des déviations moyennes.

Paramétrage des contours

La version précédente d'*HyperAtlas* affiche deux types de contours différents : ceux des unités territoriales étudiées et ceux des unités de plus haut niveau hiérarchique (ex. les pays

pour l'Europe). Les contours des unités étudiées sont gris et tracés en utilisant l'*antialiasing*, ceux des unités de plus haut rang sont tracés en noir et sans *antialiasing*.
L'ajout des contours des unités de déviation moyenne pose la question de la distinction de ces contours. Une réponse à cette question est l'utilisation de couleurs et traits différents. C'est pourquoi nous nous sommes fixés comme objectif de proposer un moyen à l'utilisateur de personnaliser les contours plutôt que de fixer ces paramètres « en dur » dans l'application comme c'était le cas dans la version précédente d'*HyperAtlas*. Nous avons donc construit le dialogue suivant (Figure 3-60).

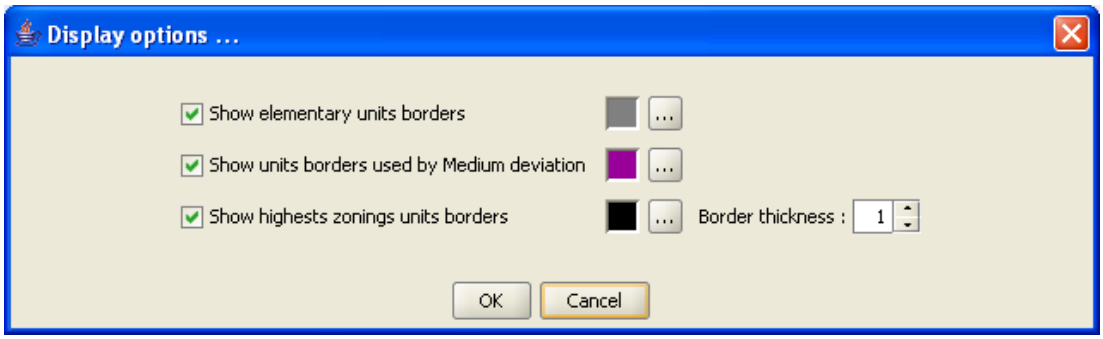


Figure 3-60 : Ecran de personnalisation du contour des unités.

Celui-ci permet à l'utilisateur de préciser s'il souhaite ou non afficher les contours des unités et de choisir avec quelle couleur les dessiner. Pour des raisons de lisibilité, il est également possible de définir l'épaisseur du tracé pour les unités de plus haut rang hiérarchique.

De plus, pour permettre aux utilisateurs de mieux visualiser les limites de l'unité qu'il survole avec le pointeur de la souris nous avons décidé de mettre en valeur celle-ci. Deux solutions sont alors envisageables : griser la couleur de fond de l'unité ou retracer ses contours d'une autre couleur. La première solution a pour avantage de ne traiter que les unités du maillage étudié, mais elle peut gêner la lecture des cartes choroplèthes car celles-ci utilisent les différences de teintes. La seconde solution ne pose pas ce problème mais nécessite, lorsque l'unité est désélectionnée, de restaurer ses contours ainsi que ceux de ses parents représentés sur la carte.
Nous avons choisi la seconde solution en traçant en rouge les contours des unités survolées par la souris. L'2 propose le détail des méthodes « `highlightUnitDisplay` » et « `restoreUnitDisplay` ». Celles-ci permettent respectivement de mettre en valeur et de rétablir les contours de l'unité territoriale passée en paramètre.

3.4 Modifications structurelles

Cette partie présente les modifications apportées à la structure de l'application. Elle concerne notamment son accessibilité via le web, son ouverture à d'autres jeux de données et l'optimisation de l'affichage de cartes grâce à la représentation des données par couches.

3.4.1 *HyperAtlas* accessible via le Web

L'un des objectifs du projet *HyperCarte* est la production d'outils d'analyse spatiale accessibles via le Web. Or, la version précédente d'*HyperAtlas* s'exécute sous la forme d'une application JAVA. Pour rendre l'application disponible via le Web il était donc nécessaire de la proposer en téléchargement sur un site. Il nous a donc paru judicieux d'étudier la possibilité de transformer cette application en *applet*. Pour cela nous nous sommes intéressés aux fonctionnalités que propose une *applet*. Une *applet* est un programme exécuté par un poste client et délivré à celui-ci via un réseau et plus particulièrement via le Web. Si au début de leur existence, les débits offerts par les réseaux limitaient leur taille, et donc les fonctionnalités qu'elles pouvaient proposer, ceci n'est plus vrai aujourd'hui avec la généralisation de l'Internet haut débit. Il est désormais possible de réaliser et de distribuer de véritables applications par ce biais rendant ainsi les qualificatifs « d'application jetable », « de micro programme » ou « d'appliquette » inappropriés. Une *applet* s'intègre parfaitement à une page Web tout comme une image ou un hyperlien. Lorsqu'un utilisateur charge une page Web embarquant une *applet*, dans son navigateur Internet, l'archive Java (JAR) - contenant le corps du programme - est téléchargée sur le poste du client puis exécuté par la machine virtuelle Java (JVM : Java Virtual Machine) de son poste.

Nous avons également constaté que depuis la version 1.2 de l'API, il était possible d'utiliser les mêmes composants d'interface (swing) que ceux utilisés dans une application JAVA classique. Ceci nous a confirmé qu'il était possible de pouvoir réutiliser l'interface utilisateur sans devoir apporter de modifications majeures à celle-ci. Nous bénéficions ainsi des travaux réalisés sur la version précédente de l'application.
Cependant, les *applets* restreignent l'accès à certaines fonctions proposées par le langage. Ainsi, pour des raisons de sécurité, les *applets* ne peuvent normalement pas accéder au système de fichiers du poste qui les exécute. Elles ne peuvent pas non plus ouvrir une connexion réseau à destination d'une machine autre que le serveur Web à partir duquel elles ont été téléchargées, rendant impossible l'ouverture de jeux de données via des URL pointant sur d'autres serveurs.
S'il est facile de comprendre l'intérêt de ces mécanismes de sécurité, ils n'en demeurent pas moins contraignants pour notre application. Cependant, SUN propose deux solutions permettant de lever ces restrictions. La première consiste à paramétrer la machine virtuelle du client pour qu'elle autorise l'accès à certaines ressources telles que des répertoires, des fichiers ou la connexion vers certaines URL. La seconde est basée sur l'utilisation des signatures électroniques : toute personne souhaitant permettre à son *applet* d'accéder à des ressources d'accès normalement restreint peut signer son *applet* à l'aide d'un certificat numérique. Si le concepteur de l'*applet* ou son organisation ne dispose pas d'un certificat numérique, il peut en générer grâce à un outil fourni avec le kit de développement Java (JDK). Notons que dans ce cas le certificat généré n'est pas validé par une autorité certificative ; de plus amples renseignements peuvent être trouvés dans les annexes de ce document.

Lors du lancement de l'*applet*, l'utilisateur est notifié que cette *applet* est signée. Il peut dès lors choisir de faire confiance à l'émetteur du certificat et accorder l'accès à l'intégralité de ces ressources au programme comme s'il s'agissait d'une application ordinaire, ou il peut refuser la signature et continuer à restreindre le comportement de l'*applet*.
Dans notre cas nous avons choisi la seconde solution car il nous paraissait contraignant de demander aux utilisateurs d'éditer les fichiers de configuration de leur JVM pour accorder des droits à notre *applet*. Nous avons donc généré un certificat que nous avons utilisé pour signer l'archive (Jar) de notre programme. Cette signature est générée à l'aide de l'utilitaire « `jarsigner` » fourni avec le JDK.
Après avoir constaté qu'il était possible de rendre notre application exécutable sous la forme d'une *applet*, nous avons réalisé les modifications permettant à celle-ci de s'exécuter sous les deux formes. Cela est possible car l'application et l'*applet* n'utilisent pas le même point d'entrée même si le reste du code est commun au deux. Dans ce but nous avons adapté le code commun pour tenir compte des nouvelles contraintes posées par l'utilisation d'une *applet* signée. Lors du lancement de l'*applet* nous testons quels sont les droits qui lui sont accordé pour ensuite adapter son comportement. Ce test est réalisé grâce à l'objet « `SecurityManager` » ou gestionnaire de sécurité. Un exemple de test est présenté par le Code 3-8.

```
try {
    this.securityMngr = System.getSecurityManager();

        if (this.securityMngr != null)
            this.securityMngr.checkPermission(new AllPermission());

    // We must create the FileChooser after setting up the Look'n
    // Feel
    // if we want it use the selected Look'n Feel.
    File chooserDir = new File("." + File.separator + "dataset");
    if (chooserDir.exists() && chooserDir.isDirectory()) {
        this.fileChooser = new HCFileChooser(chooserDir);
    } else {
        this.fileChooser = new HCFileChooser();
    }

    Settings.getInstance().setSecurityRestricted(false);

} catch (SecurityException secEx) {
    Settings.getInstance().setSecurityRestricted(true);
}
```

Code 3-8 : Test des permissions accordées à l'*applet* extrait du code de la classe HyperAtlas.

Dans un premier temps, on tente de récupérer le gestionnaire de sécurité « `this.securityMngr = System.getSecurityManager();` ». Puis, si celui-ci existe, nous testons les permissions accordées à l'*applet* « `this.securityMngr.checkPermission(new AllPermission());` ». A ce stade, l'envoi d'une exception de type « `SecurityException` » nous indique que les possibilités de l'*applet* sont limitées pour des raisons de sécurité. En fonction de ce test nous pouvons finalement positionner un paramètre d'environnement présent dans la classe « Settings » pour ensuite adapter l'exécution en fonction du contexte de sécurité. Par exemple, nous masquons les menus permettant d'ouvrir d'autres jeux de données ou celui permettant la génération du rapport. Nous nous assurons ainsi qu'aucun message d'erreur ne viendra perturber l'utilisation de l'outil.

L'adaptation d'*HyperAtlas* au lancement sous forme d'*applet* a également soulevé d'autres problèmes tels que la méthode d'accès aux données. Si, dans les versions précédentes du logiciel, celui-ci lisait le contenu d'un fichier nommé « hypercarte.serial.hyp » situé à la ra-

cine du répertoire d'installation de l'application, ce mode d'accès n'est pas adapté aux *applets* pour deux raisons essentielles : les *applets* interdisent les accès au disque dur de l'hôte qui les exécute et, même dans le cas d'*applets* signées, il est trop contraignant pour un utilisateur de devoir télécharger sur son poste de travail le jeu de données pour l'ouvrir par la suite via l'*applet*. Partant de ce constat deux solutions sont alors possibles : rendre le fichier d'objets sérialisés téléchargeable sur le même serveur que l'*applet* ou inclure celui-ci dans l'archive de l'*applet*. Parmi ces solutions nous avons choisi d'inclure le fichier dans l'archive JAR comme les autres ressources nécessaires au programme tel que les icônes. Nous avons effectivement constaté qu'ainsi ce fichier bénéficie de la compression apportées par l'archive. A titre d'exemple, le fichier des données européenne au niveau NUTS3 occupe 3 Mo d'espace disque alors que le jar contenant le programme et ce fichier n'occupe que 1,8 Mo. Ces améliorations sont présentées plus en détails dans la partie suivante de ce document.

3.4.2 Amélioration de l'architecture logicielle d'accès de données

Cette partie a pour but de décrire les travaux réalisés sur la couche de données de *HyperAtlas.* Ces modifications concernent aussi bien l'exploitation de nouveaux jeux de données que l'architecture logicielle d'accès à celles-ci. Ainsi, outre les modifications visant à permettre l'utilisation de nouveaux jeux de données, nous avons voulu permettre à notre application d'être indépendante vis-à-vis des moyens d'accès aux données. C'est pourquoi nous présentons dans un premier temps les travaux visant à permettre l'exploitation de divers jeux de données, pour ensuite expliquer comment nous avons découplé l'application des données. Enfin, nous détaillons les mécanismes mis en place pour améliorer les performances de l'accès aux données.

L'exploitation de divers jeu de données

La première étape que nous nous sommes fixés en termes de gestion des données a été de permettre à *HyperAtlas* d'exploiter d'autres jeux de données que celui de l'Europe. Effectivement, avant ce travail, il était nécessaire d'avoir une version du logiciel par jeu de données, ce qui freinait énormément la distribution de notre outil.

Le cœur du problème réside dans le fait qu'*HyperAtlas* ne contraint pas à l'utilisation d'un système de coordonnées spécifique pour la définition des coordonnées des contours des unités territoriales. Les coordonnées géographiques peuvent donc être exprimées dans n'importe quel système de coordonnées projeté (Lambert II, …) ou non (coordonnées arbitraires). La complexité réside donc à adapter le positionnement et la taille de la carte à la taille de l'écran (voir Figure 3-55). Avant d'approfondir la méthode que nous avons mise en place pour adapter dynamiquement la taille et la position de la carte à celle de l'écran, nous rappelons la manière dont l'application procédait pour afficher la carte de l'Europe à l'écran.

La première chose à noter est que les coordonnées des polygones sont les coordonnées géographiques de ceux-ci est non les coordonnées écran. Ce changement de repère est donc réalisé au moment de l'affichage de la carte grâce à des primitives Java permettant ces changements de repères. Pour des raisons de lisibilité, toutes ces transformations ont été regroupées dans une seule méthode nommée « `setGraphics` » de la classe « `AbstractMap` » (*cf.* Figure 3-51) dont le code est présenté infra.

```
protected Graphics2D setGraphics(Graphics graphics) {

    Graphics2D g = (Graphics2D) graphics;

    g.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
            RenderingHints.VALUE_ALPHA_INTERPOLATION_SPEED);
    g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
    g.setRenderingHint(RenderingHints.KEY_COLOR_RENDERING,
            RenderingHints.VALUE_COLOR_RENDER_SPEED);
    g.setRenderingHint(RenderingHints.KEY_DITHERING,
            RenderingHints.VALUE_DITHER_DISABLE);
    g.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
            RenderingHints.VALUE_TEXT_ANTIALIAS_OFF);

    g.translate(240 + this.panX, 220 + this.panY);
    g.rotate(-Math.PI / 2);
    g.scale(this.scale * 0.92, this.scale * 0.92);

    g.setStroke(new BasicStroke(this.settings.getThickness()));

    return g;
}
```

Code 3-9 : Détail de la méthode setGraphics (Graphics graphics).

Cette méthode commence par définir des paramètres de représentation graphique tel que l'utilisation ou non de l'antialiasing grâce à la méthode « `setRenderingHint` » de l'objet « `graphics` » passé en paramètre.
Le changement de repère est opéré ensuite par la méthode « `translate(x,y)` » qui effectue une translation de vecteur $\overrightarrow{T(x, y)}$. Puis on opère une rotation d'angle $\frac{-\pi}{2}$ grâce à l'utilisation de la méthode « `rotate(double angle)` ». Enfin la méthode « `setScale (x, y)` » réalise la mise à l'échelle (homothétie car, dans ce cas, x est égal à y).
Notons également qu'après avoir réalisé le changement de repère, nous adaptons l'épaisseur du tracé grâce à la méthode « `setStroke(float thickness)` ». Sans cette dernière étape, les traits seraient invisibles car leur épaisseur serait proche de zéro.
Comme expliqué précédemment, cette méthode utilise des valeurs statiques pour les paramètres de translation et de mise à l'échelle ce qui pose les problèmes d'adaptation à de nouveaux jeux de données.
Pour résoudre ces problèmes, nous avons développé des méthodes calculant les paramètres adéquats à l'aide du rectangle englobant de la carte et des coordonnées du panneau dans lequel la carte est dessinée. Ainsi, avant le premier affichage d'une carte, une méthode nommée « `initMap` » est appelée pour effectuer le changement de repère adéquat. Pour effectuer le changement de repère, elle se base sur trois étapes (*cf.* Code 3-10). La première consiste à récupérer le rectangle englobant de la carte. Puis elle détermine les coordonnées de l'écran dans le repère de la carte pour positionner les valeurs de la translation, enfin elle calcule l'*offset* de zoom nécessaire à la mise à l'échelle de la carte.

```
/**
 * Get map and display border to compute a zoom offset enabling to
 * scale the map to fit it in its display panel.
 */
protected void initMap() {

    this.initMapBorder();
    this.recalculateDisplayBorder();
    this.calculateZoomOffset();
}
```

Code 3-10 : Méthode initMap extraite de la classe AbstractMap.

Le détail des méthodes « initMapBorder », « recalculateDisplayBorder » et « calculateZoomOffset » est disponible dans les annexes de ce mémoire.

La réduction du couplage entre la couche des données et le reste de l'application

Après avoir permis à l'application de charger tout type de jeu de données, nous avons rencontré le problème posé par la volumétrie de celles-ci. Effectivement, dans son fonctionnement *HyperAtlas* repose sur des données chargées dans la mémoire vive. Or comme nous l'avons vu, la taille des données est proportionnelle au nombre d'entités gérées par le système ainsi qu'à leur complexité. Aussi, parvenons-nous très rapidement aux limites de la capacité mémoire. Il est donc nécessaire de proposer d'autres méthodes de gestion des données. La Figure 3-61 présente un diagramme de classes simplifié de la partie « données » de la version précédente d'*HyperAtlas*. Lors de l'ouverture d'un fichier d'objets sérialisés, seuls trois objets sont lus : « `Unit` », « `Zoning` » et « `Area` ». Effectivement la classe « `Unit` » embarque une table de hachage (`HashTable`) statique contenant l'ensemble des objets « `Unit` » du système. Cette table utilise les codes textuels des unités comme index et permet ainsi de retrouver une unité grâce à un identifiant textuel. Les deux autres objets (« `Zoning` » et « `Area` ») fonctionnent sur un principe similaire. Lorsqu'un objet d'une autre couche doit accéder à ces données, il fait d'abord appel aux méthodes statiques des classes pour récupérer l'instance qui l'intéresse.

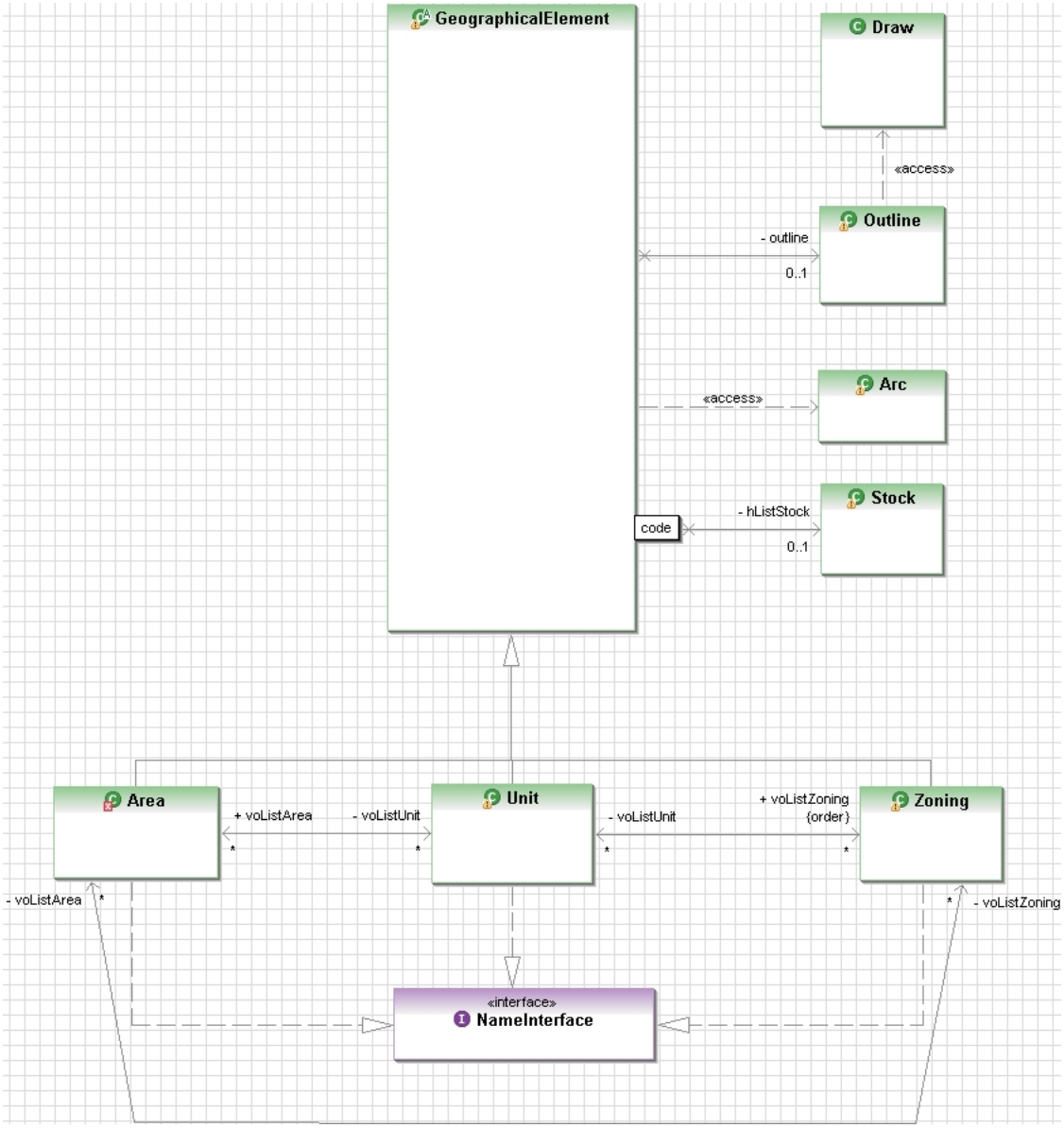


Figure 3-61 : Diagramme de classes de la partie donnée de la version 1.0.0.c de HyperAtlas.

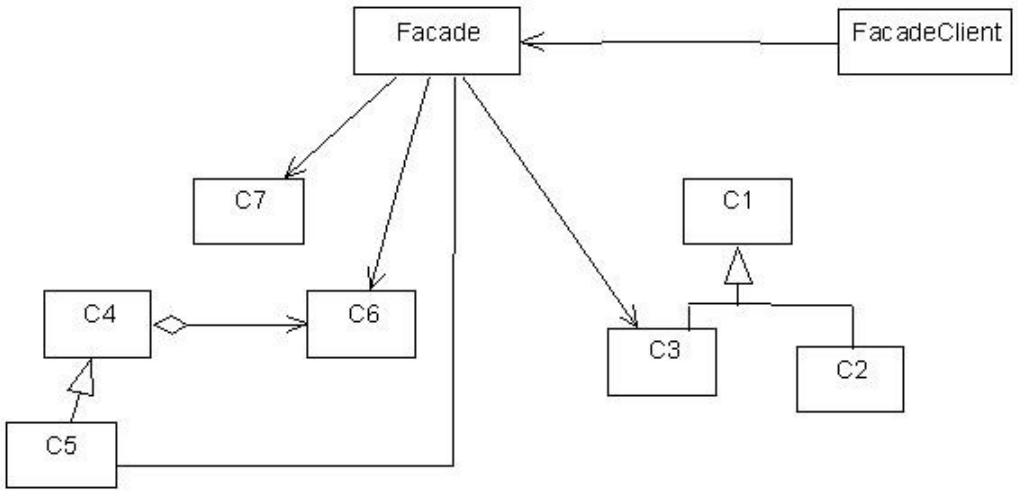
Ce mode de fonctionnement pose deux problèmes : l'intégralité des données du jeu doit être chargée en mémoire et toute évolution est rendue difficile car le couplage entre la couche des données et les autres couches est fort.

Pour parvenir à nos fins et réduire ce couplage, nous avons donc mis en place une façade dont le rôle est de masquer la complexité d'un sous-ensemble de données. Cette façade est issue du patron façade proposé par M. Gamma [Gamma et al., 99]. L'idée sous-jacente de ce patron de conception est de proposée une classe offrant l'ensemble des services d'une collection de classes. De la sorte, les autres éléments d'une architecture logicielle passe par cette façade et ignore la complexité du sous système qu'elle couvre. Avant de détailler l'implémentation de ce patron, ce document propose de faire un bref rappel à son égard avant de détailler la manière dont il a été implémenté dans l'application.

Description

Lorsque l'on désire faciliter l'utilisation d'un sous-système, ce patron permet de fournir une interface simplifiée qui évite le couplage direct entre les clients et les éléments du sous-système.

Structure et constituants



« `Facade` » définit une interface unifiée pour le sous-système et donne accès à certains composants publics,
C1 à C7 implantent des fonctions du sous-système,
« `FacadeClient` » manipule le sous-système en passant par la façade.

Forces et faiblesses

+ diminue le couplage entre le client et le sous-système,

+ masque des éléments privés du sous-système,

- l'interface unifiée présentée par la façade peut être trop restrictive pour utiliser l'ensemble des fonctions du sous-système.

Ainsi, nous avons créé une classe nommée : « `HCFileInput` » servant de façade au sous ensemble des classes représentant les données (`Area`, `Zoning`, `Unit`, …). De plus, pour pouvoir implémenter efficacement un accès à d'autres sources de données nous avons extrait une interface de cette classe, de telle sorte que tous les clients utilisent cette interface. L'ajout d'un nouveau support se fait simplement en implémentant cette interface et ne demande donc pas de modifications dans le reste de l'architecture de l'application. Nous avons pu implémenter ainsi, en plus de la lecture d'un fichier sur le disque du poste client, la lecture d'un fichier inclus dans l'archive de l'application, l'accès à un fichier via une URL et même la lecture des données via une base de données (*cf.* Figure 3-62).

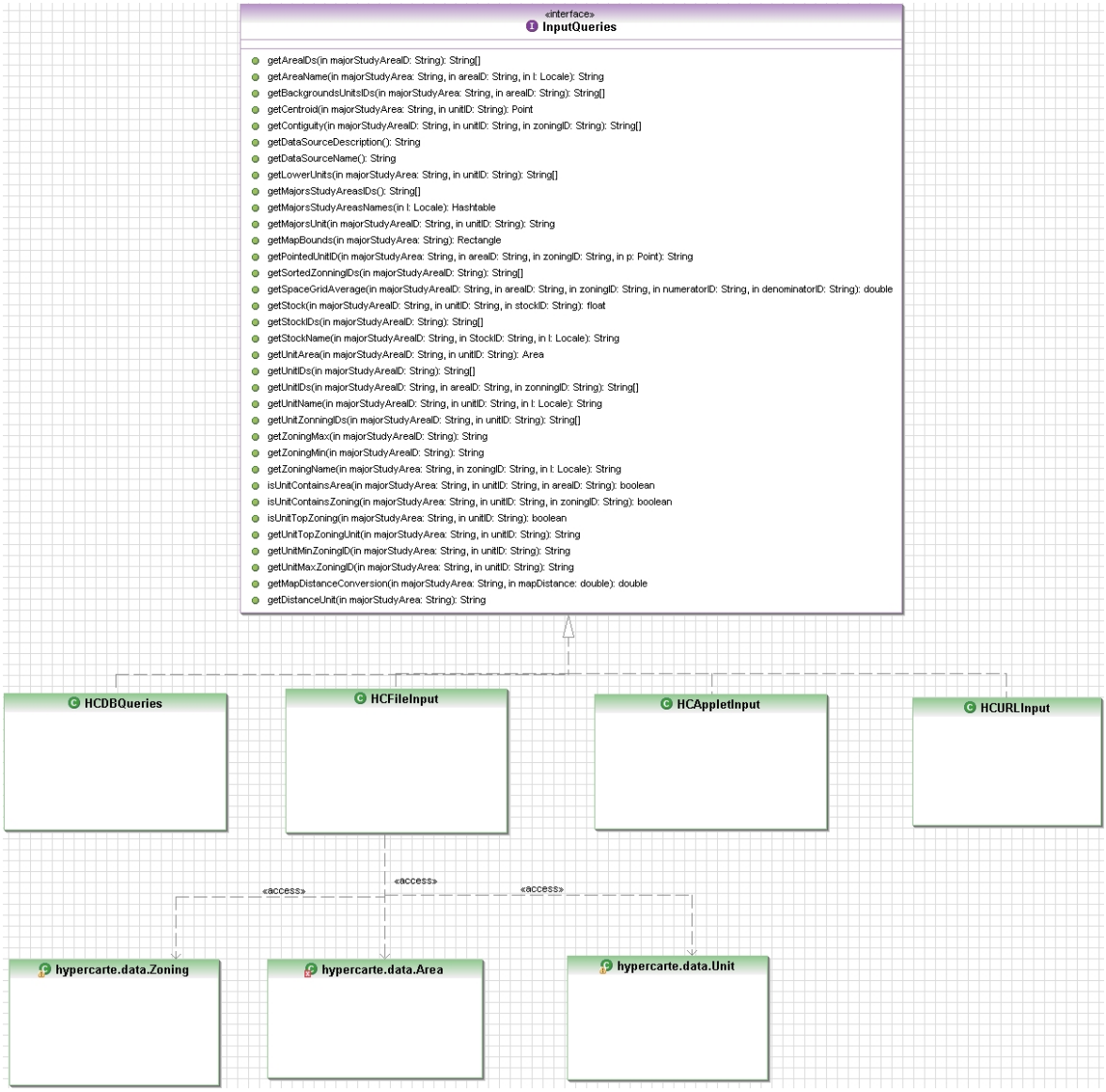


Figure 3-62 : Architecture d'accès aux de données de l'application *HyperAtlas*. Toutes les façades permettant l'accès aux données implémentent l'interface « `InputQueries` ».

Amélioration des performances de l'accès aux données

Même si la façade présentée dans la partie précédente permet à l'application d'aller chercher les informations nécessaires directement dans une base de données, la lecture systématique de toute information dans une source externe à l'application augmente rapidement les temps d'accès. C'est la raison pour laquelle nous avons mis en place un mécanisme de cache dont l'objectif est de stocker en mémoire les informations les plus utilisées. Ainsi *HyperAtlas* n'accède aux sources externes que lorsqu'il n'a pas l'information souhaitée d'où un gain de temps important. Pour réaliser ce mécanisme de cache nous avons dû créer des structures de données simplifiées. En effet, la structure de donnée existante est beaucoup trop lourde car elle embarque toutes les informations, y compris celles qui ne sont pas utilisées dans le contexte d'étude choisi, tel que l'ensemble de tous les contours des unités, l'intégralité des stocks, etc.

Le cache doit donc contenir des objets utilisés par les couches logique et application. A ce titre, ces objets doivent donc contenir les informations utiles pour ces couches comme par exemple, des objets géométriques « `java.awt.Area` » pour la couche présentation ou les valeurs nécessaires aux calculs réalisés dans la couche logique.

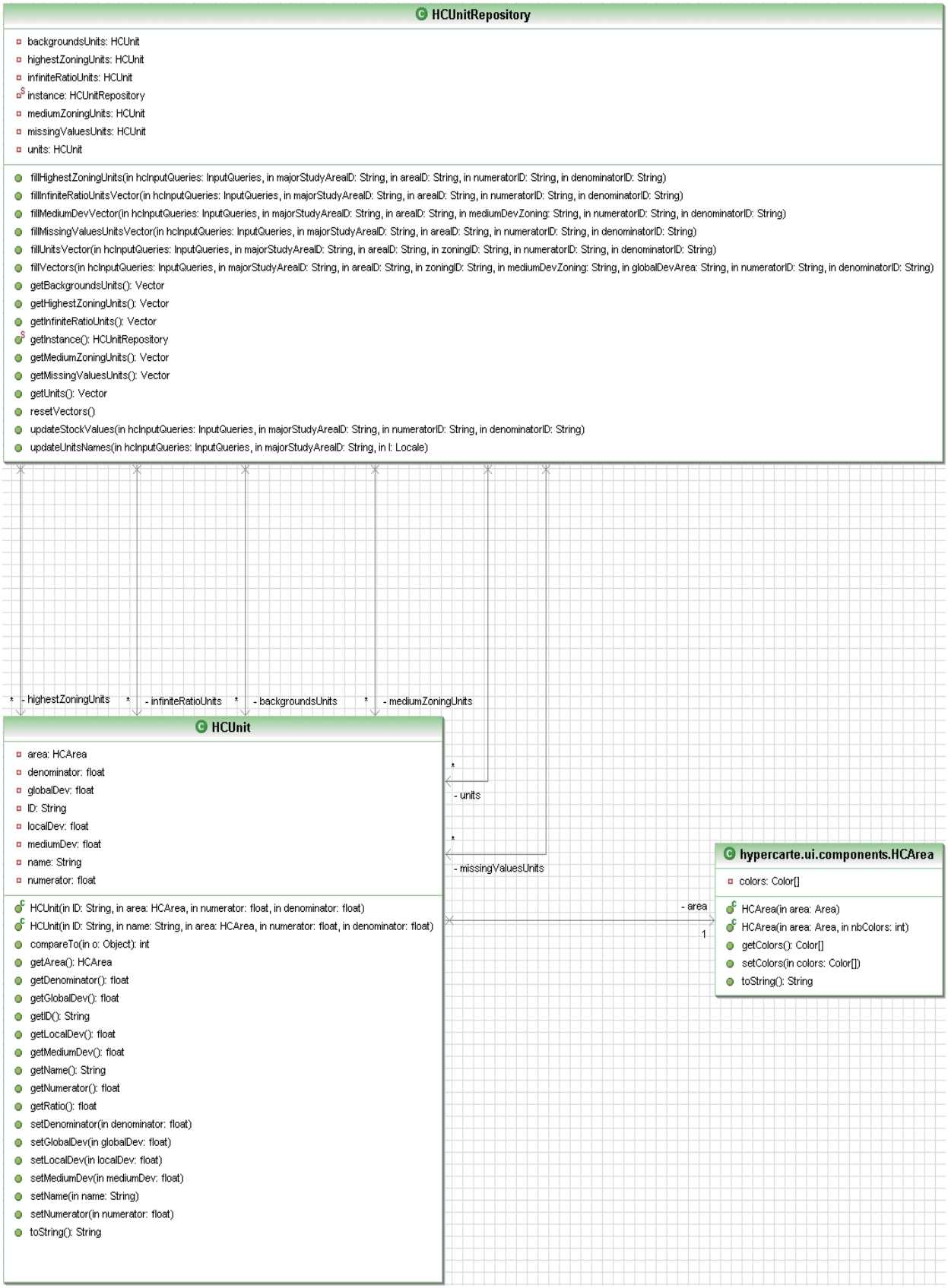


Figure 3-63 : Diagramme de classes présentant les classes utilisées pour réaliser un cache mémoire des données.

La Figure 3-63 illustre les classes implémentées pour la mise en cache de données, parmi elles on distingue :

- les classes « HCUnit » et « HCArea » qui fournissent une représentation simplifiée des unités territoriales et de leur représentation géométrique,
- la classe « HCUnitRepository » qui constitue ce *buffer*.

La classe « HCUnit » contient une chaîne de caractères représentant le code de l'unité, des nombre à virgule flottante (*float*) représentant les valeurs du numérateur, du dénominateur ainsi que les valeurs de références utilisées par les trois déviations : locale, moyenne et globale. Cette classe est également composée d'un objet de type « HCArea » qui est une spécialisation (héritage) de l'objet « java.awt.Area » contenant les informations utiles à la représentation graphique de l'unité, telles que les contours et les couleurs de fond à utiliser en fonction des cartes à afficher.
La classe « HCUnitRepository » implémente le patron « Singleton » pour n'avoir qu'une seule instance dans le système. Cette instance contient plusieurs collections d'objets « HCUnit ». Celles-ci contiennent soit des unités à étudier (unités contenues dans le contexte d'étude, les unités nécessaires à l'étude des déviations moyennes), soit des unités seulement utilisées pour la représentation graphique (unités décoratives utilisées pour la représentation du fond de carte, unités de plus haut rang hiérarchique...).

Lorsque l'utilisateur change un paramètre du contexte d'étude (aire d'étude, maillage), *HyperAtlas* fait appel aux méthodes de cette classe pour vider et remplir les collections affectées par ce changement.
Si l'utilisateur change d'autres paramètres tels que le numérateur ou le dénominateur, *HyperAtlas* parcourt la collection des unités simplifiées pour mettre à jour la valeur de leur champ numérateur ou dénominateur.

Une fois ces opérations terminées, la couche logique intervient pour effectuer les calculs nécessaires. Pour cela, elle parcourt les collections contenues dans l'instance du « HCUnitRepository », effectue les calculs et stocke les résultats dans les objets « HCUnit » concernés.

Enfin, la couche représentation prend le relais en parcourant toutes les collections contenues dans ce cache pour dessiner les unités à l'écran. Pour ce faire, elle récupère l'objet « HCArea » contenu dans chaque « HCUnit ». Grâce à lui, elle peut dessiner les contours des unités et connaître la couleur à utiliser pour remplir la forme ainsi obtenue.

En conclusion, ce cache contenant des représentations simplifiées des objets de la couche donnée, permet de ne pas accéder en permanence à un support beaucoup plus lent que la mémoire, comme une base de données. Il offre pour avantage de ne charger en mémoire que le strict nécessaire, contrairement à la structure d'objets utilisés dans la couche donnée. L'utilisation de ce mécanisme permet également de mieux découper les traitement : on récupère d'abord les données, puis on les traite pour finalement les afficher.
Cependant, son inconvénient principal réside dans le fait qu'il crée des doublons d'informations lorsqu'il est utilisé avec le système historique de chargement de l'intégralité des données en mémoire. En contrepartie, il évite également de réitérer pour des raisons de mise à jour de l'affichage, des calculs déjà effectués.

3.4.3 Représentation en couches des cartes

Lorsque nous avons étudié la problématique du passage à l'échelle pour les jeux de données, nous avons réalisé divers tests de performance. Or nous nous sommes aperçus que le temps de traçage des cartes n'est pas aussi négligeable que nous le pensions. Ainsi, nous avons constaté que de systématiquement redessiner intégralement la carte pour la moindre modification pose rapidement des problèmes de performance, lorsque l'on traite un grand nombre d'objets graphiques complexes. Ceci est d'autant plus vrai que pour certaines modifications demandées par l'utilisateur il n'est pas nécessaire de redessiner tous les objets.
C'est en observant la manière dont les cartes sont affichées que nous avons remarqué que les entités graphiques étaient tracées dans un ordre bien précis :

1. les unités de décorations utilisées pour ajouter de la lisibilité à la carte,
2. les polygones plein des unités du contexte d'études,
3. les contours de ces polygones,
4. si nécessaire, les contours des unités utilisées par le calcul de la déviation moyenne,
5. enfin, les contours des unités de plus haut niveau hiérarchique.

Or, Sun propose dans son « JAVA Tutorial » une méthode permettant d'accélérer les tracés répétitifs d'objets complexes. Cette technique nommée « double buffering » consiste à utiliser une image pour effectuer des tracés lorsque que cela est nécessaire. On met à jour cette image en y dessinant des objets graphiques, puis on la peint dans son panneau de destination via la méthode « `repaint` ». Cette méthode rafraîchit juste l'image affichée, sans modifier son contenu. Imaginons que cette image soit composée de plusieurs milliers d'objets graphiques complexes on ne redessine qu'une image tampon et non les milliers d'objets qui la composent, d'où un gain de temps important lors du réaffichage.

Nous avons donc créé un objet nommé « Layer » dont le but est de fournir une représentation de ces couches de représentation. La Figure 3-64 présente le détail de cette classe. Nous notons qu'elle contient un attribut de type « BufferedImage » utilisé comme image tampon pour le dessin des unités. C'est dans cette image que seront effectués tous les tracés de la couche et c'est finalement cette image qui sera affichée comme représentation d'une couche. Une chaîne de caractères permet d'attribuer une description de la couche. Deux booléens permettent de définir un état pour la couche permettant de savoir si elle est affichée et si elle est à jour. Nous pouvons également remarquer la présence d'un attribut de type « Graphics2D » utilisé comme point d'accès pour le dessin dans l'image. Effectivement, JAVA ne permet pas de dessiner directement dans une image, mais il permet d'extraire un objet

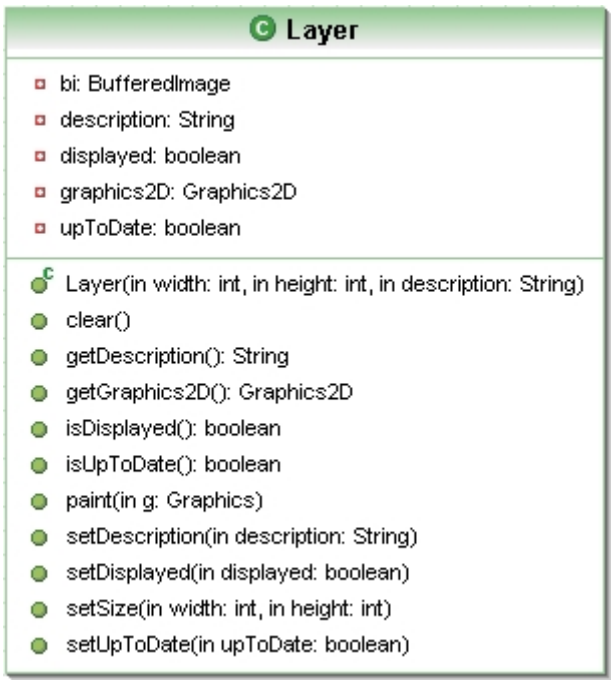


Figure 3-64 : Détail de la classe « Layer ».

de type « Graphics2D » permettant d'écrire dans l'image. Ce fonctionnement est assez similaire à celui des flux (Stream) permettant l'accès à un fichier par exemple.
Parmi les méthodes de cet objet, outre les méthodes d'accès aux attributs (« Getters »), se trouve des méthodes permettant d'effacer le contenu de l'image (« `clear()` »), de définir la taille de l'image et finalement la méthode « `paint(g Graphics)` » qui affiche l'image tampon dans le graphique passé en paramètre à la méthode.

Après avoir implémenter cette classe représentant une couche, nous avons redéfini les classes reflets des cartes de l'application. Un diagramme de classe est proposé en annexe à ce document (partie 1 de l'2). Nous pouvons y remarquer les liens entre la classe « BufferedMap » et les couches. Ainsi, cette classe contient une collection de couches ordonnées selon leur ordre d'affichage, elle contient également certaines couches déclarées de manière statique. Ces couches sont celles communes à toutes les cartes : fond de carte, contours des unités du contexte d'étude, contours des unités utilisées par le calcul de la déviation moyenne, contours des unités de plus haut rang hiérarchique…
Chaque carte surcharge également les méthodes de gestion des événements (*cf.* chapitre 3.2.2), elle peut ainsi adapter son comportement à ces événements et ne modifier que les couches devant l'être. Ensuite, lorsqu'il est nécessaire d'afficher ou de réafficher une carte, l'application parcourt sa liste de « `layers` » pour les afficher. Cette opération est réalisée dans la méthode « `paintComponent` » de la classe « `BufferedMap` ». Ensuite, chaque classe représentant une carte surcharge la méthode « `paintMap` » pour définir les couches qu'elle doit afficher. Le détail de ces méthodes est présenté en annexe à ce document (*cf.* partie 5 de l'2).

3.5 Les nouvelles fonctionnalités

3.5.1 Les nouvelles fonctionnalités pour la représentation

- Menu clic droit
- Zoom sur unité
- Centrage de carte

3.5.2 Edition des seuils utilisés pour les palettes de déviation

Parmi les demandes formulées aux membres du projet *HyperCarte*, une concerne l'édition des seuils utilisés pour les cartes choroplèthes. En effet, jusqu'alors les seuils étaient calculés à l'aide de quantiles et l'utilisateur ne pouvait choisir que le mode de découpage de la plage de valeur (arithmétique ou géométrique). Or, il apparaît intéressant pour un utilisateur de pouvoir définir des seuils personnalisés correspondant à une réalité socio-économique (taux de renouvellement de la population, seuil de pauvreté, …).
Pour cela, nous avons donc décidé de développer une nouvelle interface permettant de proposer l'édition de ces seuils dans les options d'une carte, comme le montre la Figure 3-65.

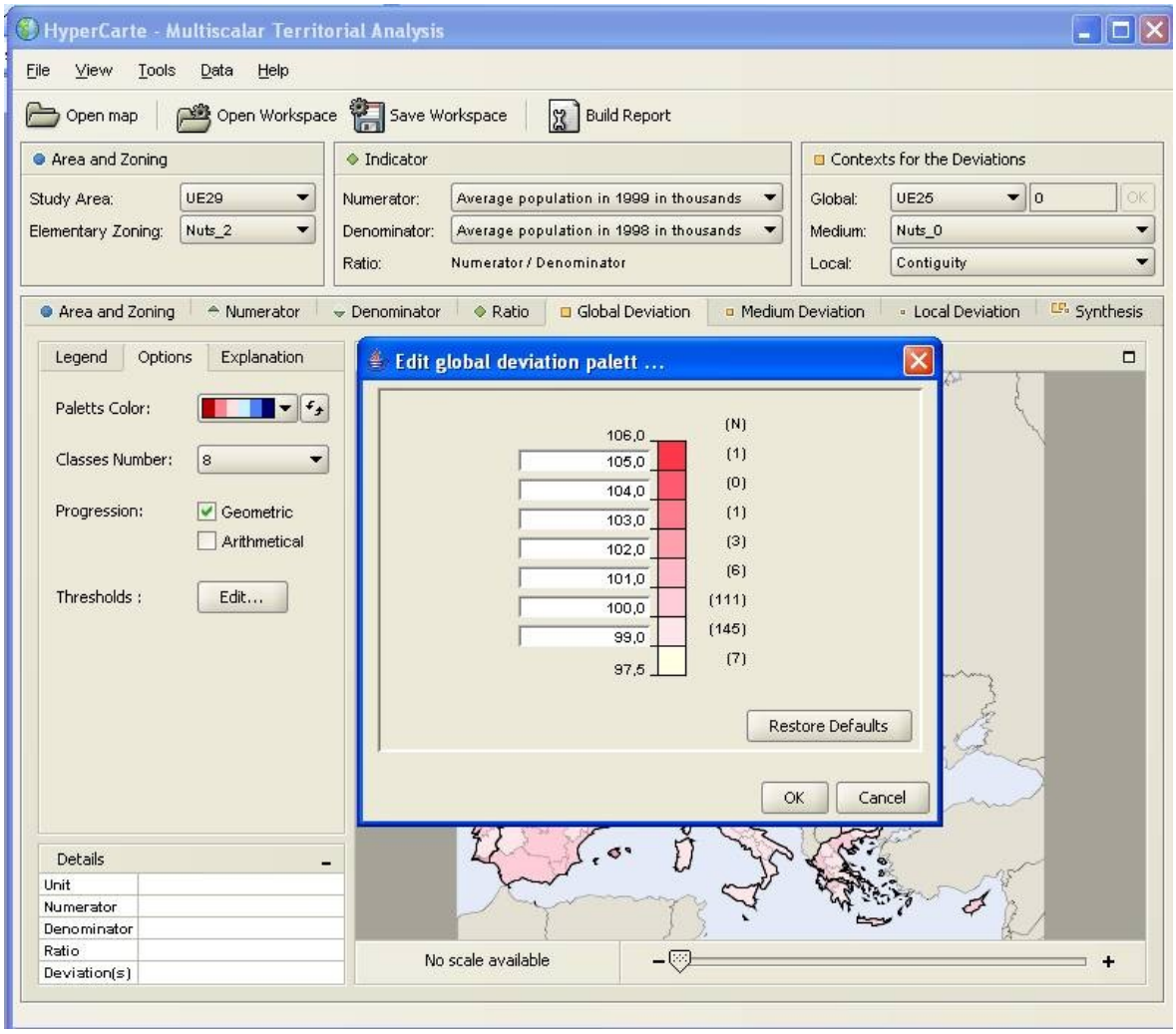


Figure 3-65 : interface de modification des seuils utilisés par les cartes choroplèthes.

Bien entendu nous avons prévu que l'interface que nous proposons puisse suffisamment guider l'utilisateur pour prévenir les éventuelles erreurs de saisie. Ainsi, nous veillons à ce que les plages saisies par l'utilisateur restent cohérentes c'est-à-dire que nous vérifions le non chevauchement de ces plages.
Si lors de la saisie une erreur est détectée l'utilisateur en est tout de suite averti et la saisie posant problème passe en rouge.

3.5.3 Ouvrir une autre carte dynamiquement

Comme nous l'avons vu, nous avons modifié et surtout découplé les données de l'application. Ceci nous a permis de pouvoir ouvrir une carte en cours d'exécution. Il est donc maintenant possible pour un utilisateur d'ouvrir un jeu de données contenu dans un fichier en choisissant « Ouvrir » dans le menu « Fichier » d'*HyperAtlas*. L'utilisateur peut alors parcourir son disque dur pour chercher le fichier qu'il souhaite ouvrir.

Or *HyperAtlas* peut être intégré à une page web sous la forme d'une *applet* Java et qu'une *applet* ne pouvant accéder au disque dur de la machine que sous certaines conditions, nous ne rendons donc ce menu disponible que si le contexte de sécurité le permet.

De plus, outre l'ouverture d'un fichier nous avons souhaité proposer à l'utilisateur d'accéder a un fichier situé à une URL quelconque. C'est pourquoi nous proposons une interface de saisie d'URL pour accéder à un ou plusieurs fichiers hébergés par des serveurs Web.

3.5.4 L'import de données statistiques

Nous nous sommes directement inspirés de la fonction d'import de donnée proposé par le logiciel *GeoClip* pour permettre à un utilisateur d'importer des données statistiques par simple « copier/coller ».

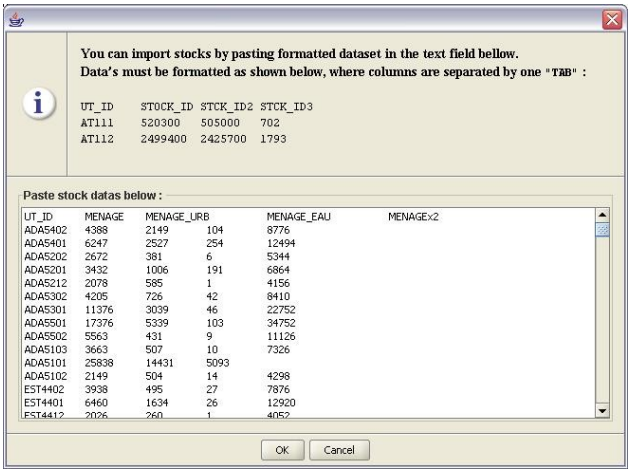


Figure 3-66 : Dialogue d'import des données statistiques.

Pour cela, l'utilisateur peut entrer ses données dans un fichier Excel formaté comme présenté dans 3de ce document. Ce fichier se compose de plusieurs colonnes. La première, intitulée UT_ID contient les codes des unités territoriales concernées, les suivantes se composent d'un code stock et des données statistiques pour le code. Chaque colonne devant être séparée de la suivante par une tabulation.
Notons qu'il n'est pas nécessaire que les données importées soient complètes. En d'autres termes, il est possible d'importer des statistiques ne concernant qu'une partie des unités présentes dans l'aire d'étude. Les unités n'ayant pas de valeur définie pour un stock donné sont colorées en gris.

Après avoir importé ses données l'utilisateur peut, s'il le souhaite, enregistrer un nouveau fichier *HyperAtlas* contenant les données du jeu de données initial enrichi des données importées par l'utilisateur. Notons que, dans le cas où les statistiques importées possèdent

les mêmes codes que les stocks déjà présents dans l'application, les stocks concernés sont mis à jour.

3.5.5 La recomposition des unités territoriales

Comme nous l'avons vu, les unités territoriales s'organisent hiérarchiquement selon la relation unités composantes/unités composées (*cf.* Figure 3-24). Cette relation que nous appelons maillage est à la base de nombreux calculs réalisés dans l'application. Tels que la détermination des contours des unités composées par d'autres unités de rang hiérarchique inférieur. Il en est de même pour les stocks. Par exemple, la population d'une région sera calculée en faisant la somme de la population des départements qui la composent.

Il est donc apparu pertinent pour les membre du projet d'étudier les apports d'une possible recomposition du maillage territorial. Pour cela nous avons modifié l'interface d'*HyperAtlas* pour permettre la modification de la composition d'une unité territoriale.

Cette fonctionnalité est composée d'une suite d'activités présentées dans par le diagramme d'activité suivant (Figure 3-67).

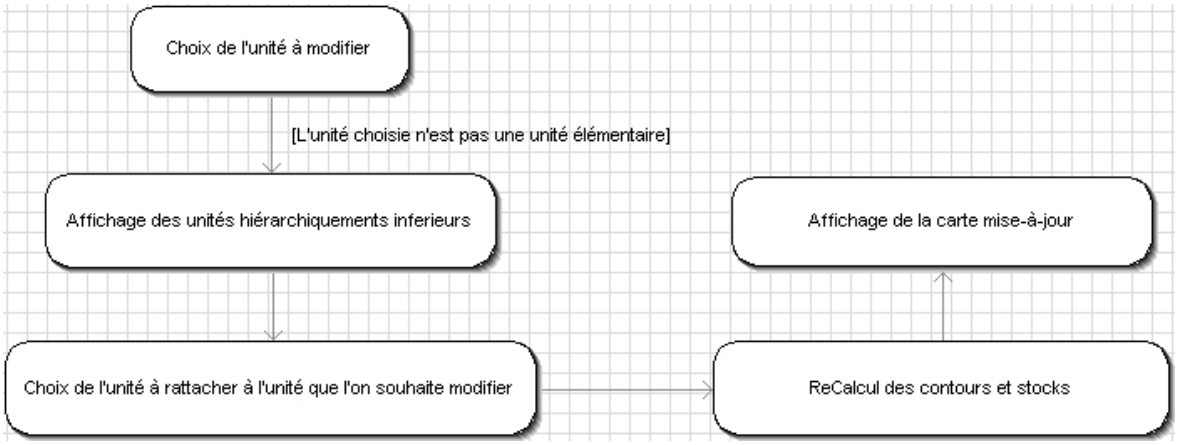


Figure 3-67 : Diagramme d'activité de la fonction de modification de la composition d'une unité territoriale.

Comme nous le montre le diagramme, l'utilisateur choisit une unité territoriale parmi les unités composées. Dés lors *HyperAtlas* affiche les unités de rang inférieur pour permettre à l'utilisateur de choisir l'unité qu'il souhaite faire entrer dans la composition de l'unité choisie en premier lieu. Ce choix effectué, l'outil re-calcul contours et stocks pour les unités concernés, puis affiche la carte modifiée.

3.6 Synthèse

Des évolutions concernant la personnalisation des cartes ont également été réalisées et permettent aux utilisateurs de s'approprier encore davantage cet outil.
Les évolutions souhaitées dans *HyperAtlas* nous ont conduit à revoir une partie de son architecture, afin de rendre l'accès aux données beaucoup plus souple et surtout de rendre l'application indépendante des données et types de données auxquels elle peut accéder.

Cette nouvelle architecture d'accès aux données est devenu la base d'un nouvel outil qui repose sur cette architecture en lui ajoutant de nouveaux services lui permettant la gestion des données. Cet outil, nommé *HyperAdmin*, est présenté dans le chapitre suivant.

CHAPITRE 4

DÉVELOPPEMENT ET ÉTUDE D'HYPERADMIN

4.1 Introduction

Ce chapitre présente des éléments de la méthode employée pour l'analyse, la conception et la réalisation de l'outil *HyperAdmin*. Pour cela, nous présentons la méthode utilisée et l'illustrons par des exemples concrets issus de la conception d'*HyperAdmin*.

Le processus que nous avons choisi s'appelle le 2TUP « 2 Track Unified Process » ou processus en Y. Il est par ailleurs sous-jacent à la structure de ce chapitre.

Nous définissons un processus comme étant un ensemble d'étapes, en partie ordonnées, qui concourent à l'obtention d'un système logiciel ou à l'évolution d'un système existant [ROQ, 2003].
Parmi ces processus, nous distinguons les processus unifiés (Unified Process) qui sont des processus de développement logiciel construit sur UML[17]. Ce sont des processus itératifs et incrémentaux, pilotés par les cas d'utilisation, qui visent à réduire les risques.

Le 2TUP est un processus unifié, apportant une réponse au besoin continuel de changement des systèmes d'information, en préconisant de travailler en parallèle sur deux axes : un axe fonctionnel et un axe technique. A l'issue des évolutions du modèle fonctionnel et de l'architecture technique, la réalisation du système consiste à fusionner les résultats obtenus dans chacune des branches. Le résultat de cette fusion est la production d'un schéma de développement en forme de Y, comme le montre la Figure 4-68.

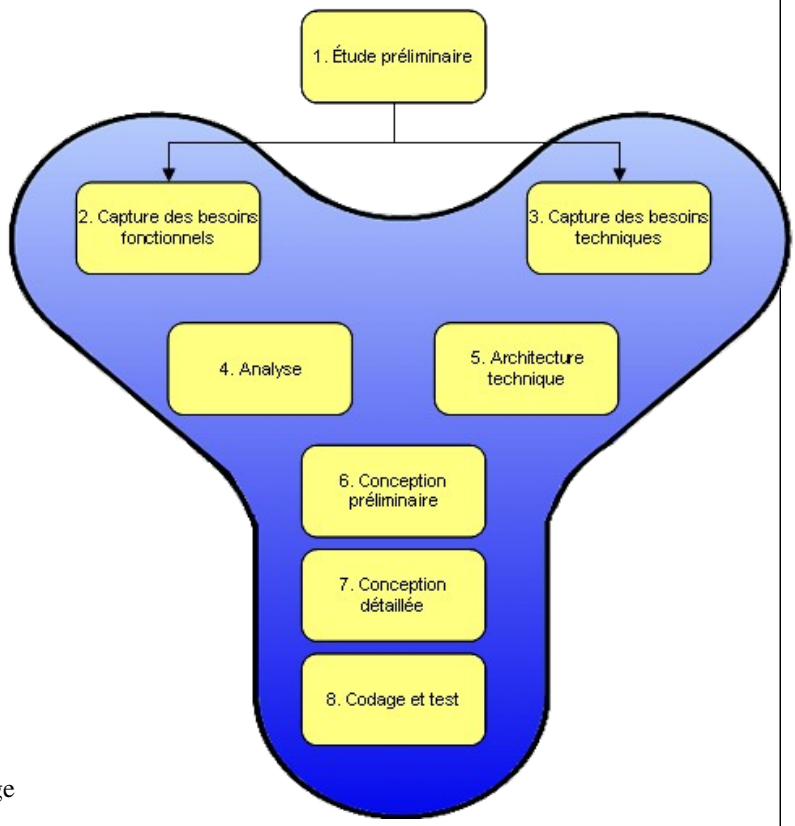


Figure 4-68 : Le processus de développement en Y introduit par 2TUP

Le découpage de ce chapitre suit donc les dif-

[17] UML : Unified Modeling Language

férentes étapes proposées par la méthode. Ainsi, après notre introduction nous commençons la modélisation des besoins par l'étude préliminaire qui présente nos attentes pour *HyperAdmin*.
Nous distinguons ensuite les besoins fonctionnels et les besoins techniques. Ces étapes achevées, nous procédons à l'analyse objet du projet, puis à la définition de l'architecture technique. Les sixièmes et septièmes étapes constituent la conception préliminaire et détaillée. Enfin, nous terminons ce chapitre en présentant au lecteur les réalisations logicielles effectuées suite à ses travaux.

4.2 Etude préliminaire

Cette étude utilise principalement le texte pour effectuer un premier repérage des besoins fonctionnels et opérationnels.

4.2.1 Cahier des charges

Présentation et objectifs

Le projet *HyperAdmin* a vu le jour pour permettre de créer, mettre à jour et exporter, de manière conviviale, des jeux de données pour le module d'analyse territoriale (M.A.T.) du logiciel *HyperAtlas*. Pour cela, il doit permettre, dans un premier temps, d'importer des données (fichiers textes, Excel, MIF/MID), de les structurer, de les stocker dans une base de données et de les réexporter sous la forme d'un fichier d'objets sérialisés utilisable par *HyperAtlas*. Il simplifiera ainsi le processus actuel de création de jeux de données qui nécessite le lancement successif de plusieurs scripts et programmes convertissant des données textuelles en un fichier d'objets sérialisés

Dans un deuxième temps, il permettra la création de nouveaux jeux de données à partir de celles contenues dans la base de données. Il permettra également de modifier ou enrichir les jeux de données présents dans la base. Une gestion thématique des indicateurs statistiques est également à étudier pour faciliter l'usage de *HyperAtlas*. Un utilisateur se verra alors proposer une liste de ratios pertinents organisés en thèmes et sous thèmes. Par exemple, un utilisateur pourra choisir d'étudier les « taux de natalité en 1999 » dans le thème " démographie ", au lieu de choisir lui-même le nombre de naissances en 1999 à diviser par la population totale en 1999.

Dans un troisième temps, il devra permettre de gérer des comptes utilisateurs pour la mise en place de droits d'accès aux données. Chaque utilisateur se verra attribuer des droits sur les divers projets et sur la base de données en général. Ces droits seront les suivants : visualisation (lecture) d'un projet, modification d'un projet, création d'un nouveau projet, suppression d'un projet. Il doit être possible de positionner des droits d'accès pour un groupe d'utilisateur ou individuellement pour chaque utilisateur.
Nous définissons la notion de droits effectifs comme étant l'ensemble des droits que possède un utilisateur. Ces droits peuvent lui avoir été attribués directement ou bien provenir de ses appartenances à des groupes d'utilisateurs.
La gestion de ces droits implique de faire la distinction entre deux types de profil : un profil « administrateur » qui aura un accès total à toutes les données et qui permettra de définir les droits d'accès des utilisateurs, et un profil utilisateur permettant le travail sur les informations contenues dans le système en accord avec les droits qui lui sont accordés.

Enfin, il permettra l'ajout de « méta-données » sur les données renseignant sur leur provenance, leur auteur, leur année, leur origine … Ces méta-données faciliteront notamment les recherches d'informations et offrent la perspective, d'utilisation d'ontologies.

Structure des données à représenter

Etant donnée la nature du projet, « création d'un outil de gestion de données », il est important d'énumérer et de décrire les données que nous manipulons par la suite. Ces données sont les suivantes :

Des projets (Projects) : un projet représente un ensemble d'unités territoriales, de stocks et d'aires d'étude.

Des maillages territoriaux (Zonings) : ils correspondent à un découpage, une partition de l'espace comme, par exemple, les départements ou les régions.

Des aires d'étude (Areas) : une aire d'étude représente un sous-ensemble nommé d'unités territoriales, associées à une hiérarchie de maillage dans un projet donné.

Des unités territoriales (TU : Territorial Units) : les unités territoriales sont toutes les unités géographiques, quel que soit leur niveau hiérarchique. Elles ne portent pas nécessairement de données attributaires, mais elles ont une description (existence) graphique dans l'application. Cette classe d'objets regroupe donc aussi bien des pays, des communes que des espaces européens (UE15, UE25) ou mondiaux, suivant l'échelle d'étude de l'utilisateur.

Des indicateurs statistiques (Stocks) : ils représentent des données socio-économiques associées à des unités territoriales. Les stocks pourront également être associés à des méta-données telles que leur année, leur provenance, etc.

Des descripteurs d'indicateurs statistiques (Stocks metadata) : porteur d'informations sur les stocks, telles que l'année, leur provenance, etc.

Des langues (Languages) : permettant la désignation et description des entités représentées.

De ce qui précède, nous pouvons déduire qu'un projet comprend une hiérarchie de maillages, constitués d'unités territoriales. Cependant, toutes les unités territoriales n'appartiennent pas à un maillage : certaines ne sont utilisées que pour créer un fond de carte et n'entrent jamais dans une aire d'étude. De plus, un projet regroupe un certain nombre de stocks qui décrivent des réalités socio-économiques sur toute ou partie des unités territoriales d'un projet.

Nota Bene : pour l'instant, *HyperAdmin* ne gère qu'une seule hiérarchie de maillage par projet, car la gestion de plusieurs hiérarchies pose rapidement des problèmes complexes, tels que les correspondances entre les nomenclatures des unités territoriales dans les différentes hiérarchies. Mais, les plus grandes difficultés résident dans le changement des frontières des unités territoriales de base dans le temps. Cela signifie que certaines unités ont fusionné, ou se sont fractionnées, avec le temps (*e.g.* la réunification de l'Allemagne, ou à l'inverse l'éclatement de l'URSS). Ces changements peuvent avoir lieu à tous les ni-

veaux de la hiérarchie : des communes au pays. Ces aspects complexes constituent l'un des objectifs du projet européen « Long Term DataBase » (LTDB).

Après avoir énuméré les concepts manipulés, dans la partie suivante, nous décrivons la manière dont un utilisateur fournit les données permettant la constitution de ces structures.

Les fichiers fournis par l'utilisateur

Etant donné le nombre d'unités territoriales que l'application doit gérer, il est impensable que l'utilisateur saisisse manuellement les données. Il a donc été décidé de fournir à l'application un ensemble de fichiers se divisent en trois catégories :

- Les fichiers de contour, décrivant les contours des unités territoriales ;
- les fichiers de structure, détaillant les relations entre les diverses entités d'un projet ;
- deux fichiers de stocks, contenant les statistiques liées aux unités territoriales.

Nous présentons plus en détails ces fichiers.

Les fichiers de contours

Leur objectif est de fournir les contours des unités territoriales. Notons qu'ils ne contiennent que les contours des unités composant le maillage le plus fin. En effet, rappelons qu'*HyperAdmin* peut exploiter la notion de composition des unités territoriales pour agréer les contours et déterminer ceux des unités parentes.

Format

Le seul format actuellement géré par l'application est le format MIF/MID, format d'échange de MapInfo [@MIF]. Ce format exploite deux fichiers : un fichier d'extension « MIF » et un fichier d'extension « MID » qui sont tous les deux encodés en ASCII.

Le fichier « .MIF »

Le fichier « MIF » comprend un en-tête et la géométrie des entités géographiques (régions, pays, villes, etc.). L'en-tête peut contenir les informations suivantes :

- ✓ la version du format utilisé ;
- ✓ le jeu de caractères (Charset) ;
- ✓ le caractère séparateur (par défaut, il s'agit de la marque de tabulation) ;
- ✓ le paramètre « `Unique` » permettant de créer des tables dérivées ;
- ✓ le paramètre `Index` indiquant les colonnes indexées ;
- ✓ le paramètre « `CoordSys` » permettant de spécifier le système de coordonnées utilisé ;
- ✓ le paramètre « `Transform` » pour convertir les coordonnées conformément au quadrant Nord-Ouest ;
- ✓ le paramètre « `Columns` » qui décrit le format des données tabulaires contenues dans le fichier MID associé. Par exemple, ceci signifie que le fichier « MID » n'aura qu'une colonne, constituée d'entrées de type chaîne de caractères d'au plus 100 caractères.

```
Columns 1
  unit Char(100)
```

Ensuite le mot clé **Data** annonce le début de l'énumération des contours.

Chaque contour est structuré comme suit:

```
Region n
p
X1 Y1
...
Xi Yi
    Pen (1,2,0)
    Brush (0,1)
    Center X Y
```

Chaque entrée « Region » correspond à la description des contours d'une Unité Territoriale (UT), et possède une entrée dans le fichier « MID » qui donne l'identifiant de l'UT. Aucun doublon n'est supporté dans le fichier « MID ».
« n » indique le nombre de polygones que possède cette UT. Par exemple, la France est composée d'au moins deux polygones : la métropole et la Corse. Si une UT est composée de plusieurs polygones, on place dans une seule entrée « Region » autant de polygones qu'il est nécessaire, et on ne dédouble pas le nombre de « Region ».
« p » indique le nombre de points qui définissent un polygone. Ces points sont définis par les couples de coordonnées cartésiennes X et Y sur chaque nouvelle ligne.
Après chaque entrée de polygone, on définit facultativement la couleur et l'épaisseur du contour avec « Pen » et « Brush », et le centre du polygone « Center » par ses coordonnées X Y. Ces données ne sont pas lues par l'application.

Exemple :

```
Version 300
Charset "WindowsLatin1"
Delimiter ","
Index 1
CoordSys NonEarth Units "m" Bounds (-3000000, -3000000) (3000000, 3000000)
Columns 1
  unit Char(100)
Data

Region  1
  11
2186917.593 -1518464.703
2186829.009 -1692861.786
2129979.423 -1729141.275
1933829.46 -1729141.275
1928265.747 -1699690.677
1941660.768 -1656068.01
2005909.794 -1679948.187
2047505.1 -1676110.68
2186917.593 -1518464.703
    Pen (1,2,0)
    Brush (0,1)
    Center 2140313.457 -1623802.989
Region  2
  7
108071.871 -293320.749
96339.456 -282096.297
102833.097 -261179.193
106485.534 -258631.56
123883.98 -262981.491
122621.886 -282959.13
108071.871 -293320.749
    Pen (1,2,0)
    Brush (0,1)
    Center 110111.718 -275976.153
 5
-407753.01 -311500.065
-417000.993 -311417.496
```

```
-411718.965 -289228.641
-406514.985 -302217.573
-407753.01 -311500.065
    Pen (1,2,0)
    Brush (0,1)
    Center -411757.989 -300364.353
```

Le fichier « .MID »

Ce fichier liste les codes des unités territoriales par leur ID, dans le même ordre d'apparition que les régions dans le fichier « MIF ». Le fichier « MID » ne doit contenir qu'une seule colonne.

Le format des entrées sur cette colonne est le suivant: **" UT_ID "**

Exemple:

```
UT_ID
" AT111"
" AT112"
" AT113"
```

Les espaces entre les guillemets n'ont pas d'importance car la chaîne subit un traitement pour les supprimer.

Contenu

Pour le bon fonctionnement du module d'agrégation de l'application, et obtenir un fond de carte de qualité à partir des informations géométriques, il faut que les contours des unités territoriales de plus bas niveau soient sans « trou » et que les unités territoriales soient réellement contiguës. Ce fichier ne doit contenir que de polygones valides selon les recommandations de l'OGC.

Les fichiers de structure

Ces fichiers contiennent toutes les informations permettant de décrire la structure des données, c'est-à-dire, quelles sont les entités qui composent le projet ainsi que la description de leur relation. Ils doivent donc préciser quelles sont les unités territoriales, les aires d'étude et les maillages qui compose le projet ainsi que leurs relations (par exemple, quelle unité appartient à quel maillage…).

Format

Les fichiers utilisés pour décrire ce format sont historiquement des fichiers texte mais pour des raisons de convivialité pour l'utilisateur nous avons décidé d'offrir également la possibilité d'utiliser des fichiers Microsoft Excel.
Dans le cas où l'utilisateur souhaite utiliser des fichiers texte, il doit obligatoirement fournir 9 fichiers pour le fonctionnement de l'application, suivant le modèle physique de données.
Ces fichiers sont nommés ainsi :

- unit.txt
- area.txt
- zoning.txt
- unitsup.txt
- unitzoning.txt
- unitarea.txt
- unitlanguage.txt
- arealanguage.txt
- zoninglanguage.txt

L'utilisateur peut également fournir un seul fichier Excel, devant obligatoirement contenir neuf feuilles nommées de manière similaire aux fichiers texte (unit, area, zoning, unitsup, …).

Fichier texte	Feuille Excel	colonnes	Représentation
Unit.txt	Unit	UT_ID	Ensemble des codes des unités territoriales
Area.txt	Area	UT_ID	Ensemble des aires d'étude
Zoning.txt	Zoning	Zoning_ID	Ensemble des maillages
unitsup.txt	Unitsup	UTSup_ID UT_ID	Hiérarchie entre unités et leurs unités parentes
unitarea.txt	Unitarea	UT_ID Area_ID	Description de la relation unité / aire d'étude
unitzoning.txt	Unitzoning	UT_ID Zoning_ID	Description de la relation unité / maillage
unitlanguage.txt	Unitlanguage	UT_ID UT_NAME Language_ID	Affectation de leurs noms aux unités en fonction d'un langage (code ISO 639-2)
arealanguage.txt	Arealanguage	area_ID area_NAME Language_ID	Affectation de leurs noms aux aires d'étude
zoninglanguage.txt	Zoninglanguage	zoning_ID zoning_NAME Language_ID	Affectation de leurs noms aux maillages

Tableau 4-2 : Détail de la composition des fichiers de structure.

Contenu

Ces fichiers listent les codes des diverses entités et les réutilisent pour définir leurs interactions. Des exemples du contenu de ces fichiers sont fournis en annexe à ce document (*cf.* 3)

Les fichiers de stocks

Ces deux fichiers fournissent les informations statistiques associées aux unités de plus bas niveau de maillage. Depuis la nouvelle version d'*HyperAtlas*, il est possible de gérer des données partielles. Effectivement, il n'est pas toujours possible d'avoir des statistiques pour chaque unité territoriale.

Format

Comme pour les fichiers de structure, la description des stocks peut se faire à l'aide de fichiers texte ou d'un fichier Excel. Ce fichier Excel doit contenir deux feuilles nommées « `stock` » et « `languagestock` ». Il en est de même pour les deux fichiers texte.

Contenu

Le fichier « stock.txt » comprend en première colonne l'identifiant des unités territoriales. Les colonnes suivantes sont toutes les variables de stock que l'on souhaite intégrer dans l'application, les colonnes sont séparées par des tabulations. Il y a autant de colonnes que de stocks présents dans l'application. La première ligne fournie les codes représentant chaque stock. Ce fichier doit se présenter comme ci-dessous :

```
UT_ID      AREAKM2    GDP99ME    GDP99MP    UNT99
AT111      702        520.3      505        0.6
AT112      1793       2499.4     2425.7     1.6
AT113      1471       1326.4     1287.2     2
```

Le fichier « `languagestock` » comprend en première colonne l'intitulé des variables statistiques présentes dans le fichier « `stock` ». La seconde colonne comprend la description de ces variables dans une langue donnée (intitulé long, unité de mesure, date). Le fichier doit se présenter comme suit :

```
Stock      FR
AREAKM2    superficie de l'unité territoriale en km²
GDP99ME    PIB en 1999 en millions d'euros
UNT99      nombre total de chômeurs en 1999 en milliers
```

4.2.2 Choix des technologies

Cette partie présente les divers choix technologiques faits lors de la conception du logiciel *HyperAdmin.* Ces choix s'opèrent à tous les niveaux de notre architecture, de l'accès aux données à l'affichage de celles-ci. Notons également que des notions de temps de réalisation ont également été prises en compte. De plus, nous avons eu pour volonté de réutiliser certaines technologies utilisées dans le projet *HyperAtlas* pour bénéficier des connaissances et des optimisations réalisées dans le logiciel *HyperAtlas*.

Le stockage et l'accès aux données

Les objectifs du projet on été déterminants. Rappelons qu'*HyperAdmin* a pour vocation de gérer et de centraliser les divers jeux de données de l'application. Il a également pour but de permettre l'ajout de nouveaux jeux de données et la mise à jour des données existantes. De plus, il est prévu que notre outil puisse gérer des permissions d'accès en fonction des jeux de données.
Dans ce contexte, l'utilisation d'une base de données semble appropriée de par la proximité de ses fonctionnalités avec celles que nous cherchons à implémenter. Cependant les données manipulées par *HyperAdmin* sont différentes de celles que manipulent classiquement les bases de données. En effet, outre des données basiques telles que des nombres ou des chaînes de caractères, notre outil doit gérer des données géométriques. Outre le stockage et la relecture de ces objets géométriques, *HyperAdmin* doit exploiter des relations propres à ces objets telles que des relations de voisinage ou de contiguïté.

Nous avons donc, dans un premier temps, étudié et comparé les fonctionnalités offertes par divers S.G.D.B.[18]. Très rapidement, nous avons constaté que certain SGBD permettent de stocker et manipuler des données spatiales en implémentant les recommandations formulé par l'OGC [@OGC]. Ce sont notamment les cas de PostgreSQL, MySQL et Oracle. Voici une brève présentation de ces S.G.B.D..

Oracle 10g

Oracle est un système de gestion de base de données édité par la société du même nom. Il bénéficie d'une solide réputation et est le leader du marché selon un rapport édité par le groupe Gartner [@GART]. De plus, outre la gestion des données classiques Oracle propose également une extension « Oracle Spatial » basée sur le modèle de données proposé par l'E.P.S.G.[19]. En d'autres termes, Oracle peut gérer les données contenues dans une carte, c'est-à-dire des points, des lignes, des polygones, etc. La Figure 4-69 illustre les divers types de géométrie supportés, (une géométrie étant une collection ordonnée de sommets). Le modèle de données utilisé par Oracle se compose d'éléments, de géométries, de couches, de systèmes de coordonnée et de tolérances.
Un élément est la brique de base nécessaire à la construction d'une géométrie. Ce sont, par exemple, des points, des chaînes de segments ou des polygones. Notons que le sens d'énumération des points d'un polygone n'est pas sans importance pour Oracle. En effet, le contour extérieur d'un polygone doit être décrit par une liste de points dont l'énumération doit suivre le sens contraire des aiguilles d'une montre alors que les contours des éventuelles enclaves doit se faire dans le sens des aiguilles d'une montre.

[18] S.G.B.D. : Système de Gestion de Base de Données.
[19] E.P.S.G. : European Petroleum Survey Group.

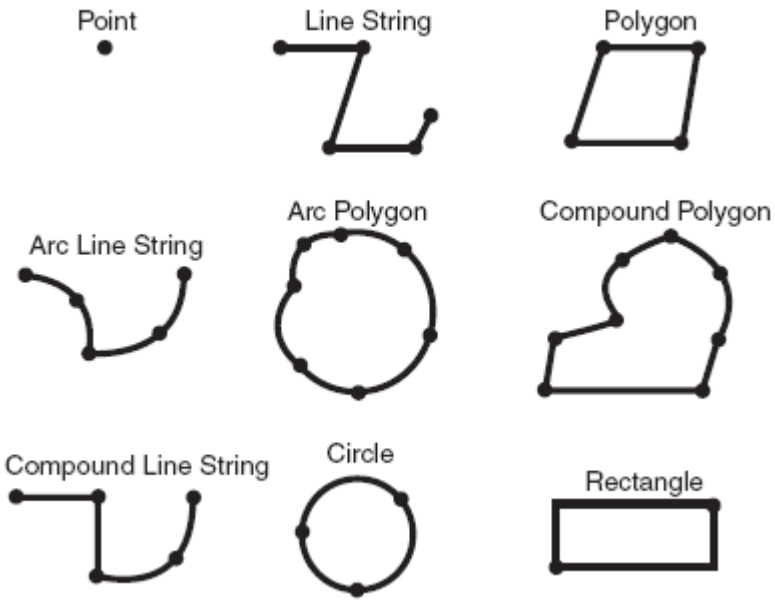


Figure 4-69 : Types de géométries supportées par Oracle Spatial.

Dans ce modèle, une géométrie est définie comme un ensemble d'éléments. Cet ensemble peut contenir un seul élément - un polygone, par exemple - ou plusieurs. Dans le cas où la géométrie se compose de plusieurs éléments de types distincts, on dira qu'elle est hétérogène.
Outre la notion de géométrie, il est possible de définir des couches similaires à celles présentées dans le 2. Une couche est donc une collection de géométries portant un sens particulier, par exemple la couche décrivant la topologie des sols, ou la couche représentant des frontières.
La notion de système de coordonnées est également gérée et permet de définir si les coordonnées d'un objet sont des coordonnées cartésiennes abstraites, des coordonnées géographiques ou bien encore des coordonnées projetées.
Enfin, le modèle de données d'Oracle spatial prend en compte la notion de tolérance. Cette tolérance permet à l'utilisateur de spécifier une distance minimale en deçà de laquelle deux points sont considérés comme identiques. Cette notion revêt toute sont importance selon le type de données manipulées. En effet, imaginons qu'un géographe travaille sur une base de données contenant les contours des Etats européens, deux points espacés de moins de cent mètres peuvent alors être considérés comme étant identiques. Maintenant, imaginons qu'un architecte utilise cette base de données pour stocker les plans des villas, il devient alors impensable que deux points distants de moins de cent mètres soit considérés comme identiques. Pour ces considérations, Oracle laisse le champ libre à l'utilisateur pour définir la tolérance qu'il juge adéquate à ses données.
Outre le stockage de données géométriques, Oracle spatial propose une série de fonctions permettant de réaliser des opérations de comparaison et de modification des géométries. La Figure 4-70 présente les relations topologiques supportées par l'extension spatiale d'Oracle et la Figure 4-71 quelques exemples d'opérations topologiques.

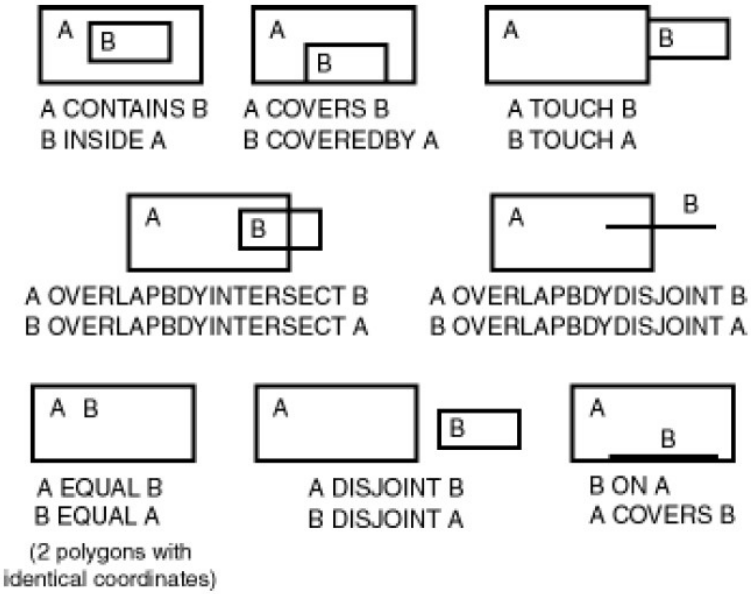


Figure 4-70 : Exemple de relations topologiques prises en charge par le module spatial d'Oracle.

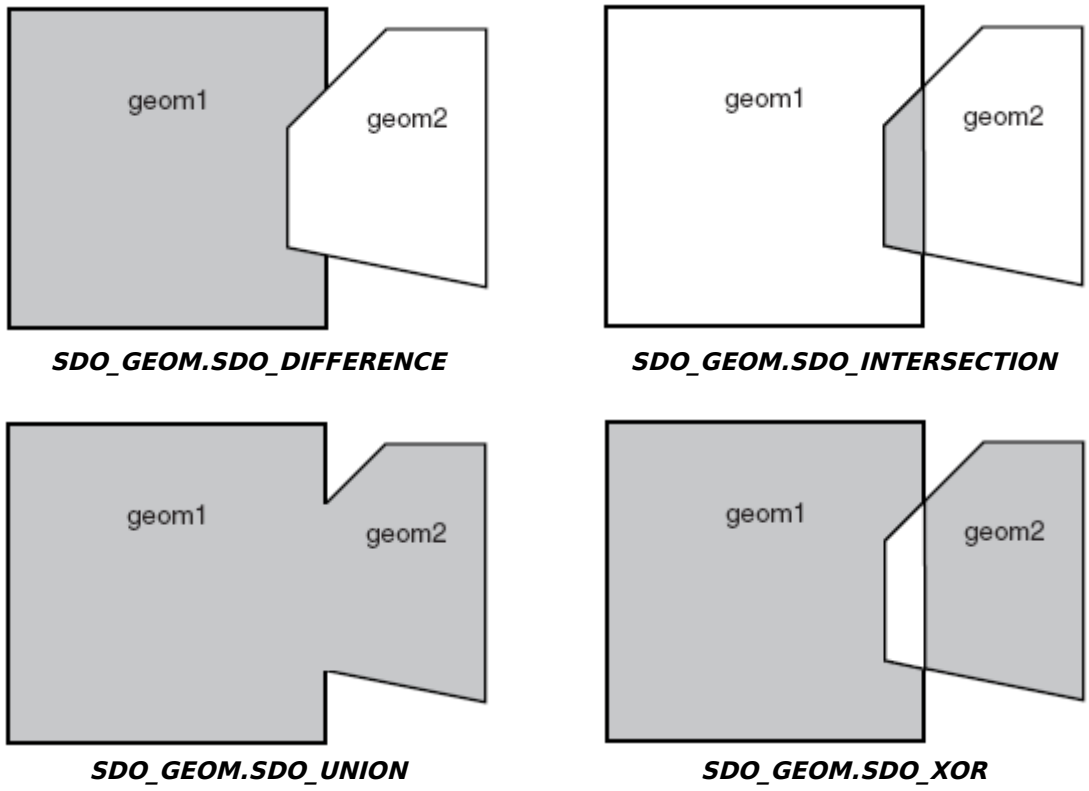


Figure 4-71 : Exemple d'opérations topologiques prises en charge par Oracle spatial.

De plus, pour diminuer les temps d'accès aux données, Oracle utilise également des mécanismes d'indexation permettant de limiter le nombre d'objets à évaluer lors de la réalisation d'une recherche. Deux types d'index sont disponibles : les index à base d'arbres quaternaire (QuadTree) et les index d'arbre R (R-Tree).

Le « QuadTree » utilise des découpages successifs de l'espace en carré pour former une grille régulière dont la finesse dépend du pas choisi [@INFRES], comme le montre la Figure 4-72. Chaque maille du quadrillage contenant tout ou partie d'une géométrie est redécoupée en quatre jusqu'à ce qu'un critère d'arrêt soit atteint, ce critère peut être la taille

de l'un des carreaux du quadrillage ou encore le nombre de carreaux couvrant la géométrie. Les objets de l'espace sont alors référencés par les cellules qui le recouvrent et ces résultats stockés dans une table nommée SDOINDEX.

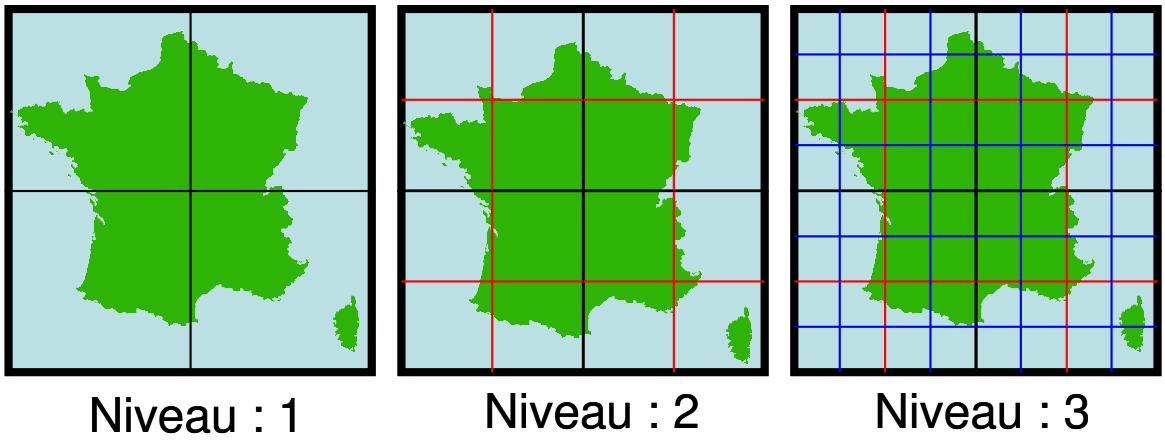


Figure 4-72 : Exemple de découpages utilisés par les méthodes d'indexation par « QuadTree ». La finesse du maillage augmente avec le niveau.

A un niveau donné de maillage, les carreaux obtenus peuvent être triés linéairement par l'application systématique d'une méthode de parcours suivant une courbe de remplissage de l'espace, comme le montre la Figure 4-73. Des codes numériques peuvent également être utilisés pour référencer chaque cellule, ceux-ci sont connus sous le nom de code de Morton. Notons qu'à un niveau de décomposition donné, tous les codes des mailles ont la même longueur. Ainsi l'opérateur d'égalité de SQL « = » peut être utilisé avec de bonnes performances lors de la comparaison des carreaux.

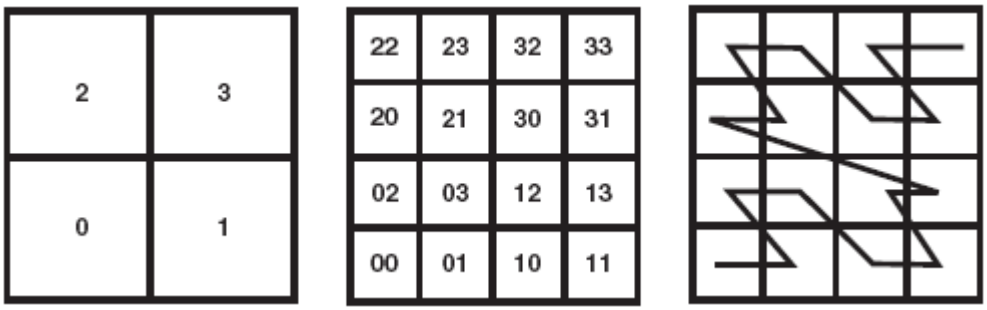


Figure 4-73 : Découpage d'un QuadTree et code de Morton

Cependant, il est important de noter que les performances de ce type d'index sont étroitement liées au rapport entre la finesse du maillage et la taille des géométries. Ainsi, si l'on imagine un découpage avec des carrés de taille importante sur des géométries de taille modeste, l'index n'augmente que de peu les performances. De même, si les même géométries sont superposées à un maillage très fin, le nombre des carreaux se superposant aux géométrie deviens relativement important, augmentant la sélectivité des requêtes mais pénalisant les performances du parcours d'index.
Pour palier ce type de considération, Oracle fournit une méthode permettant de calculer la taille des mailles de manière à offrir les meilleurs compromis. Ainsi la fonction « SDO_TUNE.ESTIMATE_TILING_LEVEL » retourne le niveau de maillage préconisé. L'optimisation de ce type d'index n'est donc pas chose aisée, c'est l'une des raisons pour lesquelles Oracle recommande l'utilisation d'index basés sur les arbres R (R-Tree).

Ces index sont le pendant spatial des « BTree » fréquemment utilisés pour l'indexation des données. Il consiste à créer une structure d'arbre équilibré, c'est-à-dire qui comprend le même nombre de niveaux dans chaque branche. Une recherche dans ce type d'arbre a donc toujours la même durée [@JARG]. Pour atteindre cet objectif R-Tree ne se base pas sur un quadrillage de l'espace mais repose sur l'utilisation des rectangles minimaux englobants de chaque géométrie pour approximer celles-ci, comme le montre la Figure 4-74. L'algorithme des arbres R, proposé par M. Guttman a pour principe de créer un index hiérarchisé autour de ces rectangles englobants que nous appellerons MBR[20] par la suite. La Figure 4-75 nous montre la hiérarchisation dans un index R-Tree. Dans cette figure les éléments numérotés de 1 à 9 sont des géométries contenues dans une couche (layer).

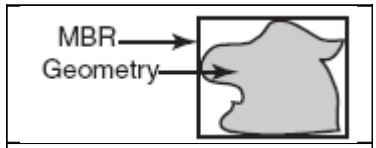


Figure 4-74 : Exemple de géométrie et de son rectangle minimum englobant.

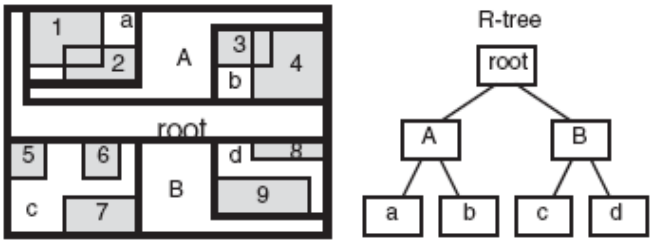


Figure 4-75 : Index hiérarchisé R-Tree basé sur les rectangles englobants minimaux.

Les feuilles *a*, *b*, *c* et *d* de l'arbre contiennent les MBR des géométries. Par exemple, la feuille *a* contient les MBR des géométries 1 et 2, la *b* les MBR des géométries 3 et 4 ...
De même A contient les MBR de *a* et de *b* et ainsi de suite jusqu'à la racine de l'arbre.

Index R-Tree	Index QuadTree
L'approximation des géométries ne peut être réglée avec précision. (Utilisation des rectangles englobants minimaux.)	Les géométries peuvent être approximées précisément en augmentant le niveau de quadrillage.
Création et optimisation aisée de l'index.	L'optimisation de ce type d'index est complexe car elle dépend de la finesse du maillage défini.
Requiert moins d'espace pour le stockage de la structure d'index.	Requiert davantage d'espace pour le stockage de la structure d'index.
Très efficace pour les recherches basées sur le voisinage.	Plus lent que son homologue pour les recherches de voisinage.
Les performances de l'index peuvent être affectées par des mises à jour importantes.	Les performances de l'index sont inchangées par les mises à jour.
Indexation possible jusqu'à quatre dimensions.	Index limité aux structures en deux dimensions.

[20] MBR est l'abréviation de « Minimum Bounding Rectangle ».

Requis pour les index couvrant la Terre entière.	Ne peut être utilisé pour un index portant sur la Terre entière.

Tableau 4-3 : comparatif entre les deux types d'index spatiaux fournis par Oracle.

En conclusion Oracle est une base données complète offrant de nombreuses possibilités et si son utilisation semble plus qu'indiquée dans le cadre de notre projet elle souffre néanmoins d'un coût très élevé par rapport à ses principaux concurrents dont les bases sont gratuites et Open Source.

PostgreSQL 8.1 et PostGIS

Développé à partir du moteur POSTGRES 4.2 conçu par le département de science de l'informatique de l'université de Berkeley en Californie, PostgreSQL est un Système de Gestion de Base de Données Objets Relationnel, pionnier dans de nombreux concepts. Une politique de gratuité et d'ouverture des sources a permis à ce descendant direct de l'implémentation de Berkley de s'imposer comme l'une des références dans le monde des bases de données libres. Aujourd'hui, PostgreSQL implémente une grande partie de la norme SQL et offre de plus d'autres fonctionnalités.
Si PostgreSQL implémente par défaut des types géométriques, une extension nommée PostGIS permet de supporter les données géographiques et implémente la spécification "Simple Features for SQL" publiée par l'OGC. Il est donc possible de gérer des relations entre géométries et de réaliser des opérations topologiques telles que l'union ou la différence de géométries. Plus d'informations sur ces possibilités peuvent être trouvées dans le paragraphe 2.1.3

Cependant, contrairement à Oracle, PostgreSQL n'implémente pas de méthode simple permettant l'exploitation de structures hiérarchiques. Aussi une procédure stockée doit être réalisée pour permettre l'équivalent de l'instruction "CONNECT BY PRIOR" fournie par son concurrent.
PostgreSQL offre malgré tout, une alternative réelle et gratuite à l'utilisation d'Oracle.

MySQL 4.1 et 5.0

Apparu pour la première fois en 1995, MySQL est un système de base de données relationnel développé dans un souci de performances élevées. Historiquement le premier moteur à être géré fut le moteur MyISAM qui utilise plusieurs fichiers grandissant avec la taille des données. Ce moteur a pour avantage d'être très rapide mais ne gère, ni la notion de clé étrangère, ni la notion de transaction[21]. Or, ces deux éléments sont des notions essentielles pour assurer la cohérence des données contenues dans la base. Outre ce moteur, il est également possible d'utiliser un moteur InnoDB créé et maintenu par InnoBase (une Filiale d'Oracle). Ce moteur gère les transactions et les clefs étrangères. En contrepartie, les bases qui l'utilisent occupent bien plus d'espace sur le disque.
De plus, MySQL implémente nativement une partie de la spécification "Simple Features for SQL" publiée par l'OGC. Cependant, dans la version stable de MySQL à l'époque de notre étude les fonctions d'union et de différence entre géométries n'étaient pas implémentées. Elles le sont néanmoins dans la version 5.0 bêta.

[21] Une transaction étant un ensemble d'opérations réalisées de manière atomique, cohérente, isolées et durable. Une définition plus complète peut être trouvée dans le glossaire fourni à la fin de ce document.

Si MySQL, dans sa dernière version peut être utilisé, il n'a pas paru prudent au membre du projet d'utiliser une base de données dans sa version bêta.

Conclusion

Si Oracle 10g propose les fonctionnalités nécessaires au projet, son coût n'en demeure pas moins prohibitif pour le projet. MySQL, quant à lui, ne fournit pas, dans sa version stable, des opérations topologiques essentielles telles que l'agrégation de géométries. PostgreSQL est donc la solution qui s'adapte le mieux aux besoins du projet car il offre toutes les opérations indispensables pour le projet et demeure gratuit et ouvert.

L'affichage des données

Comme nous l'avons vu précédemment, *HyperAdmin*, en plus de permettre l'ajout et la modification de jeu de données dans une base de données, doit permettre également à l'utilisateur d'avoir un rendu graphique des données traitées et de prévisualiser le résultat dans *HyperAtlas*. Dès lors, deux possibilités s'offrent à nous : soit nous recréons une nouvelle interface graphique pour *HyperAdmin*, interface dans laquelle nous prévoyons de pouvoir lancer *HyperAtlas,* soit nous repartons de l'interface existante d'*HyperAtlas* pour l'adapter aux besoins de *HyperAdmin*.

Le développement d'une nouvelle interface pourrait permettre l'utilisation d'un client léger pour l'import et la gestion des données dans *HyperAdmin*. La visualisation se ferait via l'export des données sous forme de fichiers et leur ouverture dans *HyperAtlas*.

Dans le cas contraire, la reprise de l'interface d'*HyperAtlas* offre elle aussi plusieurs avantages :

- une adaptation rapide des utilisateurs d'*HyperAtlas* à notre nouvel outil
- la capitalisation de tous les travaux d'ergonomie et d'esthétique réalisés pour *HyperAtlas*
- une visualisation directe des résultats des ajouts ou modifications de données, sans avoir besoin de lancer l'application *HyperAtlas*
- la diminution des temps d'étude et de conception autour de l'interface

En revanche, la reprise de cette interface nécessite l'utilisation d'un client lourd de type application ou *applet*.
Si nous étudions à présent la possibilité de reprendre cette interface, nous constatons que le découpage en couches quasi indépendantes de *HyperAtlas* nous facilite la tâche. Effectivement comme constaté dans le 3, la couche de représentation d'*HyperAtlas* est indépendante des couches de logique et d'accès aux données car elle exploite le contenu d'un tampon d'unités territoriales et ne se soucie pas de savoir comment et par quoi celui-ci a été rempli. De plus, cette couche utilise également le mécanisme événementiel de *HyperAtlas*, ce qui permet de réduire encore sont couplage avec le reste de l'application.
L'interface utilisateur de *HyperAtlas* doit cependant être enrichie pour permettre une sélection conviviale du projet de travail. Nous avons donc incorporé un arbre sur la partie

gauche de la fenêtre pour permettre de choisir aisément le projet sur lequel on souhaite travailler.

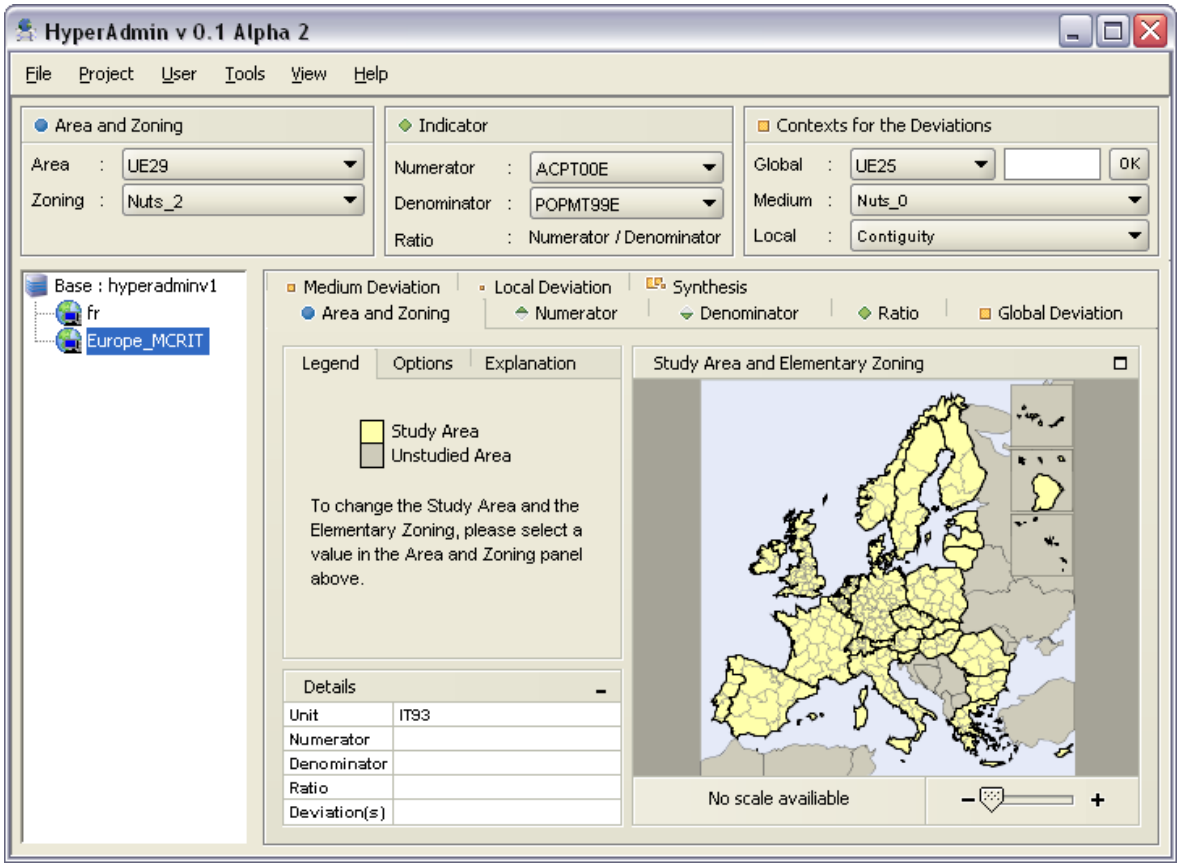


Figure 4-76 : Maquette de l'interface graphique retenue pour *HyperAdmin*.

Pour ces raisons, nous avons choisi d'adapter l'interface d'*HyperAtlas* pour qu'elle convienne aux besoins d'*HyperAdmin*. Nous souhaitons ainsi reprendre l'intégralité des cartes disponibles dans *HyperAtlas* pour permettre à l'utilisateur de visualiser ses projets à l'identique dans les deux applications. Nous ajoutons simplement à cette interface tous les écrans et dialogues spécifiques à la modification de données et à l'accès aux bases de données. Une vue de ces divers écrans est proposée dans la suite de ce document.

4.3 Capture des besoins fonctionnels

Cette partie présente, à l'aide de diagramme UML, les différentes utilisations que font les utilisateurs du système. Pour cela, nous nous basons sur ce qui a été présenté dans notre étude préliminaire.

4.3.1 Les cas d'utilisation

A partir des éléments précédents, nous déduisons les cas d'utilisation présentés ci-dessous dans le formalisme UML.

La Figure 4-77 présente le diagramme UML des cas d'utilisation. Ces cas ont été regroupés en deux grande catégories : l'utilisation et l'administration des données. Un utilisateur de *HyperAdmin* peut ainsi travailler sur des projets en gérant les différents éléments qui le composent alors que l'administrateur peut gérer les comptes utilisateurs et définir les droits

approprié que doit avoir un utilisateur vis-à-vis des données stockées dans la base de données. Nous pouvons noter qu'un administrateur est une sous-classe d'utilisateur car il est lui-même un utilisateur avec des droits particuliers.
Les diagrammes de cas d'utilisation présentés par la suite détaillent chaque cas d'utilisation générique présenté dans la Figure 4-77.

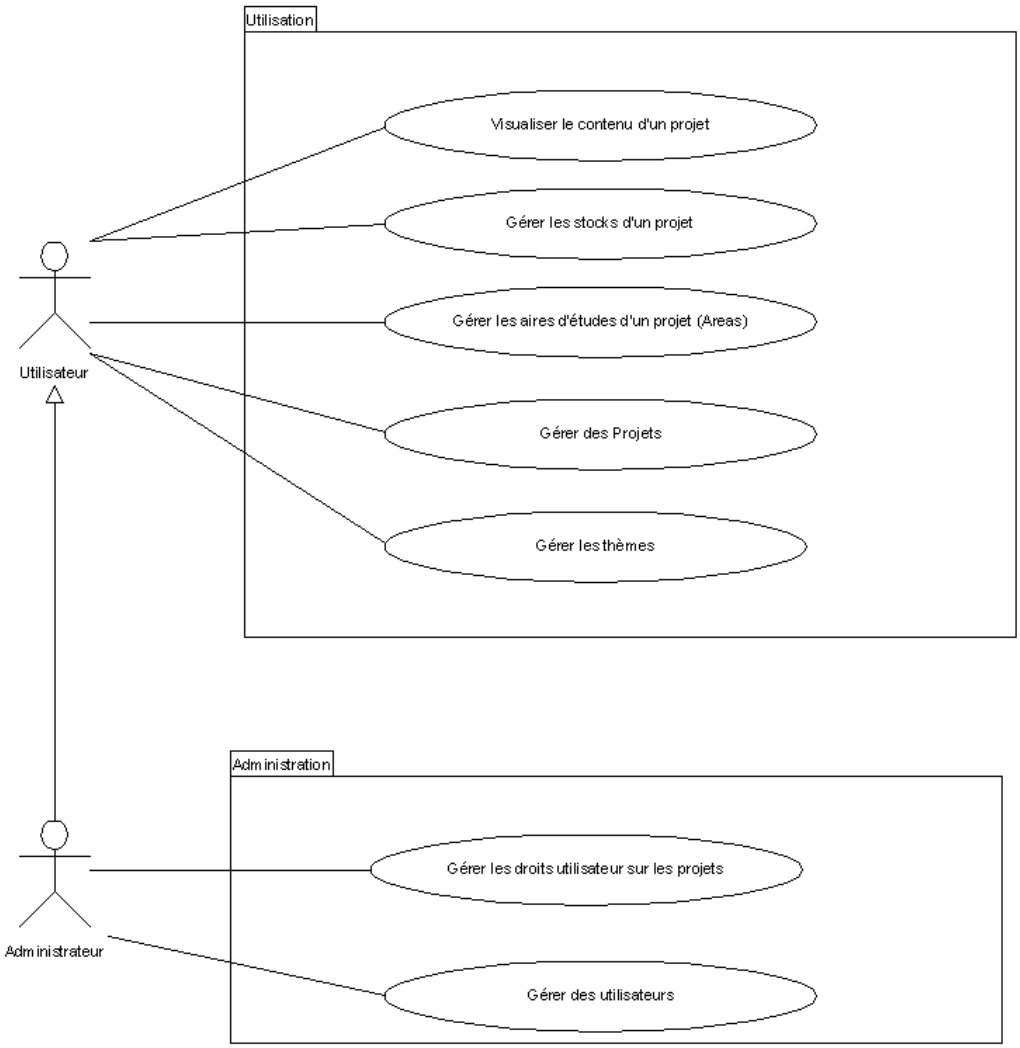


Figure 4-77 : Diagramme de cas d'utilisation de haut niveau.

Comme le montre la Figure 4-78, *HyperAdmin* devra permettre la visualisation des mêmes cartes et implémenter les mêmes fonctionnalités en matière de visualisation dynamique de cartes, que celles proposées par *HyperAtlas*.
La Figure 4-79, la Figure 4-80, la Figure 4-81 et la Figure 4-82, présentent les détails de la gestion des projets, aires d'étude et stocks présents dans l'application.

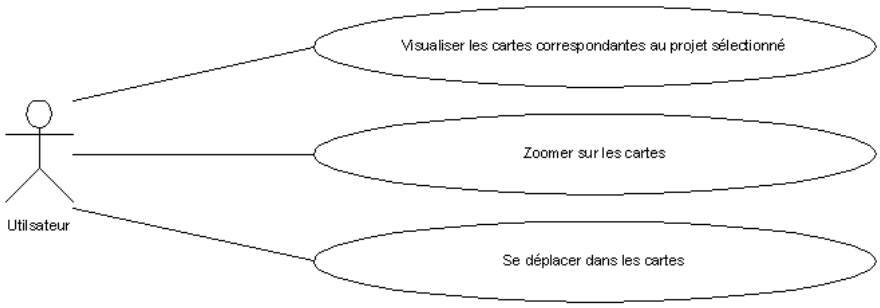


Figure 4-78 : Détail de la visualisation des projets.

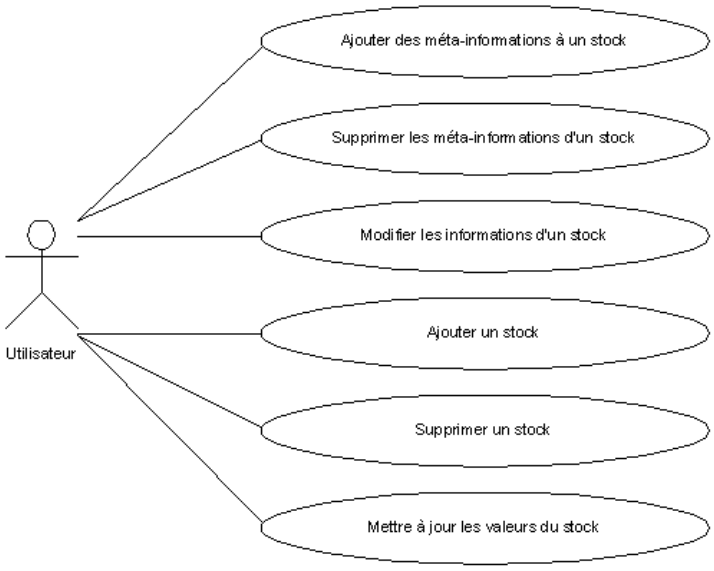


Figure 4-79 : Détail de la gestion des stocks d'un projet.

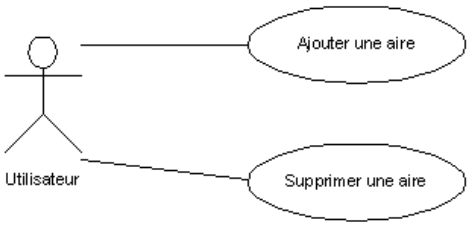


Figure 4-80 : Détail de la gestion des aires d'étude d'un projet.

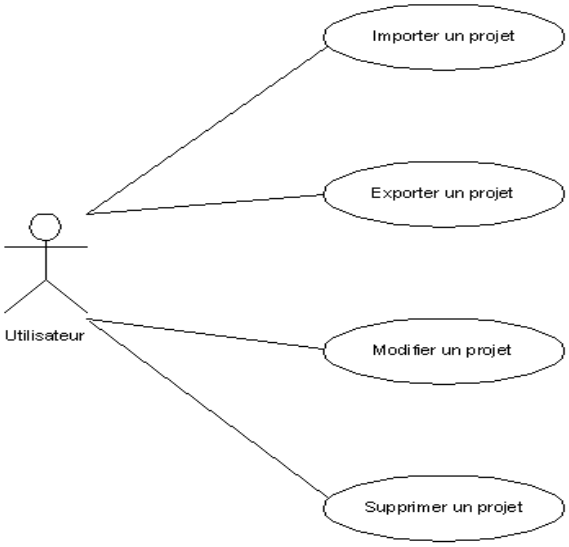


Figure 4-81 : Détail de la gestion des projets.

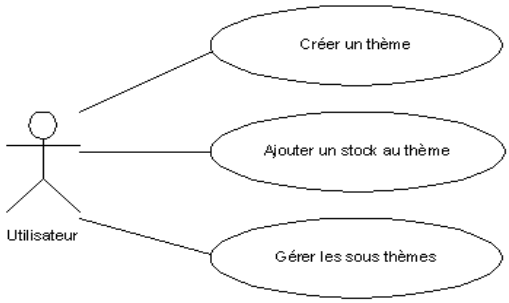


Figure 4-82 : Détail de la gestion des thèmes associés à un projet.

4.3.2 Hiérarchisation des cas d'utilisation

Après avoir présenté les cas d'utilisation nous classons ceux-ci par risque et priorité pour planifier nos itérations sur le projet. Par exemple, la gestion des projets est un point capital des fonctionnalités de l'application et, sans elle l'application n'a pas lieu d'être. Il est donc nécessaire de réaliser ce cas d'utilisation en premier dans notre processus itératif. En revanche la gestion des thèmes pour les stocks revêt un caractère secondaire en proposant une aide à la sélection de stocks pertinents. Cette fonctionnalité ne représente donc pas un risque majeur pour le projet. Le Tableau 4-4 présente un récapitulatif des cas d'utilisation avec les priorités et risques qui leurs sont affectés.

Cas d'utilisation	Risque	Priorité	Itération
Gérer des projets	Haut	Haute	1
Gérer les aires d'études	Moyenne	Moyenne	3
Gérer les stocks	Moyenne	Moyenne	2
Gérer les thèmes	Basse	Basse	6
Gérer les utilisateurs	Moyen	Basse	4
Gérer les droits utilisateurs	Haut	Basse	5

Tableau 4-4 : Tableau de définition des itérations.

4.4 Capture des besoins techniques

Après avoir répertorié l'ensemble des besoins fonctionnels, nous nous intéressons plus particulièrement aux aspects techniques issus de l'étude préliminaire.

Rappelons que nous avons décidé lors de l'étude préliminaire d'axer notre travail sur une architecture comprenant une application JAVA couplée avec une base de données PostgreSQL dotée des extensions PostGIS. Nous nous intéressons maintenant plus en détail aux spécifications matérielles et logicielles que cela suppose.

4.4.1 Spécifications techniques du point de vue matériel

L'architecture matérielle retenue pour notre projet est assez simple car elle se compose de postes de travail équipés de l'application qui peuvent interroger une ou plusieurs bases de données, comme le montre la Figure 4-83.

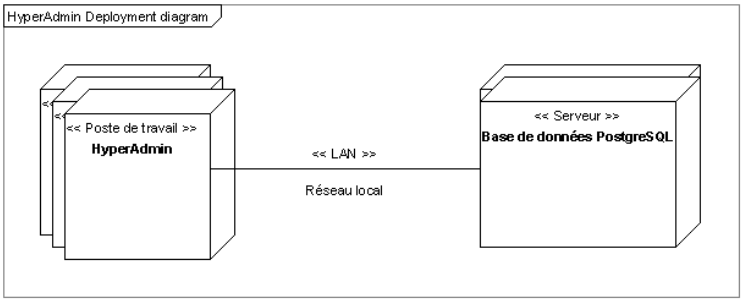


Figure 4-83 : Configuration matérielle du projet *HyperAdmin*.

4.4.2 Architecture logicielle

Comme dans le cas d'*HyperAtlas*, nous choisissons de réaliser une architecture en couche pour notre application. Une couche logicielle représente un ensemble de spécifications ou de réalisations qui, respectivement, expriment ou mettent en œuvre un ensemble de responsabilités techniques et homogènes pour un système logiciel [Roques, 2003]. La Figure 4-84 nous montre les couches retenues pour l'application.

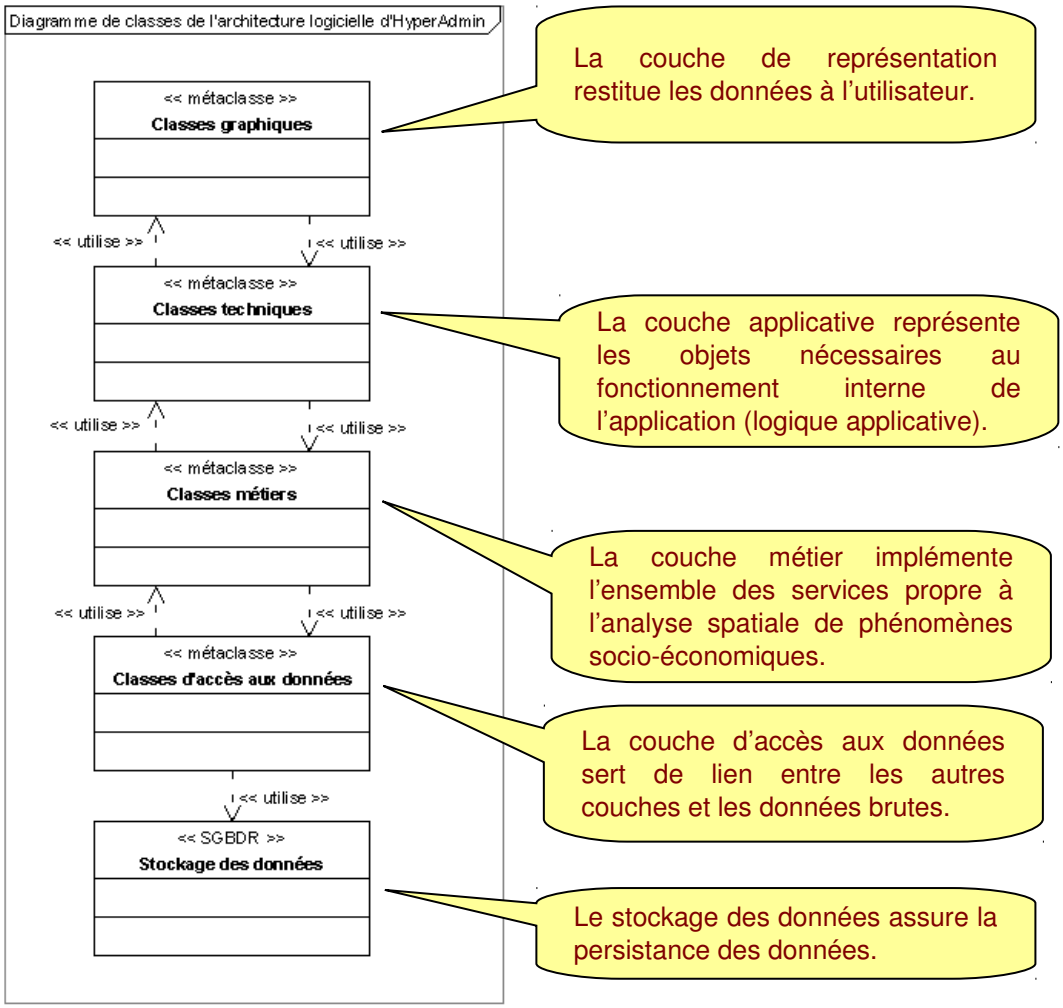


Figure 4-84 : Architecture en couches retenue pour *HyperAdmin*.

4.5 Analyse

Cette partie montre la démarche réalisée lors de la phase d'analyse du projet *HyperAdmin*. Pour cela, nous illustrons nos propos à l'aide de l'analyse réalisée sur les concepts métiers de l'application (projets, unités territoriales, maillages…).
Lors de notre analyse, nous avons élaboré un modèle statique et un modèle dynamique.
Le modèle statique représente la structure et l'organisation des classes de l'application, alors que le modèle dynamique représente le comportement que celles-ci doivent avoir.
Ces deux modèles sont fortement couplés et sont, en général, réalisés en parallèle de manière itérative. Cependant, pour les besoins de ce mémoire, ils sont présentés séquentiellement.

4.5.1 Le modèle statique

A partir de notre étude préliminaire et des captures des besoins que nous avons réalisées, nous pouvons déduire un certain nombre de spécifications qui nous guident dans l'élaboration de notre modèle. Elles nous permettent, entre autres, d'identifier les classes sous-jacentes aux concepts qui apparaissent dans le cahier des charges et d'identifier leurs relations.

Par exemple, à partir du cahier des charges et des phases de capture des besoins nous pouvons noter les points suivants :

- un projet ne traite que d'une hiérarchie ;
- un projet contient des unités territoriales ;
- un projet contient des maillages ;
- un projet contient des aires d'études ;
- un projet contient des stocks ;
- un stock peut avoir une description ;
- la description d'un stock peut caractériser un stock dans son intégralité (année, nature…) ou le caractériser en fonction de sa relation avec une unité territoriale (provenance, auteur…). Il est facile de comprendre que les valeurs de certains stocks proviennent de plusieurs instituts statistiques : INSEE lorsqu'elles concernent la France, DESTATIS pour l'Allemagne...
- un stock est porteur d'une valeur pour un sous-ensemble des unités territoriales d'un projet ;
- les maillages sont ordonnés par nature ;
- les unités territoriales peuvent être une agrégation d'unités territoriales appartenant à un maillage de rang inférieur (une région est l'agrégation des départements qui la composent) ;
- une aire d'étude est un ensemble nommé d'unités territoriales ;
- les stocks peuvent s'organiser par thèmes ;
- un thème peut contenir tour à tour des stocks ou des sous thèmes ;
- …

Partant de ces spécifications, nous pouvons relever un ensemble de concepts propres à devenir des classes et un ensemble de relations qui lie nos concepts. Par exemple, nous pouvons noter les concepts de projet, d'aire d'étude, de maillage, d'unité territoriale, de stock…

De plus, nous pouvons identifier une relation de composition entre un projet et des unités territoriales,…
A partir de cet ensemble de constatations, nous pouvons créer le diagramme de classes suivant (Figure 4-85) qui est la représentation normalisée selon UML de ces spécifications.

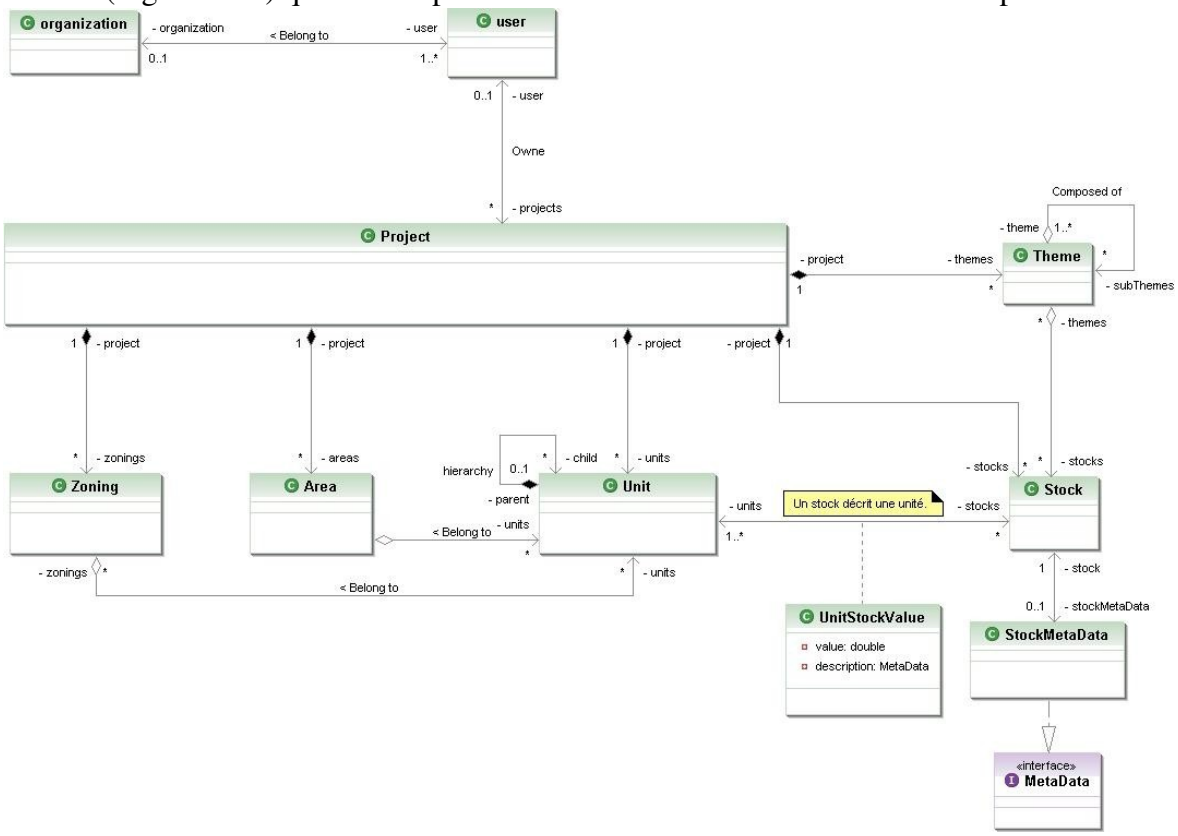


Figure 4-85 : diagramme de classes simplifié de l'application.

Une fois ce diagramme de classes établi, nous l'affinons ensuite en déterminant quels sont les attributs et les opérations des classes.

4.5.2 Le modèle dynamique

Ce modèle a pour objet la description du comportement attendu de notre application. Il s'agit donc, dans un premier temps d'identifier et de formaliser des scénarios, puis de les utiliser pour construire des diagrammes d'état du système présentant les divers états de celui-ci.

Dans la première phase, nous repartons des cas d'utilisation déterminés précédemment pour mettre en évidence les scénarios. Notons qu'un cas d'utilisation peut recouvrir plusieurs scénarios. Un scénario représente une séquence d'interactions entre le système et ses acteurs, le système étant représenté jusqu'alors comme une « boite noire ». Maintenant, nous nous attachons à remplacer cette boite noire par une collaboration d'objets issus de notre analyse statique.
Les scénarios décrivent une exécution particulière d'un cas d'utilisation, aussi il est possible de les regrouper en plusieurs catégories :

- les scénarios nominaux, qui réalisent les post-conditions du cas d'utilisation, d'une façon naturelle et fréquente
- les scénarios alternatifs, qui réalisent les post-conditions du cas d'utilisation mais selon un processus plus rare

- les scénarios aux limites, qui réalisent les post-conditions du cas d'utilisation en changeant l'état du système, de telle sorte qu'une nouvelle exécution du cas d'utilisation provoquera une erreur
- les scénarios d'erreurs qui ne réalisent pas le cas d'utilisation

Si nous prenons, par exemple, le cas d'utilisation « Création d'un nouveau projet » nous pouvons identifier les scénarios suivants :

- scénarios nominaux :
  - création d'un projet valide
  - annulation de la création d'un projet

- scénarios alternatifs :
  - création d'un projet avec substitution d'un code langage non valide selon la norme ISO

- scénarios aux limites :
  - aucun

- scénario d'exception :
  - incohérence des données nécessaires à la création

Après avoir identifié les scénarios, nous nous attachons à formaliser ceux-ci à l'aide de diagrammes UML qui ont pour but de fournir une représentation visuelle et normalisée du déroulement d'un scénario. Rappelons qu'un scénario peut se représenter par un ensemble de messages échangés par des objets définis au sens large (instance de classe, d'analyse ou d'acteur).

Si nous prenons l'exemple du scénario : « Création d'un projet valide ». La séquence de message qui la compose est la suivante :

- L'utilisateur fournit au système l'ensemble des données nécessaires à la création d'un projet.

- Un objet « ProjectManager » utilise ces données pour créer un projet et deux sources de données différentes : l'une pour gérer les données conventionnelles (« DataSource ») et l'autre spécifique aux données cartographiques (« MapDataSource »).

- Une fois les objets sources de données créés, le « ProjectMananger » utilise ceux-ci pour lire et interpréter les données contenues dans les fichiers. Notons qu'ici les classes « DataSource » et « MapDataSource » jouent le rôle d'interfaces objets entre le programme et les fichiers textes ou excels. Il commence donc à lire la liste des différentes unités territoriales présentes dans le projet.

- Cette liste d'unités établie, il utilise l'instance de la classe « MapDataSource » pour extraire les contours géographiques de ces unités et affecter ainsi les contours à leurs unités respectives.

- L'instance du « ProjectManager » poursuit son travail en lisant la liste des aires d'études et en créant celle-ci.

- Il fait de même pour les autres entités du projet telles que les maillages et les stocks.

- Après avoir créé toutes les entités qui composent le projet, le « ProjectManager » interroge son instance du « DataSource » pour créer les liens entre ces entités. Il lie ainsi les unités territoriales à leur(s) maillage(s) et à leur(s) aire(s) d'étude. Il crée aussi des relations valuées entre stocks et unités territoriales.

- Enfin, le « ProjectManager » passe l'instance de projet ainsi créé en paramètre de la méthode « createProject » de l'objet « DBOutput » qui se charge de créer le projet dans la base de données.

Cette séquence de messages peut apparaître dans un diagramme de séquence ou un diagramme de collaboration. Notons qu'il n'y a pas de différence sémantique entre ces deux diagrammes mais juste une mise en valeur d'aspects différents du scénario. L'un insiste sur la chronologie des événements, () alors que le diagramme de collaboration insiste sur les liens entre les diverses classes ou acteurs.

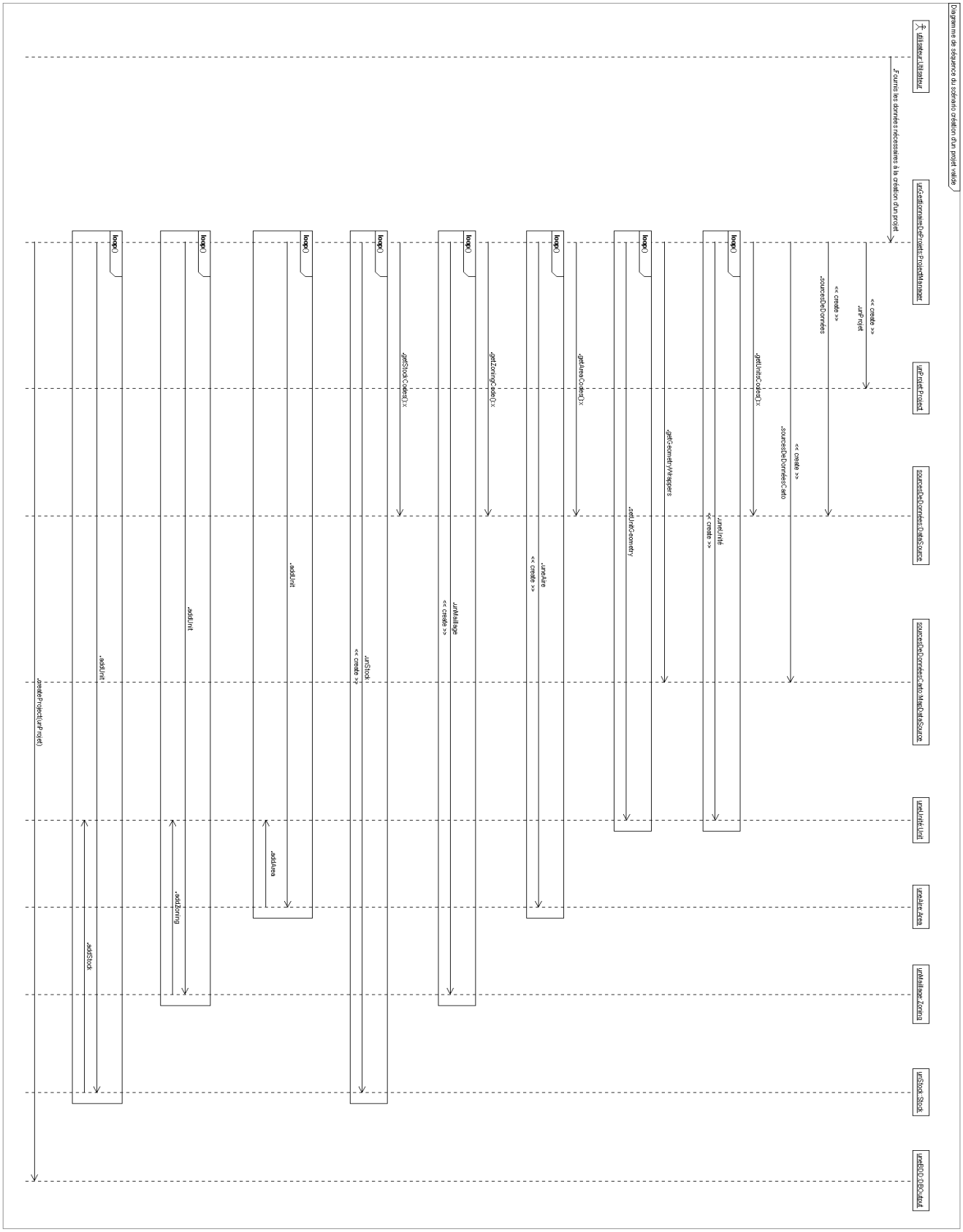


Figure 4-86 : Diagramme de séquence pour le scénario création d'un projet valide.

4.6 Architecture technique

Lors de cette étape, notre objectif est de définir des solutions techniques répondant aux besoins exprimés lors de la phase d'expression des besoins techniques. Cette phase appartient à la branche droite de notre processus en « Y » et peut être menée parallèlement à l'analyse. Effectivement, son principal objectif est de dégager des ensembles de classes collaborant entre elles pour réaliser une responsabilité technique. Par la suite, nous appelons ces ensembles des « *Frameworks* » et nous nous intéressons plus particulièrement au « *frameworks* » dits techniques, car ils permettent de réaliser des besoins techniques tel que l'affichage, le stockage des données, etc.

HyperAdmin est composé de trois *frameworks* techniques qui collaborent en son sein. Un premier *framework* gère l'accès et le stockage des données, un autre permet l'affichage de ces données, et enfin un troisième gère les différents événements qui rythment la vie de l'application.
Ces *frameworks* existent déjà dans *HyperAtlas* et ne nécessitent, pour la plupart, qu'une extension ou une remise à jour. Par exemple, *HyperAtlas* utilise un *Framework* permettant l'accès aux données mais non leur mise à jour. Dans *HyperAdmin* nous réutilisons ce *framework* en l'enrichissant pour permettre la modification des données. La Figure 4-87 nous montre l'exemple du *framework* de gestion des données. Historiquement, ce *framework* avait pour unique objectif de lire des fichiers de donnée pour *HyperAtlas*. Nous l'avons étendu pour l'écriture et la modification des données nécessaire pour *HyperAdmin*.

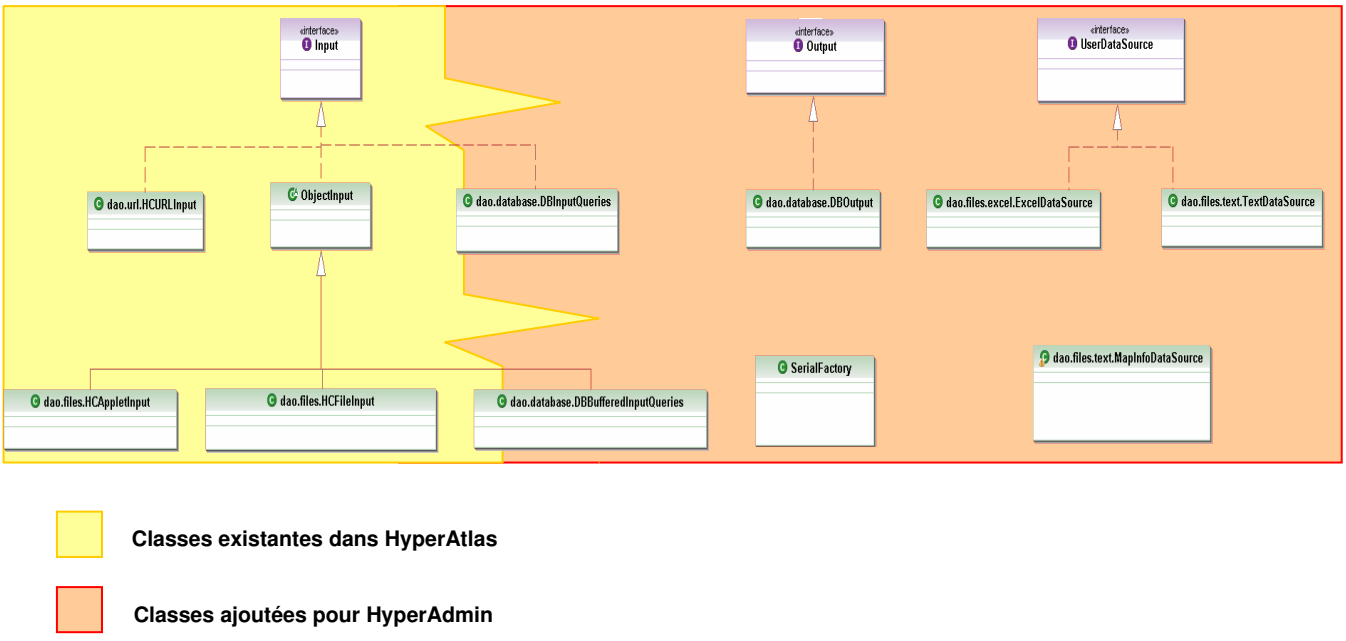


Figure 4-87 : Diagramme de classes présentant le *framework* d'accès aux données et soulignant la partie déjà existante dans *HyperAtlas* et les ajouts réalisés dans le cadre du développement d'*HyperAdmin*.

Comme nous l'avons vu, *HyperAdmin* enrichit des *frameworks* utilisés par *HyperAtlas*. Nous avons donc identifié les classes composantes de ces *frameworks* et nous les avons

isolés dans un package spécial utilisé par les deux applications. Ainsi chaque mise à jour réalisée dans le cadre d'une application profite également à l'autre, cela permet également de faciliter la maintenance applicative dans le sens où il n'est pas nécessaire de gérer une version de classe par application.

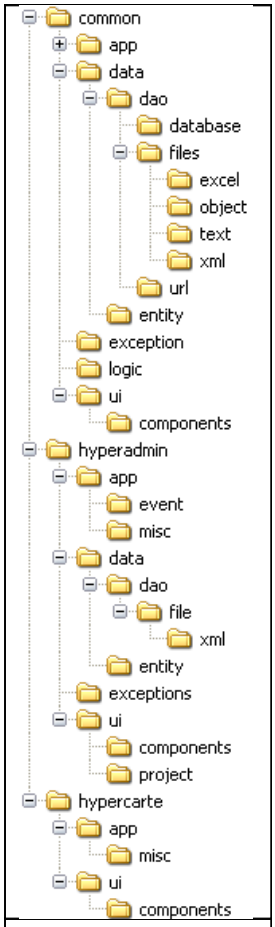


Figure 4-88 : Arborescence des packages proposée pour *HyperAdmin*.

Ainsi, nous avons regroupé les deux applications au sein du même projet Eclipse et nous avons mis en place l'arborescence de packages présentée par la Figure 4-88.

Nous pouvons constater que cette arborescence se compose de trois packages racine :

- le package « `common` » regroupe les classes communes au deux outils ;
- le package « *`HyperAdmin`* » contient les classes spécifiques à *HyperAdmin* ;
- le package « `hypercarte` » se compose de celles qui sont propres à *HyperAtlas*.

A l'intérieur de chacun de ces packages se retrouvent d'autre packages représentant les couches du système.

Ainsi, le package « `app` » contient les classes de la couche technique, le package « `data` » intègre le *framework* d'accès aux données, le package « `logic` » représente la logique métier, et enfin le package « `ui` », synonyme de « User Interface », couvre les classes graphiques nécessaires à la présentation.

Cette étape d'identification des packages terminée, il est alors nécessaire de déterminer les relations entre les différents sous-systèmes qu'ils représentent.

4.7 Conception

Cette partie a pour objectif de présenter au lecteur les choix réalisés et implémentés pour concevoir l'outil *HyperAdmin*. Elle est composée de cinq sous parties correspondant aux couches applicatives de notre système.
Dans un premier temps, nous abordons la partie stockage des données en présentant au lecteur la structure de la base de données à utiliser. Puis, nous étudions comment notre application accède à ces données et quels sont les traitements réalisés dans l'application et ceux réalisés dans la base de données.
Un bref rappel sur la couche technique et la couche métier sera également proposé.
Enfin, nous détaillons les modifications réalisées dans la couche de présentation.

4.7.1 Structure de la base de données *HyperAdmin*

Après avoir décrit le modèle physique de données (Figure 4-89), nous présentons dans un premier temps les tables issues du modèle de données illustré par la Figure 4-85. Ces tables sont une représentation relationnelle des concepts du domaine couvert par *HyperAdmin*. Le schéma suivant présente les liens entre les différentes tables de la base de données *HyperAdmin*.

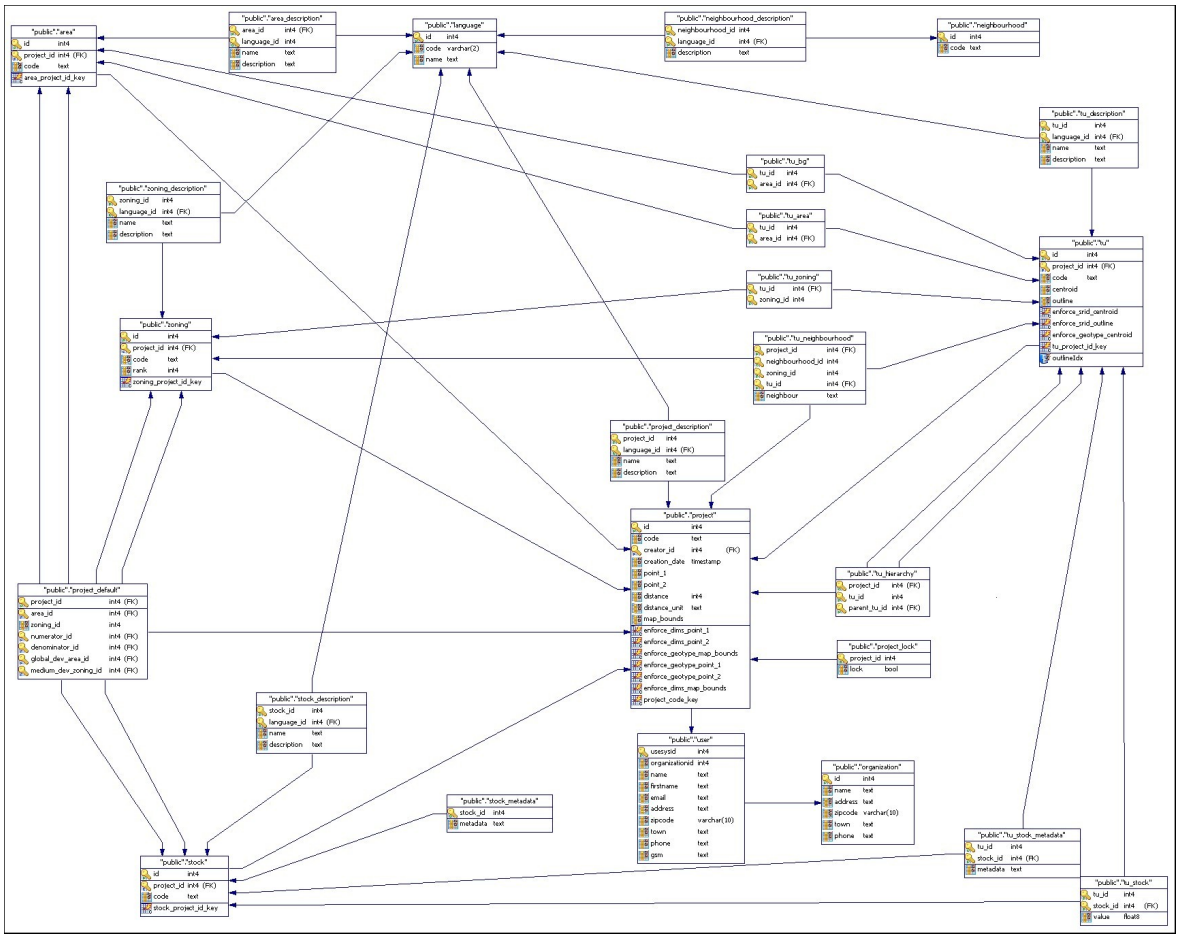


Figure 4-89 : Modèle physique de données de la base *HyperAdmin*.

Parmi toutes ces tables nous distinguons quatre types de tables : celles représentant des entités du domaine (projet, unité territoriale, …), les tables nécessaires au stockage des descriptions des entités du projet, les tables représentant des relations entre ces entités, comme par exemple, le lien d'appartenance entre unités territoriale et aires d'études.
Enfin nous utilisons des tables créées uniquement pour des raisons de performance.

La description des tables est faite en respectant le formalisme suivant :

- Les colonnes utilisées comme clé primaire sont mise en gras et soulignés (ex. **cle_primaire**)
- Les clés étrangères sont uniquement en gras (**une_cle_etrangere**)
- Les colonnes devant respectées une contrainte d'unicité sont en gras et italique (***mon_unique_code***)

Pour des raisons de place nous ne présentons pas les 27 tables constituant la base de données mais une table de chacun des trois types présentés précédemment.

La table « tu » est un exemple de tables permettant le stockage des propriétés des entités du projet *HyperAdmin*. Cette table est utilisée pour stocker les informations propres à une unité territoriale.

Table : « tu »

Cette table représente les unités territoriales des différents projets stockés dans la base.

Colonnes :

nom	type	Description
id	serial	Identifiant numérique unique d'une organisation
project_id	int4	Identifiant du projet d'appartenance de l'unité
code	text	Code alphanumérique de l'unité
centroid	geometry	Centroïde de l'unité territoriale
outline	geometry	Contour de l'unité territoriale

Tables Référencées

- « project » via la colonne « project_id ».

Les noms des unités sont exprimés dans plusieurs langues, la représentation de cette correspondance utilise une table réalisant cette association entre une entité, une langue, et la description donnée dans cette langue. Dans le cas de notre unité territoriale il existe une table nommée « tu_description » réalisant ce lien, cette dernière est présentée ci après.

Table : « tu_description »

Cette table stocke les différentes descriptions d'une unité territoriale en fonction des différentes langues disponibles.

Colonnes :

Nom	type	Description
tu_id	int4	Identifiant de l'unité décrite
language_id	int4	Identifiant de la langue choisie
name	text	Nom de l'unité
description	text	Une description facultative

Tables Référencées

- « tu », via la colonne « tu_id »
- « language », via la colonne « language_id »

On peut maintenant constater que les tables utilisées pour décrire les relations entre des entités sont assez similaires. Ainsi la table « tu_area » présentée ci-dessous, est utilisée pour stocker les liens entre unités territoriales et aires d'études.

Table : « tu_area »

Cette table représente les différentes organisations qui utilisent l'application. La gestion des droits et privilèges utilisateur nécessite d'accéder à cette table.

Colonnes :

nom	type	Description
tu_id	int4	Identifiant de l'unité décrite
area_id	int4	Identifiant de l'aire contenant l'unité

Tables Référencées

- « tu », via la colonne « tu_id »
- « area », via la colonne « area_id »

Enfin, nous nous intéressons aux tables implémentées pour ses raisons de performances. La plupart de ces tables pourraient être remplacée par des vues dans certains SGBD, mais PostgreSQL, s'il gère des vue, ne gère que des vues logiques et non des vues matérialisées. En clair, certain SGBD, comme Oracle, écrivent sur le disque les données issues d'une vue comme s'il s'agissait d'une table. En revanche, PostgreSQL implémente les vues comme des requêtes nommées et donc l'interrogation d'une vue n'est ni plus ni moins performante que l'exécution d'une requête. Ainsi la création d'une table permet de donner une existence physique à la vue et permet donc d'augmenter les performances.

Table : « bg_tu »

Cette table est utilisée pour des raisons de performance en stockant les unités territoriales n'appartenant pas à une aire d'étude mais qui sont dessinées pour constituer le fond de carte.

Colonnes :

nom	Type	Description
tu_id	int4	Identifiant de l'unité décrite
area_id	int4	Identifiant de l'aire contenant l'unité

Tables Référencées

- « tu », via la colonne « tu_id »
- « area », via la colonne « area_id »

4.7.2 La couche d'accès aux données

Nous avons choisi d'utiliser la librairie JDBC dans sa version 3 pour accéder à la base de données depuis notre application Java. Cette librairie nous permet, en plus des opérations de lecture écriture basique, de créer des requêtes précompilées appelées « `PreparedStatement` ».
Ces requêtes offre deux avantages principaux : la performance et la sécurité.
En effet, le but d'un « `PreparedStatement` » est de préparer une requête type à être exécutée, Cette préparation se décompose en général de la manière suivante :

- Analyse syntaxique de la requête
- Création d'un arbre d'exécution

Dans le cas d'un « `PreparedStatement` » le SGBD n'effectue ces opérations qu'une seule fois et en mémorise le résultat. En conséquence, lors des appels suivant à la requête ces

phases ne sont pas réitérées inutilement d'où un gain de temps croissant proportionnellement au nombre d'appels et à la complexité de la requête ainsi préparée.

De plus, dans certain cas les « `PreparedStatement` » sont recommandés pour des raisons de sécurité car ils permettent d'éviter l'injection de code SQL dans les requêtes[22].

En outre, nous utilisons des procédures stockées pour optimiser certains traitements sur les données, telles que la récupération de toutes les unités filles d'une unité à un niveau de maillage donné, ou bien encore des opérations typologiques qui exploitent l'extension PostGIS. Là encore, il est légitime de s'interroger sur la nécessité de déléguer ce type de traitement à la base plutôt qu'à la logique d'*HyperAdmin* mais la raison est relativement simple : PostgreSQL et son extension PostGIS permettent de créer des index spatiaux qui accélèrent sensiblement les calculs. Cette différence de temps est d'autant plus importante que PostGIS est basé sur des librairies écrites en langage C qui reste, encore aujourd'hui, beaucoup plus performant que les programmes Java.

Notons également que cette couche exploite la notion de transaction pour garantir la cohérence des données. Les transactions sont utilisées pour toutes les modifications concernant les projets, y compris la création d'un nouveau projet.
Rappelons qu'une définition de la notion de transaction peut être trouvé dans le glossaire de ce document.

4.7.3 Les couches métiers et techniques

Ces couches sont celles qui ont le moins changé entre *HyperAtlas* et *HyperAdmin*. Ainsi, les traitements métiers sont les mêmes que pour *HyperAtlas,* car les cartes présentées sont identiques et nécessite donc les mêmes calculs.
Les classes techniques telles que les classes de gestion des événements ont simplement été étendues avec de nouveaux événements propres à *HyperAdmin* (connexion à une base de données, création d'un nouveau projet…). La capitalisation sur ses deux couches a donc été maximale.

4.7.4 La couche de présentation

Bien que la couche de présentation d'*HyperAtlas* soit assez aboutie, des modifications et optimisations ont été nécessaires. Pour mieux comprendre ces nouveaux besoins nous rappelons brièvement le fonctionnement des interfaces graphiques Swing pour mieux situé où se trouvent les besoins d'amélioration.

Swing est une libraire permettant la création d'interfaces graphiques en Java. Elle permet de créer des fenêtres, dialogues, des menus, des boutons… Pour pouvoir exploiter ces divers éléments graphique elle utilise la notion d'événements proposée par le langage Java. Ces événements sont envoyés, par exemple, lorsque l'on clique sur un bouton ou que l'on ouvre un menu et même lorsque l'on souhaite fermer une fenêtre. Il est donc indispensable que tous ces événements soient traités dans l'ordre.

[22] L'injection de code SQL est une vulnérabilité des bases de données souvent employé par les pirates voulant disposer de droits supplémentaires sur les bases. Elle consiste à insérer du code SQL en lieu et place des paramètres d'appel d'une requête. Il s'en suit donc une modification du comportement de la requête qui interprète ce code malveillant comme étant le sien.

Pour garantir que tous ces événements arrivent et soit traités, les concepteurs de Swing ont regroupé toutes les opérations graphiques dans un processus (Thread) unique nommé EDT pour (Event Dispatch Thread).

Si ce mécanisme permet de s'assurer que tous les événements sont traités dans leur ordre d'émission, il présente néanmoins une faiblesse. Imaginons que dans une application utilisant Swing, nous souhaitons ouvrir un fichier en cliquant sur le menu ouvrir. Lorsque l'utilisateur clique sur le contrôle permettant l'ouverture du fichier celui-ci envoie un événement, cet événement est intercepté par un écouteur d'événement qui lance le traitement approprié, ici celui qui ouvre notre fichier. Maintenant nous savons que cet écouteur s'exécute dans l'EDT. Or s'il ouvre directement le fichier l'EDT ne peut rien faire d'autre pendant ce temps. Notamment il ne rafraîchît pas l'interface graphique ce qui peut conduire à des bugs d'affichage si l'ouverture du fichier prend du temps, comme le montre la Figure 4-90.

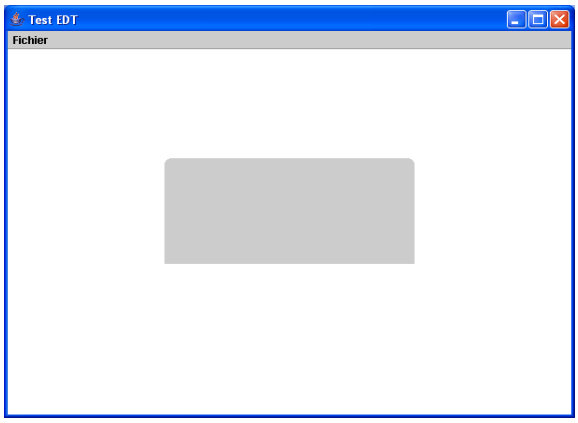


Figure 4-90 : Exemple d'un problème de rafraîchissement lié à l'exécution d'une opération longue dans l'EDT.

Pour palier, ce genre de problèmes, il est donc nécessaire de lancer les traitements métier en dehors du processus graphique EDT. Ainsi, à chaque action doit correspondre un processus (Thread) s'exécutant en parallèle à l'EDT et se synchronisant avec celui-ci.

4.8 Synthèse

Les développements menées dans le cadre du projet HyperAdmin ont conduit au développement d'une version *bêta* de l'outil. Cette version permet entre autre de lire des fichiers de contours (format MIF/MID), des fichiers de structures et des fichiers contenant les stocks à rattacher aux unités territoriales. Comme nous l'avons vu, les fichiers de structures et de stocks peuvent être soit des fichiers textes soit des fichiers Excel.
Une fois les données des fichiers extraites, elles sont stockées et organisées sous forme de projets dans une base de données PostgreSQL.
Chaque projet peut également être exporté sous la forme d'un jeu de données exploitable par *HyperAtlas*. Cette version *bêta* de l'application nous a permis de générer des jeu de données concernant le Cameroun, la Roumanie et une nouvelle version des données européennes.
Cependant, le manque de temps nous a conduit à une réalisation de tests unitaires et de tests d'intégrations ne respectant pas un processus normé mais à réaliser ces tests au fur et à mesure du développement.

Un travail de normalisation de ces tests ainsi que l'aboutissement de la phase de validation reste à mener pour garantir la qualité de la future première version d'*HyperAdmin*.

CHAPITRE 5

CONCLUSION ET PERSPECTIVES

5.1 Problème posé

Ce travail a débuté avec l'objectif d'ajouter des fonctionnalités et de faire évoluer *HyperAtlas*. Ces ajouts ont été principalement guidés par les retours d'expériences des équipes du laboratoire Géographie-cités, mais également dans un but de prospection et de recherche. Notons que la plus attendue des fonctionnalités, charger d'autres cartes dans l'application, correspond également à l'évolution la plus importante apportée sur l'architecture d'*HyperAtlas*.
Comme présenté dans la section 3.2, la version précédente de l'application a été développée à partir d'un prototype. Celui-ci a donné lieu à de nombreuses améliorations portant notamment sur l'ergonomie de l'interface utilisateur et sur les performances de l'affichage des cartes.

Cependant, lors du développement un seul et unique jeu de données était disponible. Ceci a eu pour conséquence de rendre l'application dépendante de ce dernier. En d'autres termes, certains paramétrages nécessaires à l'affichage de la carte était directement issu du jeu de données et n'étaient pas calculés par l'application. Ainsi, lorsque d'autres jeux de données ont été conçus par les équipes du laboratoire Géographie-cités les résultats de leur exploitation par *HyperAtlas* ne furent pas satisfaisants : les cartes produites étaient de tailles extrêmement variables et parfois affichées en dehors des limites de l'écran. Afin de pallier ce problème, les équipes du laboratoire Géographie-cités ont été contraintes de procéder à des transformations sur les coordonnées des unités territoriales de telle sorte à ce que l'affichage des cartes soit le plus correcte possible. En d'autres termes, l'exploitation de nouvelles données nécessitait systématiquement une adaptation de celles-ci à l'outil. Malgré toutes ces adaptations, le calcul de l'échelle étant basé sur le jeu de données initial, celle-ci était souvent fausse !

De surcroît, le développement d'un nouvel outil apte à intégrer et gérer les données de manière convivial a été décidé. Cet outil doit permettre d'importer des fichiers de formats différents, de stocker leur contenu dans une base de données et de réexporter des jeux de données exploitables par *HyperAdmin*.

5.2 Bilan des réalisations

La première étape de ce travail fut d'analyser la version existante d'HyperAtlas afin de proposer une solution permettant de rendre l'application indépendante vis-à-vis de son jeu de données.

Nos travaux d'analyse et de conception nous ont permis de changer l'architecture d'accès aux données et ainsi de la rendre beaucoup plus souple et indépendante du reste de l'application. L'objectif étant de garantir l'intégration de nouveaux jeux de données sans avoir recours à une adaptation pour leur exploitation dans *HyperAtlas*.

Par la suite, nous avons développé de nouvelles fonctionnalités proposant une plus grande personnalisation des cartes. Depuis la réalisation de ces évolutions il est désormais possible pour un utilisateur de personnaliser l'affichage : choix des contours des unités territoriales, leur couleur et leur épaisseur.
Il peut également bénéficier de la notion de transparence apportée sur les disques des cartes à disques proportionnels. Enfin, il est désormais possible pour un utilisateur d'éditer les seuils qu'il souhaite utiliser pour le calcul des déviations entre unités.
Parallèlement à cela des développements ont été conduits pour répondre à une démarche plus expérimentale sur les possibilités offertes par l'application. Ainsi, les membres du projet ont adapté une version de l'outil rendant possible la modification de l'arbre de composition des unités territoriales. Dés lors, la mise en évidence de l'impact de la composition territoriale sur l'étude d'indicateurs et de leurs déviations est alors possible.
Ces améliorations, ont d'ailleurs permis aux membres du projet de présenter *HyperAtlas* lors d'une conférence données au cameroun. Lors de cette conférence, les équipes du laboratoire Géographie-cités ont pu intégrer des nouvelles données, fournies par des intervenants externes, sur la ville de Yaoundé.

Il a ensuite été décidé de continuer dans cette voie et de développer un nouvel outil capable de créer de manière conviviale des jeux de données, de les enrichir et de les administrer. Le projet *HyperAdmin* vise à répondre à ces objectifs.
Dans ce projet ma contribution a consisté à concevoir une première version de cet outil permettant d'importer des données brutes et d'exporter des jeux de données exploitables par *HyperAtlas*.
Le processus nommé 2TUP pour « 2 Track Unified Process », a été employé pour guider l'analyse et la conception de cet outil. De surcroît, nous avons souhaité que le format des données traitées par notre application soit conforme aux standards proposés par l'Open Geospatial Consortium (OGC). Le volume présumé des données a également guidé le choix du mode de stockage de celle-ci vers une base de données pour des raisons d'efficacité et de performance. Ainsi une étude comparative entre les différents éditeurs proposant des extensions à leurs bases de données a été menée. Le résultat de cette étude nous a orienté vers l'utilisation de PostgreSQL, une base de donnée libre et de son extension nommée PostGIS dotant la base d'un support de données spatiales répondant aux normes proposées par l'OGC.
Ces étapes ont finalement abouti à la réalisation d'une version *bêta* de l'application permettant la lecture de fichiers de données, leur exploitation (calcul des contours, gestion de la hiérarchie, …) et enfin le stockage du résultat dans une base de données PostgreSQL. Cette version permet également la génération de fichiers de données exploitables par l'application *HyperAtlas*.

5.3 Perspectives

Les perspectives d'évolution et d'amélioration sont nombreuses pour les deux projets parmi lesquelles nous pouvons distinguer des évolutions techniques et des évolutions fonctionnelles.

Pour *HyperAtlas*, par exemple, il est possible d'étudier des déviations locales qui sont les rapports entre des unités territoriales et leurs voisines. Pour l'instant cette notion de voisinage se limite au calcul de contiguïté des limites administratives de ces unités. Mais il parait tout à fait pertinent de proposer d'étudier d'autre type de voisinage. Par exemple, comparer une unité aux autres unités qui sont à moins de 200Km, ou pourquoi pas à moins de deux heures de trajet selon un type de transport défini.
Nos partenaires de Géographie-cités et des équipes PARIS disposent de ce genre d'informations. Dés lors, il leur est possible de nous fournir des matrices de voisinage.

Outre ces aspects de personnalisation de l'une des cartes offerte par l'outil il semble prometteur de s'intéresser à l'ajout de nouvelles cartes. Nous pourrions ainsi voir les cartes comme des « plugins » qu'il serait possible de charger dans l'application.

Parmi toutes les évolutions possibles, la plus prioritaire consiste à développer un module d'analyse spatiale qui permettrait de s'affranchir des maillages territoriaux. Celui-ci utiliserait des fonctions mathématiques pour proposer des cartes lissées à l'utilisateur. Pour cela, les stocks ne sont plus attribués à des unités territoriales mais à des points de l'espace, chaque point influençant ses voisins.
Pour mieux comprendre ce concept, prenons un exemple similaire à celui de la gravité, chaque point est porteur d'une masse et influence ses voisins. Plus deux points sont proches et plus ils s'attirent mais au-delà d'une certaine distance leur interaction devient quasi nulle.
Avec ce point de vue il est possible de rendre quasi continu des phénomènes considérés auparavant comme discrets.

Ces évolutions fonctionnelles d'*HyperAtlas* trouvent également leur pendant dans *HyperAdmin.* Ainsi, *HyperAdmin* ne se contentera pas de gérer des données mais de devenir un véritable outil de personnalisation d'*HyperAtlas*, en générant soit des jeux de donnée, soit de nouvelle cartes utilisables dans *HyperAtlas*.

Outre ces aspects fonctionnels, des améliorations techniques sont encore possibles. Par exemple, il est envisageable qu'*HyperAdmin* utilise des outils de correspondance (*mapping*) Objet/Relationnel, de type « Hibernate », pour accéder à une ou plusieurs bases de données. Ces outils permettant l'utilisation de caches devraient augmenter la rapidité d'accès aux données.
L'utilisation de bases de données XML peut être également un plus pour la gestion des méta-données que l'on souhaite associer aux stocks.

En conclusion, le projet *HyperCarte* et les outils qui en sont issus sont ouvert à de nombreuses évolutions techniques et fonctionnelles qui permettent de pousser plus loin les analyses faites dans les sciences sociales.

5.4 Bilan personnel

Personnellement, ce stage m'a beaucoup apporté. D'un point de vue technique il m'a permis de faire évoluer mes compétences dans le domaine du génie logiciel. J'ai ainsi pu m'orienter, comme je le souhaitais vers des postes d'ingénieur d'étude ou même de réfé-

rent technique JAVA alors que cela était difficilement envisageable avec mon parcours axé d'avantage vers l'administration système et réseau que sur le développement de logiciel. Ces connaissances ont pu être mises en pratique dès la fin du stage.

D'un point de vue humain il m'aura permis de côtoyer le milieu de la recherche pendant une année et de travailler avec des personnes compétentes qui m'ont beaucoup appris et qui ont développé mon intérêt pour la géomatique et la géostatistique.

ANNEXE 1

LA SIGNATURE NUMÉRIQUE

Cette annexe a pour objectif de présenter au lecteur les principes de bases de la signature électronique. Il ne s'agit pas de faire une description détaillée des différents algorithmes de chiffrage mais plutôt de faire un récapitulatif de ce qu'est la signature électronique et de la manière dont elle fonctionne.

Principe la signature électronique

De la même manière qu'il est nécessaire de signer certains documents papiers, il est devenu rapidement indispensable de pouvoir s'assurer de la provenance d'un document électronique et à plus forte raison pour un programme que l'on souhaite exécuter sur son ordinateur.
Ainsi pour garantir l'authenticité d'un programme ou d'un document électronique il est nécessaire de pouvoir personnaliser ce dernier de manière unique et inimitable comme une signature manuscrite personnalise un document papier. Cette personnalisation utilise un mécanisme de cryptage dit asymétrique. Ce cryptage ou chiffrage est dit asymétrique car il n'utilise pas les mêmes clés pour crypter et pour décrypter les données, une personne peut alors crypter un document ou un programme en utilisant une clé qu'il ne divulguera pas (clé privée) et le destinataire pourra le déchiffrer avec une autre (clé publique).

Par exemple imaginons que M. Dupond souhaite envoyer un document à M. Durand via un réseau informatique non sécurisé.
M. Dupond souhaite que ce document parvienne à M. Durand sans qu'il ne subisse aucune modification où altération. M. Durand lui, souhaite s'assuré que le document qui lui est adressé provienne véritablement de M. Dupond et de personne d'autre.

Pour parvenir au résultat escompter M. Martin va chiffrer le programme en utilisant sa clé privée qu'il est le seul à posséder. Puis il envoie le document à M. Durand qui lui possède la clé publique de M. Martin. A la réception du message deux cas sont alors possibles soit M. Durand peut déchiffrer le document et auquel cas il sais que ce document est bien celui envoyé par M. Martin. Soit il ne peut le déchiffrer ce qui indique que le document est soit incomplet ou a été modifié soit qu'il ne provient pas de M. Martin.

Les certificats

Le principe de chiffrement asymétrique décrit précédemment présente cependant une faiblesse : il est nécessaire que la clé publique possédée par le destinataire des données soit bien celle de l'émetteur et non celle d'un usurpateur. C'est pour palier à cette faiblesse que la notion de certificats à été implémenter.

Les certificats sont de petits fichiers contenant d'une part des informations sur l'émetteur et d'autre part leur validation par une autorité de certification.
Les autorités de certification sont des organismes dont le rôle est de délivrer les certificats, de leur assigner une date de validité. Ces autorités définissent des politiques de certifications en fonction du mode d'enregistrement et de l'usage du certificat. Elle distingue alors des classes de certificats et elle définit en même temps les conditions et le niveau de sa responsabilité. Ainsi selon la classe de certificat choisie des contrôles plus ou moins poussés sont effectués sur l'identité du demandeur du certificat.

ANNEXE 2

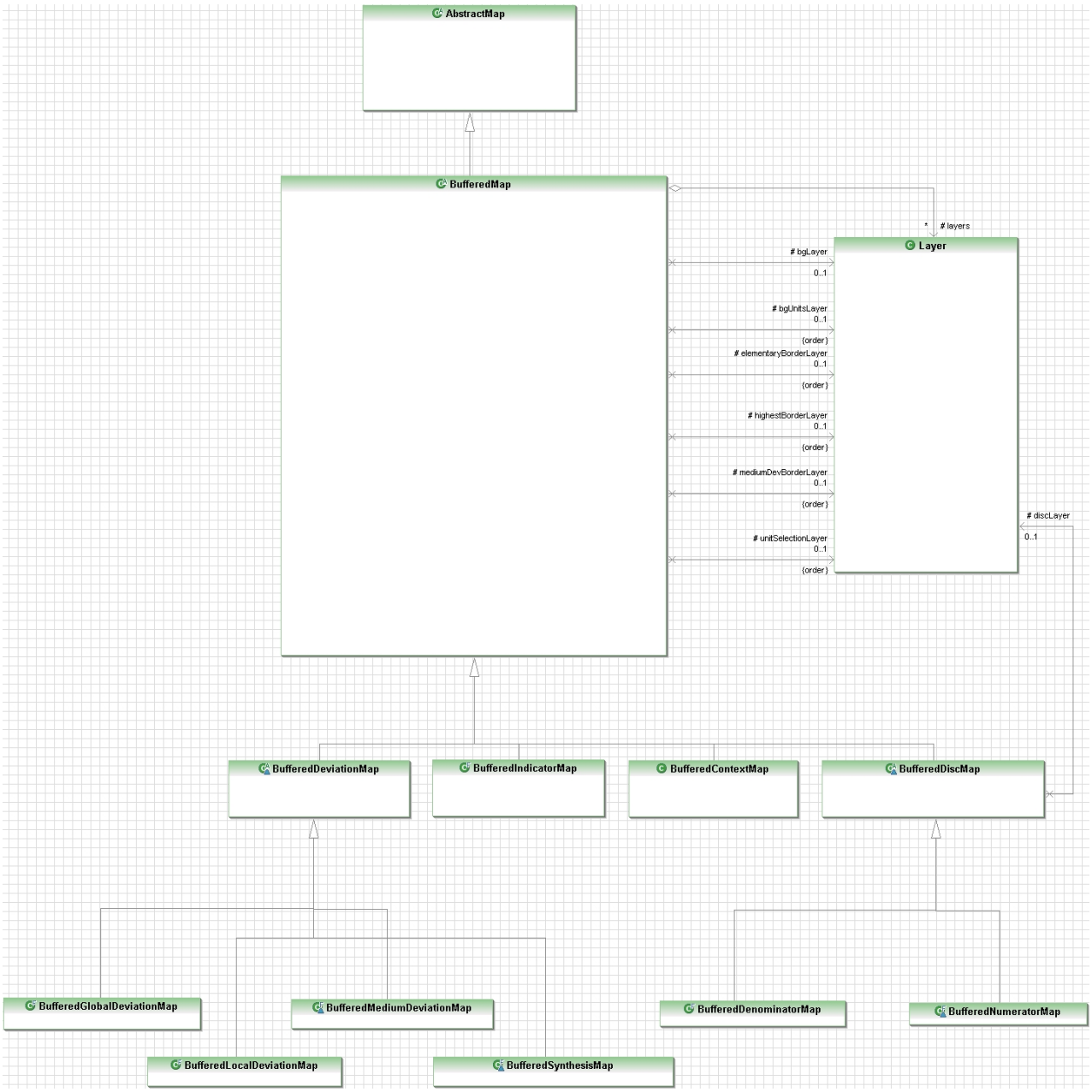
EXEMPLES D'ÉLÉMENTS DE LA PARTIE PRÉSENTATION DES APPLICATIONS HYPERATLAS ET HYPERADMIN

Cette annexe propose des exemples de codes sources utilisés dans la partie interface graphique commune aux deux applications *HyperAdmin* et *HyperAtlas*.

Elle présente d'abord un diagramme de classe présentant une partie de l'architecture de l'affichage des cartes.
Elle détaille par la suite, les méthodes de bases permettant de dessiner les unités territoriales dans les différentes cartes.
Puis, elle décrit les deux méthodes « highlightUnitDisplay » et « restoreUnitDisplayay » nouvellement implémentée pour la mise en valeur des unités territoriales.
Ensuite, elle fournie le détails des méthodes développées pour fournir un nouveau support d'affichage de cartes aptes à afficher n'importe quelle carte.
Enfin, elle détaille les méthodes d'affichage des cartes utilisant les méthodes venant d'être présentées.

1. Diagramme des classes utilisées pour la représentation en couches

Ce diagramme de classes représente les classes mises en œuvre pour dessiner les cartes des applications *HyperAtlas* et *HyperAdmin*.



2. Extrait des méthodes de la classe Map

Cet extrait de code montre les méthodes « drawUnit » et « drawUnits » permettant pour l'une de dessiner une unité territoriale (objet de type HCUnit) et pour l'autre de dessiner l'ensemble des unités contenues dans une liste.

```
abstract class BufferedMap extends AbstractMap
    implements IGlobalEventListener, IIndexedEventListener, ActionListener {


                                                ...
    /**
     * Draw borders and background of a specified unit
     *
     * @param g :
     *            composant graphique pour affichage
     * @param unit :
     *            unit to draw
     * @param borderColor
     * @param backgroundColor
     */
    protected void drawUnit(Graphics2D g, HCUnit unit, Color borderColor,
                                    Color backgroundColor) {
        try {
            java.awt.geom.Area frontiere = unit.getArea();
            g.setColor(backgroundColor);
            g.fill(frontiere);
            g.setColor(borderColor);
            g.draw(frontiere);
        } catch (NullPointerException nEx) {}
    }
}

                                                ...
    /**
     * Affichage des couleurs de frontière et surface des unites
     *
     * @param g :
     *            composant graphique pour affichage
     * @param listUnits :
     *            liste des unités à afficher
     * @param colorBorder :
     *            Couleur pour les contours
     * @param colorSurface :
     *            Couleur pour la surface
     */
    protected void drawUnits(Graphics2D g, Vector listUnits, Color colorBorder,
            Color colorSurface) {
        for (Enumeration e = listUnits.elements(); e.hasMoreElements();) {
            drawUnit(g, (HCUnit) e.nextElement(), colorBorder, colorSurface);
        }
    }

                                                ...
}
```

3. Détail des méthodes de mise en valeur des unités survolées par la souris

Cette partie présente les deux méthodes permettant de mettre en valeur les unités territoriales survolées avec la souris. L'autre servant à rétablir l'affichage original de l'unité.

Détail de la méthode highlightUnitDisplay de la classe DeviationMap

```
/**
 * Highlight the display of the given Unit.
 * @param unit The Unit to be highlighted.
 */

protected void highlightUnitDisplay(HCUnit unit){

    this.g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_OFF);

    this.drawUnitBorder(this.g, unit, Color.red);

    this.g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);
}
```

Détail de la méthode restoreUnitDisplayay de la classe DeviationMap

```
/**
 * Restore the basic display of the given Unit.
 *
 * @param unit
 *            Unit to be repainted.
 * @see hypercarte.ui.Map#restoreUnitDisplay(HCUnit)
 */
protected void restoreUnitDisplay(HCUnit unit) {

    SimplePaletts paletts = (SimplePaletts) Settings.getInstance().getMap(
            this.mapIndex).getPaletts();
    Color color = Color.gray;
    try {
        switch (mapIndex) {
        case Settings.MAP_INDICATOR:
            color = paletts.getMatchingColor(unit.getRatio());
            break;
        case Settings.MAP_GLOBAL_DEVIATION:
            color = paletts.getMatchingColor(unit.getGlobalDev());
            break;
        case Settings.MAP_MEDIUM_DEVIATION:
            color = paletts.getMatchingColor(unit.getMediumDev());
            break;
        case Settings.MAP_LOCAL_DEVIATION:
            color = paletts.getMatchingColor(unit.getLocalDev());
            break;
        }
    } catch (Exception e) {e.printStackTrace();}

    this.g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_OFF);
    this.drawUnitBorder(this.g, unit, color);
    this.g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

    if (Settings.getInstance().isDrawElementaryUnitsBorder())
        this.drawUnitBorder(this.g, unit, Settings.getInstance()
                .getElementaryBorderColor());

    if (Settings.getInstance().isDrawMediumBorder())
        this.drawMediumZoningBorders(super.g, Settings.getInstance()
                .getMediumDevColor());
```

```
    if (Settings.getInstance().isDrawCountryBorder()) {

        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_OFF);
        drawUnitsBorder(g, HCUnitRepository.getInstance()
                .getHighestZoningUnits(), Settings.getInstance()
                .getBorderColor());
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
    }

    if (Map.unit2ZoomOn != null) {
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_OFF);
        this.g.setStroke(new BasicStroke((float) ((Settings.getInstance()
                .getThickness() + 0.3) / Settings.getInstance()
                .getZoomOffset())));
        this.drawUnitBorder(g, Map.unit2ZoomOn, Color.red);
        this.g.setStroke(new BasicStroke((float) (Settings.getInstance()
                .getThickness())));
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_OFF);
    }
}
```

4. Détail des méthodes permettant un affichage générique des cartes

Les méthodes présentées par la suite, permettent son utilisée pour calculée le rapport entre les coordonnées géographique des unités territoriales et les coordonnées écran.
La méthode « initMapBorder » permet de définir le rectangle englobant de la carte, la méthode « recalculateDisplayBorder » permet de calculer les coordonnées du panneau affichant la carte dans le système de coordonnée de cette dernière. Enfin, la méthode « calculateZoomOffset » calcul le coefficient multiplicateur permettant de passer du système de coordonnée géographique au système de coordonnées écran.

Détail de la méthode initMapBorder

```
/**
 * Get map Bounds.
 */
protected void initMapBorder() {

    if (AbstractMap.mapBorder == null && Settings.getInput() != null) {
        try {
            AbstractMap.mapBorder = Settings.getInput().getMapBounds(
                Settings.getInstance().getProjectID()
            );
        } catch (InvalidProjectException e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
```

Détail de la méthode recalculateDisplayBorder

```
/**
 * Recalculate the display border coordinate in map system of coordinate.
 */
protected void recalculateDisplayBorder() {

    AbstractMap.displayBorder = new Rectangle(
        this.getChangeCoordinated(this.getBounds().getLocation()),
            new Dimension(
                    (int) ((this.getBounds().getHeight()) * 1 / (Settings
                            .getInstance().getZoomOffset() * Settings
                            .getInstance().getScale())), (int) ((this
                            .getBounds().getWidth()) * 1 / (Settings
                            .getInstance().getZoomOffset() * Settings
                            .getInstance().getScale()))));

    AbstractMap.displayBorder.translate(
            -(int) (this.getBounds().getHeight() / (Settings.getInstance()
                    .getZoomOffset() * Settings.getInstance().getScale())),
            0);

    Dispatcher.getInstance().dispatchEvent(
        new IndexedEvent(
                IndexedEvent.DISPLAY_EVENT__SCALE_UPDATE_REQUIRED,
                this.mapIndex)
    );
}
```

Détail de la méthode calculateZoomOffset

```
/**
 * Calculate a zoom offset to formating (scale) map in order to correctly
 * fit panel with the map.
 */
protected void calculateZoomOffset(){

    if(AbstractMap.mapBorder != null & AbstractMap.displayBorder != null){
        double maxDisplayBorderWidth  = this.getBounds().getHeight();
        double maxDisplayBorderHeight = this.getBounds().getWidth();
        double displayBorderWidth     = this.getBounds().getHeight();
        double displayBorderHeight    = this.getBounds().getWidth();

        double zoomNumber,zoomNumberMax;
        if(!Settings.getInstance().isToolbarDisplayed()){
            //Non Dynamique à changer !!!
            maxDisplayBorderWidth = maxDisplayBorderWidth-32;
        }
        if(!Settings.getInstance().isParameterPanelDisplayed()){
            maxDisplayBorderWidth = maxDisplayBorderWidth-HyperCarte.MAX_HEIGHT;
        }
        if(Settings.getInstance().isMapPanelExpanded()){
            maxDisplayBorderHeight = maxDisplayBorderHeight - Frameset.MAX_WIDTH;
        }

    if(Settings.getInstance().getCustomer().equals(Settings.CUSTOMER_ESPON)){
            zoomNumberMax = Math.min(maxDisplayBorderWidth
                    / AbstractMap.mapBorder.getWidth(),
                    (maxDisplayBorderHeight+20)
                            / AbstractMap.mapBorder.getHeight());
        }else{
            zoomNumberMax = Math.min(maxDisplayBorderWidth
                    / AbstractMap.mapBorder.getWidth(),
                    maxDisplayBorderHeight
                            / AbstractMap.mapBorder.getHeight());
        }

        if(Settings.getInstance().getCustomer().equals(Settings.CUSTOMER_ESPON)){
            zoomNumber = Math.max(displayBorderWidth
                    / AbstractMap.mapBorder.getWidth(),
                    (displayBorderHeight+20)
                            / AbstractMap.mapBorder.getHeight());
        }else{
            zoomNumber = Math.max(displayBorderWidth
                    / AbstractMap.mapBorder.getWidth(),
                    (displayBorderHeight)
                            / AbstractMap.mapBorder.getHeight());
        }


        zoomNumber = Math.min(zoomNumber, zoomNumberMax);
        zoomNumber = (Math.abs(zoomNumber));
        Settings.getInstance().setZoomOffset(zoomNumber);
        try {
            Settings.getInstance().setScaleMaxValue(
                (float) HyperCarte.getHCInput().getMapDistanceConversion(
                    Settings.getInstance().getMajorStudyAreaID(),
                    100/zoomNumber
                )
            );
            Settings.getInstance().setScaleUnit(
                HyperCarte.getHCInput().getDistanceUnit(
                        Settings.getInstance().getMajorStudyAreaID()
                    )
                );
        } catch (InvalidMajorStudyAreaException e) {
            e.printStackTrace();
        }
    }
}
```

5. Détails des méthodes « paintMap » et « paintComponents »

```
/**
 * Paint the current map panel
 *
 * @param graphics
 */
public void paintComponent(Graphics graphics) {

    super.paintComponent(graphics);

    if (Settings.getInput() != null) {
        long startTime = System.currentTimeMillis();

        //We test that the buffer image size is the same as
        //panel size.
        if (bi == null || bi.getWidth() != getWidth()
                || bi.getHeight() != getHeight()) {
            this.calculateZoomOffset();
            this.bi = new BufferedImage(getWidth(), getHeight(),
                    BufferedImage.TYPE_INT_ARGB);
            for (int i = 0; i < this.layers.size(); i++) {
                ((Layer) this.layers.elementAt(i)).setSize(getWidth(),
                        getHeight());
                ((Layer) this.layers.elementAt(i)).setUpToDate(false);
            }
            biUpToDate = false;
        }

        if (!biUpToDate) {

            paintMap(graphics);
            Graphics2D big2D = this.bi.createGraphics();
            big2D.setColor(this.getBackground());
            big2D.fillRect(0, 0, getWidth(), getHeight());

            for (int i = 0; i < this.layers.size(); i++) {
                ((Layer) this.layers.elementAt(i)).paint(big2D);
            }
            biUpToDate = true;
        }
        graphics.drawImage(this.bi, 0, 0, this);
    }
}
```

```
protected void paintMap(Graphics graphics) {

    //Test if it's the first time display.
    if (firstDisplay) {
        initLayers();
    }

    if (AbstractMap.mapBorder == null || AbstractMap.displayBorder == null) {
        this.initMap();
        this.centerMap();
    }
    this.recalculateDisplayBorder();

    if (AbstractMap.mapBorder != null && AbstractMap.displayBorder != null) {
        // We use the first test for optimisation purpose in order
        // to not instantiate and execute
        // the same code to time one for centering horizontaly and
        // another to centering verticaly.
        if (AbstractMap.mapBorder.width < AbstractMap.displayBorder.width
                & AbstractMap.mapBorder.height < AbstractMap.displayBorder.height) {
            this.centerMap();
        } else if (AbstractMap.mapBorder.width < AbstractMap.displayBorder.width) {
            this.centerMapYAxis();
        } else if (AbstractMap.mapBorder.height < AbstractMap.displayBorder.height) {
            this.centerMapXAxis();
        }
    }
```

```
    // Initalize rendering
    this.g = (Graphics2D) graphics;
    if (!BufferedMap.bgUnitsLayer.isUpToDate()) {
        updateBgUnitsLayer();
    }

    // Draw study area at elementary zoning level
    if (Settings.getInstance().isDrawElementaryUnitsBorder()) {
        BufferedMap.elementaryBorderLayer.setDisplayed(true);
        if (!BufferedMap.elementaryBorderLayer.isUpToDate()) {
            updateElementaryUnitsBorder();
        }
    } else {
        BufferedMap.elementaryBorderLayer.setDisplayed(false);
    }

    // Draw the units of the medium and highest zoning level that
    // belong to the study area
    if (Settings.getInstance().isDrawMediumBorder()) {
        BufferedMap.mediumDevBorderLayer.setDisplayed(true);
        if (!BufferedMap.mediumDevBorderLayer.isUpToDate()) {
            updateMediumDevBorderLayer();
        }
    } else {
        BufferedMap.mediumDevBorderLayer.setDisplayed(false);
    }

    // Draw the units of the highest zoning level that belong to the
    // study area
    if (Settings.getInstance().isDrawCountryBorder()) {
        BufferedMap.highestBorderLayer.setDisplayed(true);
        if (!BufferedMap.highestBorderLayer.isUpToDate()) {
            updateHighestBorderLayer();
        }

    } else {
        BufferedMap.highestBorderLayer.setDisplayed(false);
    }

    if (BufferedMap.unit2ZoomOn != null) {
        BufferedMap.unitSelectionLayer.setDisplayed(true);
        if (!BufferedMap.unitSelectionLayer.isUpToDate()) {
            updateUnitSelectionLayer();
        }
    } else {
        BufferedMap.unitSelectionLayer.setDisplayed(false);
    }
}
```

ANNEXE 3

EXEMPLE DU CONTENU DES « FICHIERS STRUCTURES » UTILISÉS PAR HYPERADMIN

<table>
<tr><th></th><th>Fichier Excel : liste des onglets</th><th>Fichiers textes</th></tr>
<tr><td>Unit</td><td>

UT_ID
AT11
AT12
AT13

</td><td>

```
UT_ID
AT11
AT12
AT13
```

</td></tr>
<tr><td>Area</td><td>

Area_ID
UE15
Arc_Atlantique
Candidat_UE
UE29
UE25+2

</td><td>

```
Area_ID
UE15
Arc_Atlantique
Candidat_UE
UE29
UE25+2
```

</td></tr>
<tr><td>unit-sup</td><td>

UT_ID	UTSup_ID
AT1	AT
AT2	AT
BE1	BE
BE2	BE

</td><td>

```
UT_ID      UTSup_ID
AT1        AT
AT2        AT
BE1        BE
BE2        BE
```

</td></tr>
<tr><td>Unit-Area</td><td>

UT_ID	Area_ID
AT	UE15
BE	UE15
DE	UE15
ES	UE15

</td><td>

```
UT_ID      Area_ID
AT         UE15
BE         UE15
DE         UE15
ES         UE15
```

</td></tr>
<tr><td>Unit-Zoning</td><td>

UT_ID	Zoning_ID
AT	Nuts_0
BE	Nuts_0
CH	Nuts_0
CZ	Nuts_0

</td><td>

```
UT_ID      Zoning_ID
AT         Nuts_0
BE         Nuts_0
CH         Nuts_0
CZ         Nuts_0
```

</td></tr>
<tr><td>Unit-Language</td><td>

UT_ID	Language_ID	UT_NAME
AT11	DE	BURGENLAND
AT13	DE	WIEN
AT21	DE	KAERNTEN
AT22	DE	STEIERMARK

</td><td>

```
UT_ID      Language_ID
           UT_NAME
AT11       DE         BURGENLAND
AT13       DE         WIEN
AT21       DE         KAERNTEN
AT22       DE         STEIERMARK
```

</td></tr>
<tr><td>Area-Language</td><td>

Area_ID	Language_ID	Area_NAME
UE15	FR	Union européenne des 15
UE25	FR	Union européenne des 25
PECO	FR	Pays d'Europe Centrale et Orientale

</td><td>

```
Area_ID    Language_ID
           Area_NAME
UE15       FR         Europe  des
15
UE25       FR         Europe  des
des 25
PECO       FR         Europe
```

</td></tr>
</table>

<table>
<tr><td></td><td></td><td>Centrale et Orientale</td></tr>
<tr><td>Zoning-Language</td><td>
<table>
<tr><th>Zoning_ID</th><th>Language_ID</th><th>Zoning_NAME</th></tr>
<tr><td>Nuts_0</td><td>FR</td><td>Nomenclature des unités territoriales de niveau 0</td></tr>
<tr><td>Nuts_1</td><td>FR</td><td>Nomenclature des unités territoriales de niveau 1</td></tr>
<tr><td>Nuts_2</td><td>FR</td><td>Nomenclature des unités territoriales de niveau 2</td></tr>
</table>
</td><td>
<pre>
Zoning_ID   Language_ID Zoning_NAME
Nuts_0   FR   Nomenclature
des unités territoriales de niveau 0
Nuts_1   FR   Nomenclature
des unités territoriales de niveau 1
Nuts_2   FR   Nomenclature
des unités territoriales de niveau 2
Nuts_3   FR   Nomenclature
des unités territoriales de niveau 3
</pre>
</td></tr>
</table>

BIBLIOGRAPHIE

[@BFS] Site Web du bureau fédéral de la statistique Suisse, accessible à l'adresse : http://www.bfs.admin.ch, consulté le 14/01/2007.

[Bissler, 04] Thierry Bissler, *Conception et développement d'une plate-forme pour la génération de Systèmes d'Information Spatio-Temporelle et Multimédia dédiés aux Risques Naturels*, Mémoire d'ingénieur CNAM soutenu le 6 juillet 2004.

[Bloch, 02] Joshua Bloch, Java Efficace, ISBN : 2-7117-4805-7, Addison Wesley.

[Booch et al., 99] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wwesley,1999

[@CASS] Interrogation et indexation spatiales - Optimisation des transferts, disponible sur le site du « Groupement De Recherche SIGMA dit Cassini » à l'adresse : http://cassini.univ-lr.fr/Pages/Journee/Nantes/Presentation/SalviusLaucius.pdf, consulté le 14/01/2007.

[@CCM] Article sur la signature électronique disponible sur le site « Comment ça marche », à l'adresse : http://www.commentcamarche.net/crypto/signature.php3, consulté le 14/01/2007.

[Cuenot, 05] Olivier Cuenot, *Modélisation spatiale multiscalaire de phénomènes sociaux*, Mémoire d'ingénieur CNAM soutenu le 31 mars 2005.

[Denègre et al., 96] Jean Denègre et François Salgé, *Les Systèmes d'information géographique*, collection Que sais-je ?, ISBN : 2 13 047932 4, Presses Universitaires de France, 1996.

[@EPFL] Support de cours de base de données disponible sur le site du Laboratoire de Base de Données de l'EPFL : http://lbdwww.epfl.ch/e/teaching/SlidesBDA/, consulté le 14/01/2007.

[@ESPON] Site Web de l' « European Spatial Planning Observation Network », accessible à l'adresse : http://www.espon.lu/, consulté le 14/01/2007.

[Gamma et al., 99] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design patterns. Catalogue des modèles de conception réutilisables*, ISBN: 2-7117-8644-7, Vuibert, 1999.

[@GART] Market Share: Relational Database Management Systems by Operating System, Worldwide, 2005, disponible à l'adresse :

http://www.gartner.com/DisplayDocument?id=492469, consulté le 14/01/2007.

[@*GEOCLIP*] Site Web dédié a l'outil *GeoClip* produit par la société E=MC3, accessible à l'adresse : http://www.*GeoClip*.fr/fr/, consulté le 14/01/2007.

[@GPL, 91] GNU Project – Free Software Foundation, 1991. *GNU General Public License, Version 2*, juin 1991. Disponible en ligne : http://www.-gnu.org/licenses/gpl.html, consulté le 14/01/2007.

[@HYP] Site Web du projet HyperCarte, accessible à l'adresse : http://www-lsr.imag.fr/HyperCarte/, consulté le 14/01/2007.

[@JARG] Définition des arbre équilibré (BTree) accessible sur le site le « Jargon Français » accessible à l'adresse : http://www.linux-france.org/prj/jargonf/B/B-tree.html, consulté le 14/01/2007.

[@JTUT] Le « JAVA Tutorial » disponible à l'adresse : http://java.sun.com/docs/books/tutorial/, consulté le 14/01/2007.

[@MIF] Description du format d'échange MIF/MID de MapInfo

[@MRNF] Site du ministère québécois des Ressources Naturelles et de la Faune, accessible en ligne à l'adresse : http://www.mrn.gouv.qc.ca/accueil.jsp, consulté le 14/01/2007.

[Muller et al., 00] Pierre-Alain Muller, Nathalie Gaertner, *Modélisation objet avec UML*, ISBN : 2-212-09122-2, Eyrolles, 2000.

[Nanci et al., 92] Dominique Nanci, Bernard Espinasse,Bernard Cohen, Henri Heckenroth, Jean-Claude Asselborn, *Ingénierie des systèmes d'information : Merise, Deuxième génération*, ISBN : 2-7361-0747-7, Sybex, 1992

[@NUTS] Définition de la nomenclature des unités territoriales statistiques disponible sur le portail Web de la communauté Européenne à l'adresse : http://europa.eu.int/comm/eurostat/ramon/nuts/basicnuts_regions_fr.html, consulté le 14/01/2007.

[@OGC] Site Web de l'Open Geospatial Consortium, accessible à l'adresse : http://www.opengeospatial.org/, consulté le 14/01/2007.

[@OSI] Site Web de l'Open Source Initiative (OSI), accessible à l'adresse : http://www.opensource.org/, consulté le 14/01/2007.

[@ORA] Site Web de l'éditeur Oracle : http://www.oracle.com, consulté le 14/01/2007.

[@OSP] Ressources Internet sur l'extension « Oracle Spatial », accessible à l'adresse : http://www.oracle.com/technology/products/spatial/index.html, consulté le 14/01/2007.

[@PGEO] Site de Philippe Warniez, concepteur de l'outil *PhilCarto* disponible à l'adresse : http://philgeo.club.fr, consulté le 14/01/2007.

[@RER] Site Internet de l'organisation « Refractions Research Inc » : http://www.refractions.net/, consulté le 14/01/2007.

[Roques, 2003] Pascal Roques, Franck Vallée, UML en Action, de l'analyse des besoins à la conception en Java, ISBN : 2-21211213-0, Eyrolles, 2003.

[@SEIG] Le serveur éducatif de l'IGN et de l'Education Nationale sur l'information géographique, accessible à l'adresse : http://seig.ensg.ign.fr/, consulté le 14/01/2007.

[@SSI] Le site du CFSSI « Centre de formation à la sécurité des systèmes d'information », accessible à l'adresse : www.formation.ssi.gouv.fr/autoformation/signature.html, consulté le 14/01/2007.

[@STCAN] Site Web « Statistique Canada » un institut similaire à l'INSEE, accessible à l'adresse : http://www.statcan.ca, consulté le 14/01/2007.

MEMOIRE D'INGENIEUR C.N.A.M. EN INFORMATIQUE

HYPERATLAS ET *HYPERADMIN* : DES OUTILS CARTOGRAPHIQUES POUR L'ANALYSE DE PHÉNOMÈNES SOCIAUX

CHRISTOPHE CHABERT **GRENOBLE, LE 29/03/2007**

RÉSUMÉ

A l'heure où l'informatique et le multimédia s'imposent dans de plus en plus de domaines, la cartographie ne déroge pas à cette règle. Les Systèmes d'Information Géographique, véritables outils cartographiques, permettent d'accéder, de visualiser et de synthétiser une grande quantité d'informations.

Le projet *HyperCarte* a ainsi vu le jour pour répondre aux nouvelles problématiques de l'étude des phénomènes sociaux se produisant à de multiples échelles : communales, départementales, régionales, nationales et même parfois continentales. Ce projet a conduit au développement d'outils informatiques propres à répondre à ces besoins. La réingénierie du logiciel *HyperAtlas* permet une génération plus générique de cartes multiscalaires à la demande. Le développement du nouveau logiciel *HyperAdmin* permet la création et la gestion des données exploitables par *HyperAtlas*.

Mots-clés : Système d'information géographique, Cartographie, Analyse territoriale multiscalaire, OGC, Base de données, Interface graphique utilisateur, Java, Modélisation objet.

SUMMARY

At the present time, data processing and the multimedia are essential in more and more fields, the cartography does not go against this rule. The Geographical Information Systems (GIS), genuine cartographic tools, make it possible to reach, visualize and synthesize a great quantity of information.

The HyperCarte project was born in seen to answer to the new problems of the study of the social phenomena occurring on multiple scales: district wide, departmental wide, regional wide, nationwide and even sometimes continental wide.
This project led to the development of data-processing tools suitable to meet these needs, it is the case of the *HyperAtlas* software allowing a generation of multiscalaire

maps and *HyperAdmin* software allowing the creation and the management of *Hyper-Atlas* data.

Keywords: Geographical Information System, cartography, multiscalaire territorial analysis, OGC, database, graphical user interface, Java, object modelling.