BROUILLON

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

présenté par Olivier Cuenot

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en INFORMATIQUE

Modélisation spatiale multiscalaire de phénomènes sociaux : Réalisation du logiciel Hypercarte

Soutenu le XX

JURY

Présidente : Mme Véronique Donzeau-Gouge

Membres : M. Eric Gressier
M. Jacques Courtin
M. Jean-Pierre Giraudin
M. Jérôme Gensel
M. Claude Grasland

Remerciements

Table des matières

Chapitre 1 – Introduction

A l'heure où l'informatique en général et le multimédia en particulier s'imposent dans de plus en plus de domaines, la cartographie ne déroge pas à cette règle. Les Systèmes d'Information Géographique (SIG[1]), véritables outils cartographiques, permettent d'accéder à une grande quantité d'information. Devant les grandes possibilités de traitement et les multiples représentations qu'offrent les SIG, la conception de représentations cartographiques des statistiques disponibles semble plus prometteuse que l'hypothétique collecte de bases de données étendues mais soumises à des traitements classiques.

1.1. Contexte

Le projet de recherche Hypercarte s'inscrit dans ce cadre. En implémentant un outil montrant une multiple représentation de la distribution d'un phénomène social, le but est de valider les concepts mis au point par les géographes et statisticiens du projet.

1.1.1. Le projet Hypercarte

Hypercarte est un projet de recherche pluridisciplinaire dont l'objectif est de valider la dimension opérationnelle des outils mis au point par des géographes. Hypercarte est également un réseau de recherches.

1.1.1.1. Les partenaires

Le projet Hypercarte est soutenu par plusieurs partenaires : le laboratoire LSR-IMAG, le laboratoire Géographie-Cités et le projet Apache.

1.1.1.1.1. Le laboratoire LSR-IMAG

L'IMAG[2] est une fédération de sept unités de recherche du Centre National de la Recherche Scientifique (CNRS), de l'Institut National Polytechnique de Grenoble (INPG) et de l'Université Joseph Fourier (UJF), implantées sur plusieurs sites de l'agglomération grenobloise.

L'équipe SIGMA[3] du laboratoire LSR[4] de l'IMAG (LSR-IMAG) réalise des recherches appliquées sur l'ingénierie des Systèmes d'information (SI), et plus précisément des SI basés sur le Web. Les deux axes de recherche de l'équipe sont d'une part l'axe 'Composants' travaillant sur la réutilisation et la traçabilité des produits et des processus d'ingénierie des SI,

[1] SIG : outil permettant de stocker, d'analyser et de représenter de données géographiques.
[2] IMAG : Institut d'Informatique et de Mathématiques Appliquées de Grenoble.
[3] SIGMA : Systèmes d'Information : inGénierie et MultimédiA.
[4] LSR : Logiciels, Systèmes et Réseaux.

et d'autre part l'axe 'Multimédia-Web' orienté vers l'ingénierie des SI basés sur des informations multimédia et sur la technologie Web.

Le champ d'investigation des recherches en informatique de l'axe 'Multimédia-Web' porte sur les SI Multimédia basés sur le Web (SIMW), décliné selon trois thématiques :

- la conception et la mise en œuvre de SIMW,
- les données multimédia,
- les SIG.

C'est au sein de cet axe que nous avons participé à la réécriture du logiciel développé par le projet Hypercarte.

1.1.1.1.2. Le laboratoire Géographie-Cités

L'équipe PARIS[5] du laboratoire Géographie-Cités, à travers l'un de ces thèmes de recherche (modélisation et analyse spatio-temporelle en géographie), élabore les outils géographiques mis en œuvre dans le cadre du projet Hypercarte.

1.1.1.1.3. Le projet Apache

Le projet Apache, mené conjointement par le laboratoire Informatique et Distribution de l'IMAG (ID-IMAG) et l'INRIA[6] Rhône-Alpes, a en charge la problématique des traitements sur les données géographiques. A terme, ces traitements seront effectués à la demande du client sur un serveur connecté à une grappe d'ordinateurs (*cluster*).

1.1.1.2. Contexte du projet

Le laboratoire Géographie-Cités et le projet Apache ont développé depuis 1996 des actions de recherche communes sur la modélisation spatiale des phénomènes sociaux et leur restitution cartographique. L'idée centrale du projet Hypercarte est de proposer une méthode d'analyse multiscalaire de la distribution spatiale des phénomènes sociaux permettant d'analyser le même phénomène à des niveaux de généralisation cartographiques différents. Cette méthode cartographique qui bénéficie d'ores et déjà d'une reconnaissance nationale (INSEE, IFEN, DATAR) et internationale (EUROSTAT, Agence Européenne pour l'Environnement) a notamment servi à concevoir les cartes de la distribution mondiale de la population et de la richesse dans le cédérom « 6 milliards d'hommes… et moi »[7].

Parallèlement, l'axe 'Multimédia-Web' possède une forte expérience dans le domaine des SIG pour avoir participé à des projets liés à la cartographie interactive appliquée aux risques naturels :

---

[5] PARIS : Pour l'Avancement des Recherches en Interaction Spatiale.
[6] INRIA : Institut National de Recherche en Informatique et en Automatique.
[7] Ces cartes se sont vues décerner le 1er Prix de Cartographie (catégorie Expert) du Festival International de Géographie de Saint-Dié en septembre 2001.

- le projet SIRVA[8], dont le but est de présenter un recensement des données historiques sur les inondations de l'Arve dans quelques communes de Haute-Savoie (Chamonix, les Houches, Servoz et Vallorcine) ;

- le projet européen SPHERE[9], qui a pour objectif de valoriser l'approche historique pour la prévention des risques naturels et, plus particulièrement, les risques d'inondations dans la moyenne vallée de l'Isère et la basse vallée du Drac ;

- le projet SIDIRA[10], qui permet de visualiser les couloirs des avalanches recensées par l'EPA[11] ayant eu lieu au cours du XX^{ème} siècle sur la commune de Vallorcine, ainsi que leurs caractéristiques.

Ces projets visent à collecter et homogénéiser des informations géographiques, et à proposer des outils de représentation et de consultation de ces informations aux décideurs politiques, aux scientifiques (géographes, historiens) voire au grand public. Au sein du projet Hypercarte, l'équipe SIGMA a en charge la conception et la réalisation de l'outil de représentation et de consultation des données géographiques.

Financé par le Centre National de Recherche Scientifique et lié au projet européen ORATE-ESPON[12], le projet Hypercarte vise à mettre au point de nouveaux concepts et de nouveaux outils d'analyse spatiale des formes sociales. La notion d'hypercartes (multiple représentation de la distribution spatiale d'un phénomène social) repose d'avantage sur une analyse conceptuelle du phénomène étudié que sur une simple exploitation des multiples possibilités de l'outil cartographique. La difficulté la plus importante consiste donc à assurer la mise en adéquation entre :

- des demandes formulées par les utilisateurs à un niveau très général telles que la cohésion territoriale ou les inégalités économiques ;

- des outils d'analyse spatiale d'apparence simple mais reposant sur des hypothèses théoriques très précises (discontinuités, barrières, effets de contexte, etc.) ;

- des données empiriques limitant de façon drastique le domaine de ce qui est réellement mesurable.

La communauté scientifique des chercheurs français et étrangers travaillant sur l'espace européen et son aménagement porte un grand intérêt au projet Hypercarte, dont le but est de simplifier la réalisation de cartes et d'en réduire les coûts.

---

[8] SIRVA : Systèmes d'Information pour les Risques naturels dans la haute Vallée de l'Arve [KHAM, 98] [VILL, 98] [DOUS, 99].

[9] SPHERE : *Systematic Paleoflood and Historical data for the improvEment of flood Risk Estimation* [TROU, 02].

[10] SIDIRA : Système d'Information pour la Diffusion des Risques Avalancheux [ROSA, 02] [ROUS, 02] [ENSE, 03] [TESN, 03].

[11] EPA : Enquête Permanente sur les Avalanches, réalisée par le Cemagref (Institut de Recherche pour l'Ingénierie de l'Agriculture et de l'Environnement).

[12] ORATE : Observatoire en Réseau de l'Aménagement du Territoire Européen.
ESPON : *European Spatial Planning Observation Network.*

1.1.2. Le logiciel Hypercarte

Le nombre de cartes possibles pour décrire un même phénomène étant virtuellement infini, le logiciel Hypercarte n'est pas une simple entité de stockage mais véritablement un outil de conception à la demande, ce qui suppose une triple compétence :

- en géographie et sciences sociales, pour définir les phénomènes potentiellement intéressants, mobiliser les outils d'analyse spatiale les plus innovants et proposer des interprétations précises des phénomènes représentés ;

- en informatique distribuée, pour gérer de façon optimale les flux de demandes, le traitement numérique de grands volumes de données, les caches, pour assurer un temps de réponse rapide ;

- en systèmes d'information et multimédia, pour construire une interface simple d'utilisation, ergonomique et avec des temps d'affichage performants.

Le logiciel répond à des contraintes importantes d'accessibilité, tant d'un point de vue purement technique, que d'un point de vue interaction avec l'utilisateur.

Le module d'analyse territoriale multiscalaire (MTA[13]) vise l'établissement de cartes observant la distribution de l'activité sociale dans les limites d'un ou plusieurs maillages territoriaux supposés pertinents par rapport aux interrogations ou aux phénomènes qui lui sont adressés [GRAS, 03a]. Le module MTA porte sur :

- l'étude des phénomènes d'emboîtement hiérarchiques, déviation d'une unité territoriale par rapport aux maillages de niveaux supérieurs – par exemple la déviation des moyennes régionales par rapport à la moyenne européenne ou nationale ;

- les phénomènes de voisinage territorial – par exemple les discontinuités entre les unités territoriales voisines, en s'affranchissant des frontières nationales.

La version actuelle d'Hypercarte est un prototype qui implémente le module MTA. Ce module met en relation les valeurs attributaires d'unités territoriales avec les moyennes de ces attributs à des niveaux supérieurs – pays ou région d'appartenance, ou à un niveau de voisinage donné – unités territoriales contiguës.

1.2. Problématique

Le prototype Hypercarte intègre les solutions proposées par les différents partenaires en réponse aux problématiques d'analyse multiscalaire de la distribution spatiale. Développé par un stagiaire de DESS [MOIS, 02] puis par un stagiaire du CNAM [MART, 04], il a permis de valider avec succès les outils géographiques conçus par l'équipe PARIS. Il a pour vocation de servir de plate-forme pour le module d'analyse spatiale multiscalaire (MSA[14]) [GRAS, 00],

[13] MTA (*Multiscalar Territorial Analysis*) : analyse territoriale multiscalaire.
[14] MSA (*Multiscalar Spatial Analysis*) : analyse spatiale multiscalaire.

puis éventuellement d'autres modules. Toutefois, son code est peu modulaire et son interface peu performante.

Le logiciel doit être plus évolutif et plus fiable pour servir de base à de futurs développements. Il doit avoir de meilleures performances, c'est-à-dire être plus rapide et plus ergonomique, avant d'être distribué aux utilisateurs cibles.

1.3. Contribution

Nous avons réécrit le logiciel implémentant le module MTA afin de le restructurer, l'homogénéiser et le simplifier, et passer ainsi d'un prototype à une version plus aboutie et distribuable.

Dans le suite du rapport, nous parlons de **version 0.9** pour le prototype – qui est la version du logiciel qui a servi de base à notre travail, et de **version 1.0** pour la version à laquelle nous sommes parvenus.

1.4. Plan du mémoire

Ce document est organisé en cinq chapitres présentant respectivement l'état de l'art, le logiciel, le prototype existant, notre réalisation et enfin, notre conclusion.

Dans le chapitre 2, après une rapide introduction sur les modes de stockage de données graphiques, nous présentons différentes techniques de représentation et des outils de développement, envisageables en matière de cartographie interactive.

Dans le troisième chapitre, nous décrivons le logiciel Hypercarte à travers ses principes généraux et quelques concepts, afin de mieux appréhender son fonctionnement et sa finalité.

Dans le quatrième chapitre, nous procédons à une analyse critique de la version 0.9 d'Hypercarte, en décrivant les techniques utilisées et les choix d'implémentation, en présentant son interface utilisateur. Enfin, nous établissons la liste des points d'amélioration que nous avons identifié concernant les performances, l'évolutivité et l'ergonomie, ainsi que la liste des principaux bogues.

Le cinquième chapitre détaille la réalisation de la version 1.0, des choix liés à la conception et des principes mis en œuvre, aux actions menées pour améliorer les performances et l'ergonomie, en passant par la restructuration du code et l'ajout de fonctionnalités.

Enfin, nous faisons un bilan du travail effectué et des perspectives d'avenir, avant de conclure par un bilan personnel de cette année passée en laboratoire de recherche.

Chapitre 2 – Etat de l'art

Dans ce chapitre, nous faisons un état de l'art de la cartographie interactive multimédia. Dans un premier temps, nous décrivons très brièvement les modes de stockage de données graphiques. Dans un second temps, nous présentons les principales techniques de représentation cartographique, puis nous nous intéressons aux solutions intégrées de développement cartographique open source[15].

2.1. Les modes de stockage de données graphiques

Il existe deux modes de stockage de l'information graphique : le mode point et le mode vectoriel (cf. figure 2.1). Ils ont des caractéristiques très éloignées, et donc des usages complémentaires. En cartographie, ils sont souvent combinés.

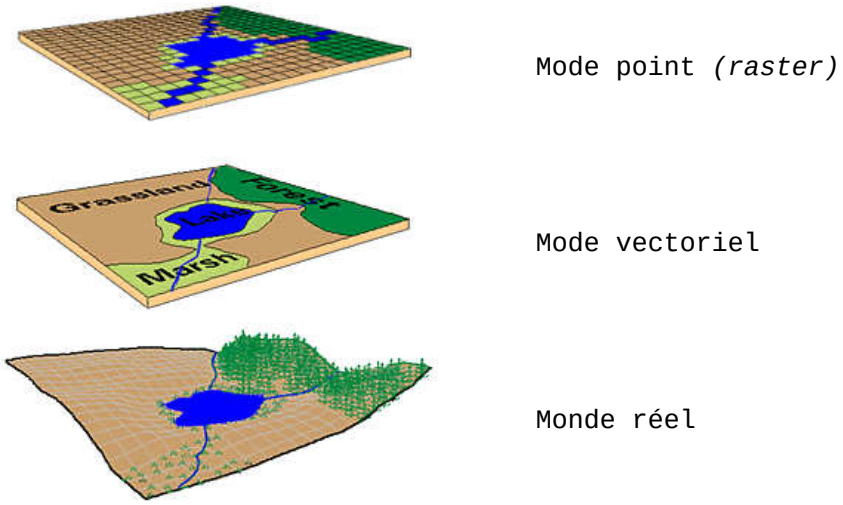


Figure 2.1 : Les deux modes de stockage de données graphiques

D'après [TROU, 02].

2.1.1. Mode point

Le mode point consiste à enregistrer une image ou une vidéo sous forme de grille de points. Le point, également appelés *pixel*[16], est le composant élémentaire. Chaque point est déterminé par une couleur.

[15] Open source : dont le code source soit est dans le domaine public, soit a des droits de copyright mais est distribué sous une licence *open-source* telle la Licence Publique Générale GNU (*General Public License GNU*) [GPL, 91].

[16] Le terme *pixel* est une contraction de l'anglais *Picture Element*.

Les données utilisant ce mode sont généralement volumineuses. Suivant le format de fichier choisi (JPEG[17], PNG[18], etc.), des algorithmes de compression permettent de réduire le volume.

L'obtention d'images ou de vidéos en mode point est facile et peu onéreux. Les sources (photo, dessin) ou les scènes du monde réel sont numérisées à l'aide de scanners, d'appareils photo numériques ou de caméras vidéo numériques.

2.1.2. Mode vectoriel

Le mode vectoriel consiste à définir une image ou une animation en utilisant des formes géométriques simples comme le cercle, le trait, le polygone, la courbe, etc. A chaque forme sont associées une couleur de tracé, une épaisseur de tracé, éventuellement une couleur de remplissage, les coordonnées des points définissant cette forme, etc.

Le mode vectoriel est particulièrement adapté à la cartographie, où les éléments à représenter (les frontières, les voies de communication, les rivières, les courbes de niveaux, les villes, etc.) peuvent être généralisés en formes simples. Les images ou animations vectorielles peuvent être agrandies sans faire apparaître de crénelage (cf. figure 2.2).

Contrairement à la numérisation, la généralisation est une opération coûteuse car elle nécessite souvent une intervention humaine. Les formats vectoriels (Flash, SVG, GML, etc.) permettent d'obtenir des fichiers de données très compactes.

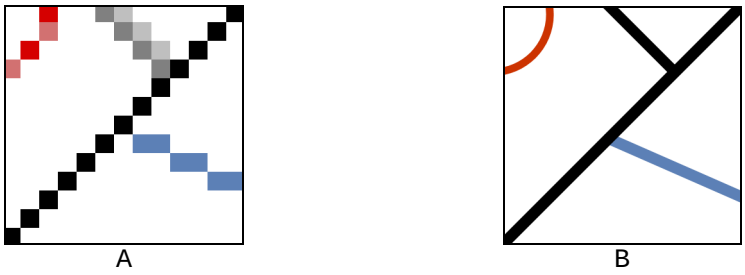


Figure 2.2 : Agrandissement d'images en mode point et en mode vectoriel

La première image (A), en mode point, fait apparaître le crénelage lié à la pixellisation de la numérisation. La seconde image (B) est vectorielle : elle n'est pas dégradée par l'agrandissement.

[17] JPEG : format de compression d'image en mode point avec perte, standardisé par le *Joint Photographic Experts Group*. Ce format est efficace pour les photographies mais la perte de qualité liée à la compression rend les dessins au trait et les textes peu lisibles. Le niveau de compression est modulable.

[18] PNG (*Portable Network Graphics*) : format de fichier pour le stockage compressé, sans perte et portable, d'images en mode point [PNG, 03]. Il est libre de droit et a été créé pour remplacer GIF (*Graphic Interchange Format*) qui ne l'est pas. Il supporte l'indexation des couleurs, l'échelle de gris (*grayscale*), le codage des couleurs sur 24 bits (*truecolor*) et la transparence (*alpha channel*).

2.2. Les techniques de cartographie interactive

Nous abordons les techniques les plus couramment utilisées en matière de cartographie interactive multimédia. Elles sont basées sur des langages de programmation comme Java, sur des langages de script évolués comme Flash ou MPEG-4 et sur des formats de données structurées tel que SVG. Ces techniques sont soit propriétaires comme Java ou Flash, soit issus de normes (SVG et MPEG-4). Elles permettent de combiner des données graphiques en mode point et en mode vectoriel.

2.2.1. Java

Java est un langage de programmation orienté objet, créé par Sun Microsystems en 1995. Java est également un environnement d'exécution, indépendant de la plate-forme du client, ce qui rend les programmes écrits dans ce langage facilement portable. Cette portabilité a permis à cette solution logicielle de connaître un essor retentissant depuis sa première mouture et d'être immédiatement adoptée par de très nombreux développeurs.

2.2.1.1. Principes de Java

Le compilateur Java transforme le code source en *bytecode*, un code abstrait indépendant de l'implémentation d'un code machine. Ce *bytecode* peut ensuite être exécuté sous de nombreux systèmes d'exploitation[19] (OS) grâce à un interpréteur de *bytecode* Java appelé machine virtuelle Java[20] (JVM), ou OS virtuel (cf. figure 2.3).

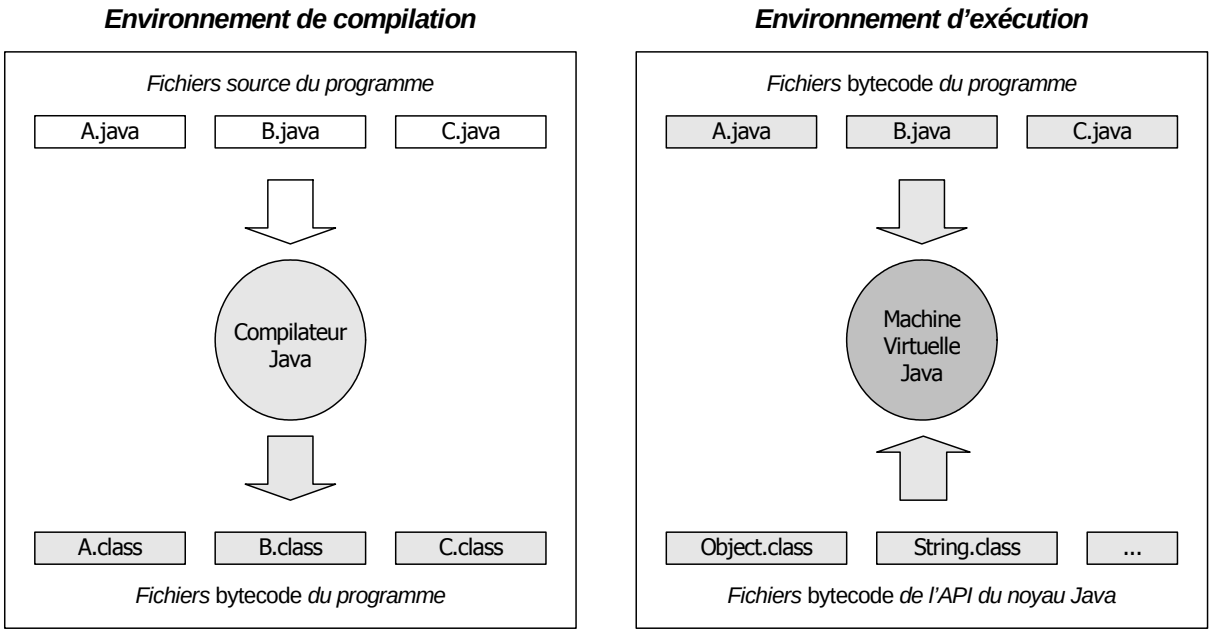


Figure 2.3 : Principes de fonctionnement de Java : compilation et exécution

[19] Système d'exploitation : *Operating System* (OS).
[20] Machine virtuelle Java : *Java Virtual Machine* (JVM).

La JVM est un environnement protégé et sécurisé qui communique avec l'OS de la machine via une couche de communication (cf. figure 2.4). Une partie de cette interface de communication est réalisée par un adaptateur, spécifique à l'OS de la machine. Il existe des adaptateurs pour la plupart des systèmes d'exploitation.

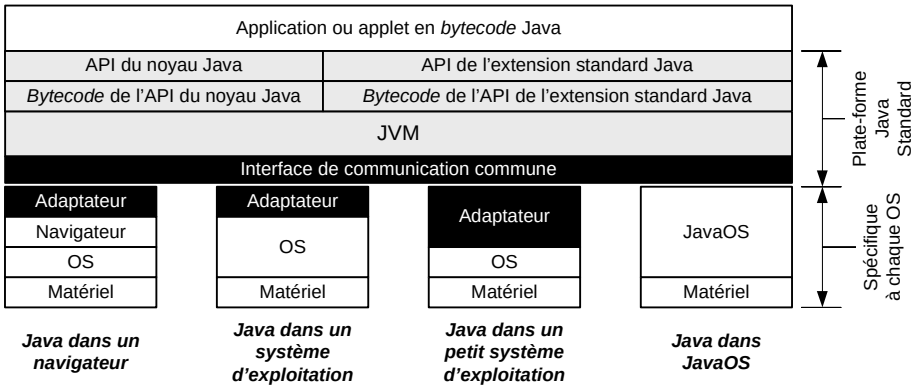


Figure 2.4 : La plate-forme Java est identique sur tous les OS, d'après [KRAM, 96]

L'interface complète de communication entre Java et le navigateur ou l'OS est composée d'une interface de communication commune et d'un adaptateur.

Dans un premier temps, nous présentons les mécanismes du langage Java qui permettent de gérer les interfaces graphiques avec l'utilisateur, puis nous abordons les bibliothèques standard de Java qui reposent sur ces mécanismes.

2.2.1.2. Rendu graphique en Java

Pour faire du rendu graphique – c'est-à-dire afficher un composant à l'écran ou l'imprimer – Java propose deux mécanismes : AWT et Swing.

2.2.1.2.1. AWT

Le paquetage de classes AWT[21] est l'outil des premières versions de Java, permettant de gérer l'interface avec l'utilisateur. AWT sert de couche intermédiaire entre l'interface graphique utilisateur (GUI[22]) de Java et le système graphique de l'OS sur lequel est exécuté le programme Java. Il permet de construire des interfaces à l'aide de composants graphiques (fenêtre, bouton, barre de défilement, étiquette, etc.). Ces derniers, également appelés « contrôles », reposent sur les objets graphiques natifs de l'OS. Les composants graphiques d'AWT sont donc dépendants des limitations de l'implémentation graphique de l'OS. Afin d'être utilisables sur toutes les plates-formes, les contrôles sont peu nombreux et leur aspect est très primitif. Pour répondre à ces limitations, Sun Microsystems a développé Swing[23].

[21] AWT : *Abstract Window Toolkit.*
[22] GUI : *Graphical User Interface.*
[23] Il existe une autre alternative à AWT que Swing : SWT (*Standard Widget Toolkit*) développé par IBM™. A l'instar d'AWT, SWT utilise les composants graphiques natifs de l'OS qui exécute le programme Java, avec toutefois une différence importante : SWT n'utilise que du Java, grâce à JNI (*Java Native Interface*),

2.2.1.2.2. Swing

Cette extension de Java repose sur AWT mais propose des composants entièrement réécrits en Java, aussi riches en fonctionnalités que ceux fournis par AWT. Ainsi, Swing n'emprunte aucun code natif à l'OS. De plus, il comprend de nouveaux composants complexes comme la liste arborescente (*tree view*), la liste déroulante (*list box*) ou les panneaux à onglets (*tabbed pane*). Swing supporte une nouvelle interface nommé PLAF (*Pluggable Look And Feel*) qui permet de choisir l'aspect des contrôles en utilisant des modules d'extension (des paquetages qui se chargent de dessiner les composants).

Les composants Swing ont la capacité de gérer leur rendu à l'aide d'un cache. Cette fonction, appelée *double buffering*, est activée par défaut. A son affichage, un composant calcule son image (position, dimension, couleurs, texte, bordures, etc.), l'enregistre dans le cache et l'affiche. Le réaffichage ne nécessite pas de nouveau calcul : il suffit de redessiner l'image du composant enregistrée dans le cache.

Notons que la cohabitation des composants AWT et Swing dans une même application est fortement déconseillée, du fait des dysfonctionnements et des rendus imprévisibles au niveau de la GUI.

2.2.1.3. Bibliothèques graphiques standard

Certaines classes des API[24] fournies en standard par Java mettent à disposition du développeur les moyens de construire des applications graphiques robustes et d'une grande technicité. Nous décrivons dans cette section deux classes : `Graphics` et `Graphics2D`.

2.2.1.3.1. java.awt.Graphics

Java met à la disposition du développeur la classe abstraite `java.awt.Graphics` permettant de gérer le contexte graphique du rendu d'un composant. Ce rendu peut être envoyé vers des périphériques (écran, imprimante) ou être utilisé pour créer des images.

Pour les objets AWT, la classe `Graphics` permet de représenter du texte, des images et des formes géométriques simples (ligne, ovale, polygone, etc.), de faire des translations et du découpage *(clipping)*.

Pour les objets Swing, cette classe intègre, en plus des fonctionnalités décrites ci-dessus, un calcul automatique de la zone à rafraîchir, rendu performant grâce au *double buffering*.

2.2.1.3.2. java.awt.Graphics2D

L'API Java 2D [J2DA, 04] est une extension de la partie graphique de l'API standard de Java, possédant des capacités étendues de rendu graphique 2D et d'imagerie. Les opérations

---

l'interface Java standard pour le C/C++ (langage dans lesquels sont développés les composants graphiques sur la plupart des OS). Cette utilisation 100% Java permet une meilleure homogénéité du code et facilite grandement le débogage donc la mise au point des composants SWT.

[24] API (*Application Programming Interface*) : interface par laquelle un programme accède à un OS.

simples ou complexes de dessin se font toujours suivant le même nouveau modèle de rendu (cf. figure 2.5). Cette API est complètement indépendante du périphérique cible. La classe `java.awt.Graphics2D` permet de faire des transformations plus complexes et de mieux gérer les polices que la classe `java.awt.Graphics` qu'elle étend. Elle autorise également une gestion de la transparence (*alpha blending*).

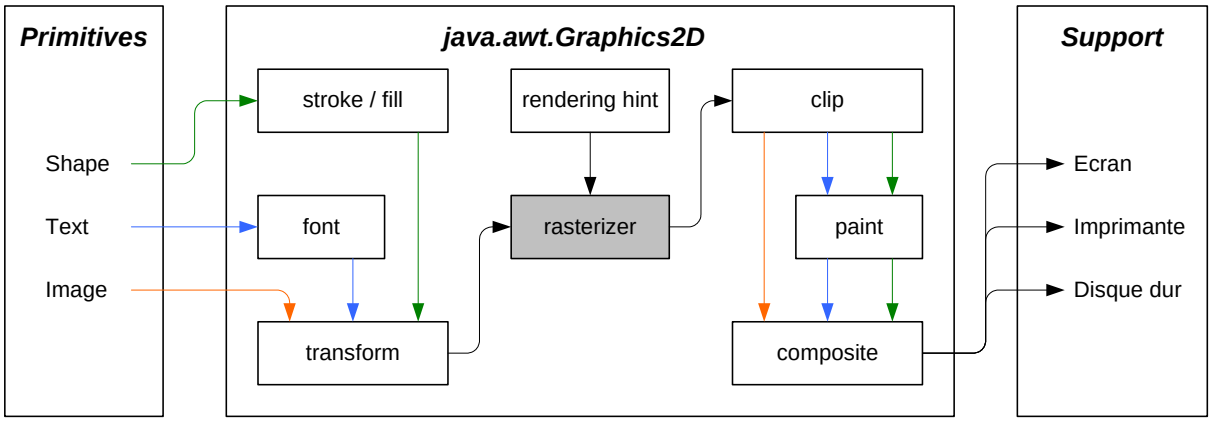


Figure 2.5 : Enchaînement des tâches lors du processus de rendu effectué par la classe `java.awt.Graphics2D`

La tâche rasterizer *effectue une bascule en mode point du rendu.*

Du point de vue « objet », cette nouvelle classe est mieux construite que son aînée, ce qui a pour conséquence de considérablement réduire le nombre de noms de méthode. Ainsi, la méthode `java.awt.Graphics.drawRect(x, y, w, h)` est invocable en utilisant la nouvelle méthode `java.awt.Graphics2D.draw(new Rectangle(x, y, w, h))`. De plus, la méthode `java.awt.Graphics.drawPolygon(new Polygon())` peut être invoquée via la nouvelle méthode `java.awt.Graphics2D.draw(new Polygon())`. Les noms des nouvelles méthodes de dessin se réduisent ainsi à : `draw` et `fill` pour dessiner et remplir une forme (*Shape*), `drawString` pour du texte (*Text*) et `drawImage` pour une image (*Image*).

Notons qu'il existe une méthode qui peut être intéressante pour la cartographie : il s'agit de la méthode `intersects` utilisée pour connaître l'intersection entre l'espace intérieur d'une forme et une zone rectangulaire. Cela permet d'évaluation les possibilités d'inclusions ou de contiguïtés.

La classe Graphics2D fournit à chaque composant un contexte de rendu 2D qu'il suffit de paramétrer avant d'appeler une des nouvelles méthodes de dessin. Ces paramètres sont les suivants (cf. figure 2.5) :

- trait (*stroke*) : taille et style ;
- remplissage de forme (*fill*) : couleur uniforme, dégradé ou motif ;
- composition (*composite*) : manière dont les nouveaux pixels sont combinés avec les anciens en utilisant plus ou moins (ou pas du tout) de transparence ;

- transformation (*transform*) : translation, rotation, changement d'échelle ou toute autre transformation affine libre ;
- zone de découpage du rendu (*clipping*) : restriction aux pixels appartenant à la zone de découpage ;
- police (*font*) : manière dont le texte est pixellisé ;
- type de rendu (*rendering hint*) : choix entre qualité du rendu et rapidité de rendu (anti-crénelage du texte, type d'interpolation graphique, etc.).

2.2.1.4. Bibliothèques open source

Il existe un certain nombre de bibliothèques Java utilisables dans des développements et des distributions de logiciels cartographiques. Nous nous intéressons uniquement aux bibliothèques open source. Elles sont soit spécialisées dans le calcul 2D comme JTS Topology Suite, soit orientées composants graphiques comme OpenMap et GeoTools.

2.2.1.4.1. JTS Topology Suite

JTS Topology Suite (JTS) [JTS, 04] est une API de méthodes d'analyse spatiale 2D basées sur des opérations géométriques (cf. figure 2.6). JTS est écrite exclusivement en Java. Elle est conforme aux spécifications SQL [OGC, 99] de l'Open GIS Consortium[25] (OGC).

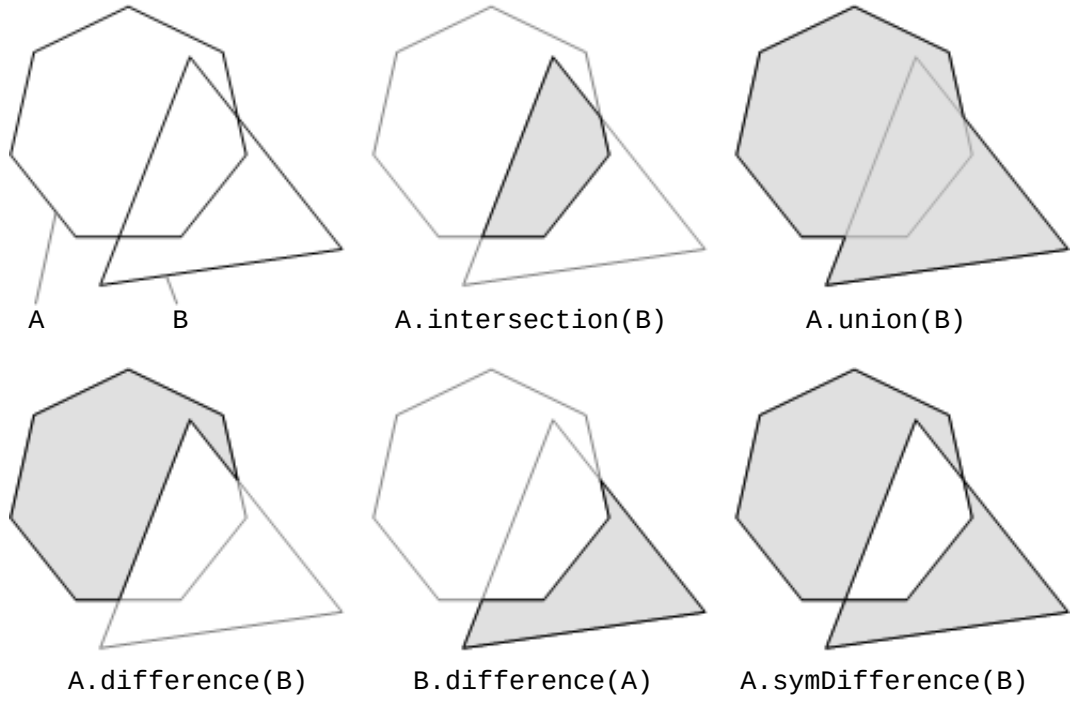


Figure 2.6 : Les opérations géométriques possibles avec JTS

[25] *Open GIS Consortium* (OGC) : organisme qui regroupe les principaux acteurs du marché des SIG et qui conçoit des architectures et langages standardisés assurant l'interopérabilité des SIG.

JTS privilégie la précision des résultats topologiques et géométriques, alors que la rapidité des calculs est un point important mais secondaire. Cependant, JTS est suffisamment rapide pour être utilisé dans la production de logiciels.

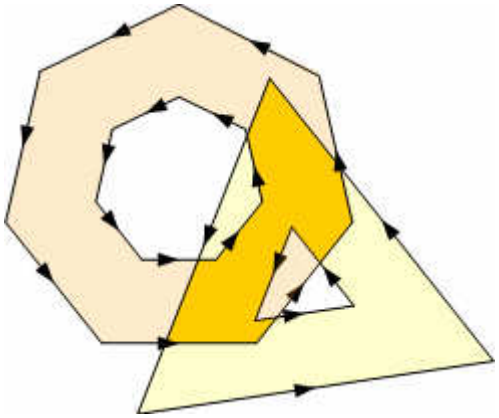


Figure 2.7 : La technique des chaînes monotones utilisée par JTS appliquée ici à l'intersection de deux polygones troués

Cette technique réduit chaque bordure de forme à une liste ordonnée de segments adjacents dont les vecteurs de direction sont tous situés dans le même quadrant. Cette technique rend la recherche d'intersections plus rapide.

Pour optimiser les performances de JTS sans grever la clarté et la lisibilité de son code, ses concepteurs ont utilisé la technique des chaînes monotones (cf. figure 2.7) où chaque bordure est décomposée en chaîne de segments orientés.

2.2.1.4.2. OpenMap

OpenMap™ [OPEN, 04] propose une série d'outils qui facilitent la réalisation d'applications utilisant de l'information géographique. Ces outils sont des composants Swing qui comprennent les coordonnées géographiques. Ils permettent d'afficher des cartes et de capturer les événements de l'utilisateur manipulant l'application. Ce sont des Java Beans[26].

Le composant de base est le `MapBean`. Ce composant est une extension du composant Swing `JComponent`. Il permet d'afficher une large palette de données cartographiques en provenance de multiples sources :

- les sources de données de l'application proprement dite,
- des données en provenance :
  - de serveurs JDBC connectés à des bases de données relationnelles,
  - de serveurs conformes à la spécifications SQL d'OGC,
  - de serveurs Java RMI,
  - de serveurs d'application CORBA.

[26] Java Beans : le Java Bean est un composant logiciel réutilisable qui peut être manipulé visuellement par un outil de construction d'applications. Pour les méthodes d'accès aux attributs et aux événements, cet objet obéit à certaines conventions de nommage.

Les seuls composants pouvant être ajoutés à un `MapBean` sont des calques, instances de classes héritant de la classe `com.bbn.openmap.Layer`. Ils sont ordonnés selon une superposition hiérarchique. Ils sont capables de créer, contrôler et dessiner/peindre les objets graphiques qui les composent. Ils reçoivent les événements déclenchés par la souris de l'utilisateur via le composant `MapBean` et peuvent y répondre. OpenMap propose un certain nombre de classes dérivées de la classe `Layer` : `RasterLayer` – qui affiche des images en mode point, `EarthquakeLayer` – qui permet d'afficher l'activité sismique récente à partir d'une base de données alimentée en temps réel par l'*U.S. Geological Survey*[27] (USGS), `ShapeLayer` – qui affiche des données fournies au format ESRI Shapefile [ESRI, 98]. Nous pouvons créer de nouvelles classes selon les besoins.

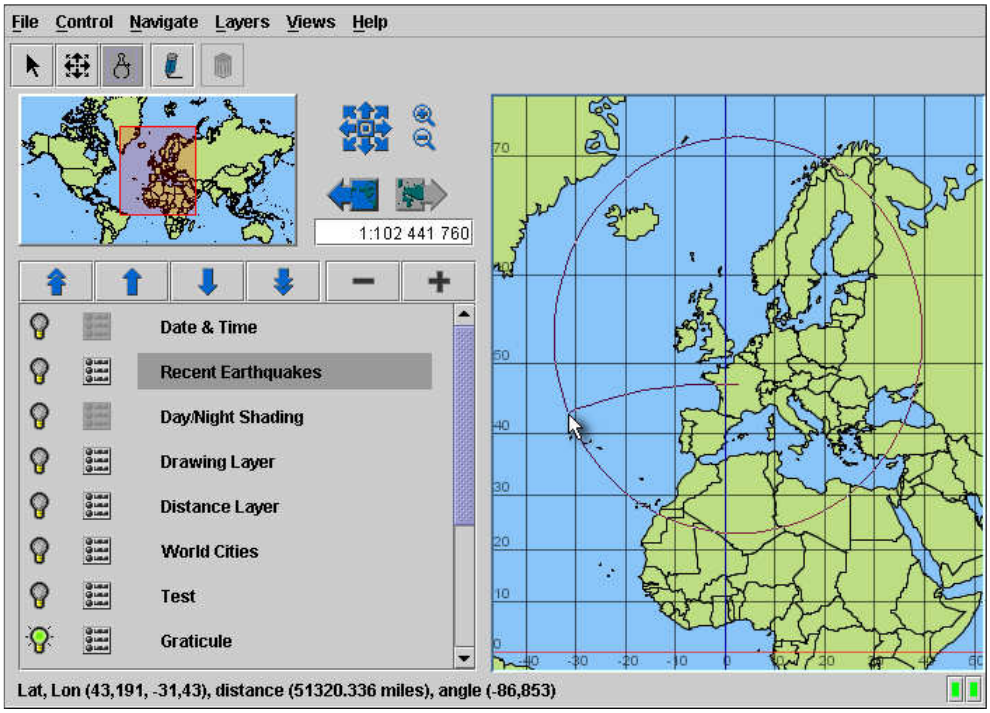


Figure 2.8 : Exemple d'utilisation d'OpenMap : la fonction de calcul de distance

L'ellipse sur la carte représente l'équidistance terrestre à un point géographique (ici le centre de la France). Ce calcul est effectué en temps réel, en réponse aux mouvements de la souris.

Le `MapHandler` est le composant central d'OpenMap qui est une extension du composant Java `BeanContext` permettant de contrôler et de connecter les composants `MapBean`.

OpenMap propose le package `OMGraphics` avec lequel des graphiques vectoriels ou des dessins en mode point peuvent être construits et manipulés.

OpenMap inclut un mécanisme de projection prenant en compte la latitude et la longitude du point central du dessin, l'échelle, la largeur et la hauteur du dessin et le type de projection

[27] *U.S. Geological Survey* (USGS) : organisme de surveillance de l'activité sismique des Etats-Unis d'Amérique.

(Mercator, orthographique, etc.). Cela permet par exemple de calculer des distances dans une projection donnée (cf. figure 2.8).

2.2.1.4.3. GeoTools

GeoTools [GEOT, 04] propose une série d'outils open source écrits en Java. GeoTools est pleinement compatible avec les spécifications de l'OGC[28].

GeoTools offre au développeur de SIG une architecture modulaire facilement extensible pour implémenter des services côté serveur, ou des applications ou applets clientes. GeoTools propose une API. Il met à la disposition du développeur des implémentations par défaut de cette API. GeoTools utilise JTS.

2.2.2. Flash

Flash [BISS, 04] [GEOC, 04] [DANZ, 03] est un outil de conception d'animations interactives à base de graphismes vectoriels et de contenu multimédia. Il est de plus en plus utilisé dans le domaine de la cartographie.

2.2.2.1. Présentation

Flash est un produit de Macromedia. Le format de fichier n'est pas un standard, il est propriétaire. Les spécifications principales ont été publiées par Macromedia [FLAS, 03]. Cependant, certaines spécifications avancées non publiées – telle la compression des scripts – rendent les logiciels de Macromedia indispensables pour la création de fichiers. De 1995 à 2000, le développement rapide du Web a permis à Flash, sans réel concurrent, de s'imposer comme la solution en matière d'animation sur le Web.

Flash autorise un téléchargement progressif. Par exemple, une carte peut être affichée après un téléchargement minimum, et les données permettant d'afficher les détails n'être téléchargées que lorsque l'utilisateur zoome sur la carte. Le format de fichier utilisé par Flash est fortement compressé, ce qui réduit les temps de téléchargement des fichiers, mais peut rendre l'indexation du contenu de ces derniers difficile.

L'animation et l'interactivité avec l'utilisateur sont obtenues grâce à un langage de script qui permet au programmeur d'orchestrer des scènes – c'est-à-dire de construire des scènes chronodatées, et de gérer le survol d'éléments, la sélection de coches, etc.

Flash permet d'interroger des bases de données. En cartographie, cela permet par exemple d'interroger des bases de données réparties contenant chacune les données d'une couche de cartes. L'accès s'effectue selon les besoins exprimés par l'utilisateur.

Le format d'impression est vectoriel, c'est-à-dire que les informations envoyées à l'imprimante sont, non pas en mode point, mais de type vectoriel.

---

[28] Le *Centre for Computational Geography*, d'où sont issus les instigateurs du projet GeoTools, est membre de l'OGC.

Notons que Flash est robuste : très peu de bogues ont été recensés depuis les premières moutures du logiciel.

2.2.2.2. Visualisation

Grâce à l'utilisation de *plug-ins*[29], des réalisations faites en Flash peuvent être intégrées et visualisées dans des pages Web.

Les *plug-ins* de Flash sont compacts : ils ne pèsent que quelques kilo-octets. Il existe des versions du *plug-in* pour tous les navigateurs Web.

2.2.2.3. Un exemple appliqué à la cartographie : GéoClip

GéoClip [GEOC, 04] est un outil de cartographie interactif sur Internet conçu par eMc3[30]. Son interface soignée est réalisée en Flash (cf. figure 2.9). Les données statiques, intégrées lors de la construction de GéoClip, peuvent provenir de MapInfo ou d'ArcGIS[31].

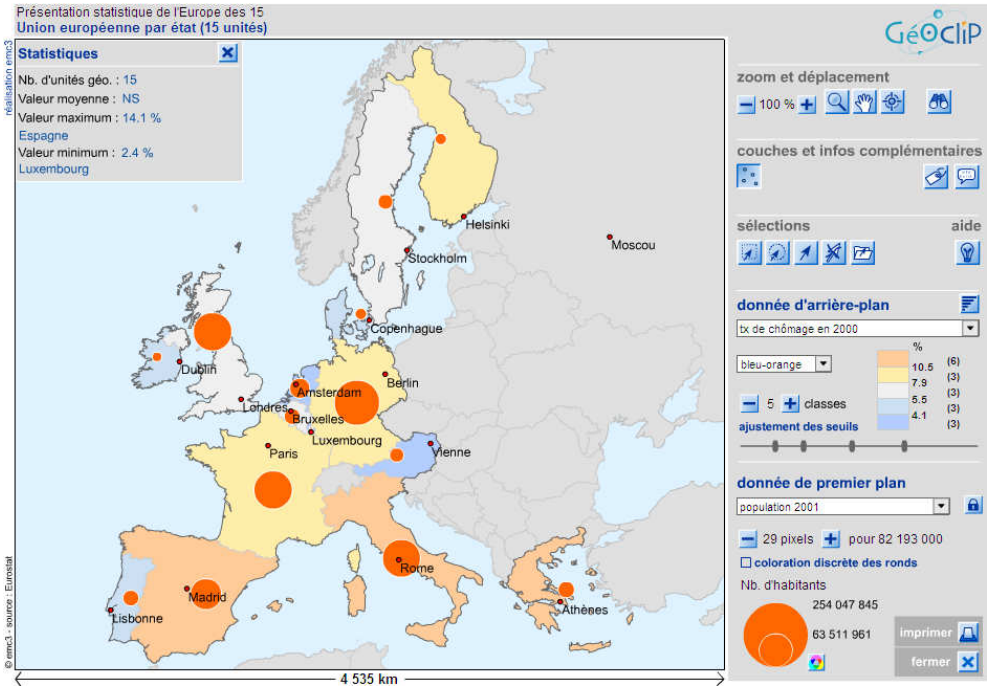


Figure 2.9 : Un outil de consultation cartographique réalisé en Flash : GéoClip

Conçu par des spécialistes en matière de statistiques et de cartographie thématique, GéoClip offre une interface d'une grande souplesse, permettant d'accueillir un nombre très important d'informations et d'adapter le mode de visualisation de la mise en image cartographique en fonction des données.

---

[29] *Plug-in* : programme utilisé comme extension d'un logiciel et permettant de lui ajouter des fonctionnalités.
[30] eMc3 est une jeune société toulousaine, spécialisée dans le traitement de données et leur valorisation.
[31] MapInfo [MAPI, 04] et ArcGIS [ESRI, 04] : solutions intégrées propriétaires, leaders sur le marché des SIG.

Les concepteurs de GéoClip proposent un « constructeur de GéoClip » gratuit, pour MapInfo (version 4.5 et plus) ou ArcGIS (version 8 et plus). Il se compose d'une interface Flash compilée, d'un script MapBasic pour MapInfo et d'une DLL pour ArcGIS.

Le minimum requis par ce constructeur est un fond de carte de polygones pourvu d'au moins deux champs : l'un pour nommer les unités géographiques, l'autre pour décrire la population de chaque unité. Il peut traiter jusqu'à dix variables numériques et autorise la définition de cinq indicateurs supplémentaires sous forme de ratio. Le constructeur fabrique alors une application géostatistique parfaitement autonome, qui peut être placée sur un site Internet ou recopiée sur un cédérom. Depuis MapInfo ou ArcGIS, le constructeur génère la page HTML de lancement et regroupe les données géographiques et statistiques dans un (ou deux) fichier(s) SWF compressé(s). Tout navigateur équipé de la version 6 du *plug-in* Flash peut alors visualiser ces fichiers.

L'interface a été spécialement optimisée pour permettre au lecteur Flash d'effectuer tri, calculs statistiques et dessins complexes sans l'aide d'aucun serveur. Le constructeur de GéoClip peut traiter jusqu'à 3 000 unités géographiques.

2.2.3. SVG

SVG (*Scalable Vector Graphics*) est un format de données permettant de définir des graphiques vectoriels 2D. Recommandé par le W3C[32] depuis novembre 2000 [SVG, 03], ce format est rapidement devenu un standard reconnu et utilisé.

2.2.3.1. Présentation

Le format SVG est basé sur une structure balisée de type XML[33]. A l'instar de tout document XML, SVG décrit une syntaxe lisible et compréhensible par l'Homme. Pour être valide, un document SVG doit être conforme à la DTD[34] de SVG.

SVG permet de définir des primitives (rectangle, ellipse, courbe de Béziers, forme librement définie, etc.), des effets de style (transparence, dégradé, etc.), des transformations géométriques (changement de dimension, rotation, translation), des interactions avec l'utilisateur (clic ou survol de souris, touche de clavier enfoncée) et des animations. L'utilisateur, par interaction, peut déclencher des animations, mais ces dernières ont la possibilité d'être scénarisées et déclenchées en fonction du temps. Pour cela, SVG suit la recommandation SMIL[35] du W3C.

---

[32] W3C (*Wold Wide Web Consortium*) : organisme qui œuvre depuis 1994 au développement du *World Wide Web* (WWW), en concevant des architectures, protocoles et langages standardisés qui assurent son évolution et son interopérabilité.

[33] XML (*eXtended Markup Language*) : langage de balisage extensible [XML, 04].

[34] DTD (*Document Type Definition*) : définition de la structure que doit respecter un document XML d'un type donné (SVG, XHTML, MathML, SMIL, XMT, etc.) pour être considéré comme valide.

[35] SMIL (*Synchronized Multimedia Integration Language*) [SMIL, 01] : langage de grammaire XML qui standardise la synchronisation multimédia.

A la manière de la balise `<img/>` du XHTML[36], SVG permet d'inclure dans un document des références à des fichiers images en mode point ou à des fichiers SVG. Des transformations géométriques et des modifications de style peuvent être appliquées à ces éléments inclus.

La taille des fichiers SVG peut être réduite de 50 à 80% grâce à l'algorithme de compression gzip (ils possèdent alors l'extension `.svgz`).

SVG est de mieux en mieux supporté par les SIG, qui permettent d'exporter au format SVG, ou de visualiser des documents SVG.

2.2.3.2. Visualisation

Pour visualiser des documents SVG dans un navigateur Web, de nombreux *plug-ins* existent. Le plus utilisé est actuellement Adobe SVG Viewer [ADOB, 01], téléchargeable gratuitement depuis le site de l'éditeur, dont la version 3.0 supporte presque toute la recommandation SVG 1.1 (la dernière version de la spécification SVG en date). Notons également qu'un navigateur Web intégrant la lecture native du SVG (sans utiliser un *plug-in*) est en cours de développement par le projet Mozilla [MOZI, 04].

2.2.3.3. Bibliothèque open source

Pour intégrer du SVG dans une réalisation logicielle, il est possible d'intégrer une bibliothèque qui se charge de fournir tous les outils nécessaires.

Pour Java, la bibliothèque la plus connue s'appelle Batik[37] SVG Toolkit [BATI, 04]. Elle est open source, écrite par *The Apache Software Foundation* (créateur du serveur Web du même nom). Lorsqu'elle est intégrée à une application Java ou une applet Java, elle permet un rendu graphique et une manipulation dynamique des documents SVG. Batik fournit un composant capable d'afficher la représentation graphique d'un document SVG et de la modifier à l'aide de méthodes de changement d'échelle, de translation et de rotation. Une API d'accès au DOM[38] est également fournie. Elle permet de recevoir des messages sur l'état du chargement du document en mémoire, de modifier dynamiquement le document SVG chargé en mémoire, ou d'être averti des actions de l'utilisateur (zoom, clic, survol) sur un élément donné du document.

2.2.4. MPEG-4

MPEG-4 [BISS, 04] [FLEU, 98] [DANZ, 03] est un format de compression de données audio et vidéo. Mais il permet également l'encodage de scènes composées d'éléments multimédia – son, vidéo, texte, graphique vectoriel 2D et 3D – synchronisés entre eux.

---

[36] XHTML (*eXtensible Hypertext Markup Language*) : langage de balisage hypertexte extensible [XHTM, 01]. Le XHTML est une reformulation en XML du HTML (*HyperText Markup Language*) – langage utilisé dans les pages Web.

[37] Le batik est un art textile millénaire en provenance d'Indonésie, plus précisément de l'île de… Java.

[38] DOM (*Document Object Model*) : spécification du W3C pour manipuler des documents XML [DOM, 04].

MPEG-4 est un standard issu du MPEG[39]. A l'instar du JPEG (cf. note de bas de page 17), ce format compresse l'audio et la vidéo avec perte, de manière modulable : plus la compression est importante, plus la perte est grande. Cependant, les scènes, décrites au format XMT[40], sont compressées sans perte dans un format binaire réversible : BIFS[41].

Ce double format XMT-BIFS est intéressant. BIFS permet un téléchargement plus rapide et XMT facilite :

- la lecture par l'Homme,
- la validation de scène par sa DTD,
- l'indexation automatique du contenu,
- la transformation par XSLT[42] vers d'autres langages basés sur XML comme SVG.

MPEG-4 permet d'animer les scènes, et d'interagir avec l'utilisateur. Comme Flash, du contenu peut être téléchargé à la demande ou selon les besoins. De plus, MPEG-4 gère le *streaming*[43], autorisant ainsi la lecture d'un fichier en cours de téléchargement.

En cartographie, grâce à l'utilisation de primitives et de transformations 3D dérivées de VRML[44], MPEG-4 permet d'effectuer des opérations sur les courbes de niveaux (comme la simulation d'inondations), sur l'orientation des pentes (telle la variation de l'ensoleillement), sur l'analyse du relief (telle la visibilité d'un point depuis un autre point) ou encore de faire la projection du panorama visible en un point donné.

Les outils de création de scènes sont très peu nombreux. A l'heure actuelle, nous n'avons pas trouvé d'implémentations cartographiques de cette norme, qui nous semble pourtant prometteuse.

[39] MPEG (*Motion Picture Expert Group*) : groupe chargé de définir des normes vidéo, pour le compte de l'ISO (*International Standard Organization*). Le MPEG est également à l'origine du format MPEG-2 – utilisé pour les DVD Video et pour la télévision numérique, et du format MPEG-2 Audio Layer 3 (communément appelé MP3).

[40] XMT (*eXtensible MPEG-4 Textual*) : format textuel de description de scènes du MPEG-4, basé sur XML.

[41] BIFS (*BInary Format for Scene*) : format binaire d'enregistrement de scènes du MPEG-4.

[42] XSLT (*eXtensible Stylesheet Language Transform*) : langage basé sur XML qui permet de décrire des transformations entre deux langages basés sur XML.

[43] *Streaming* : utilisation de flux de données.

[44] VRML (*Virtual Reality Modeling Language*) : langage de modélisation de scènes tridimensionnelles.

2.3. Les environnements de développement cartographique

Nous présentons dans cette section des outils de développements intégrés comme MapServer ou JUMP, qui permettent de gagner du temps de développement en fournissant une palette d'outils et de fonctionnalités facilement réutilisables.

2.3.1. MapServer

MapServer [MAPS, 04] est un environnement open source de développement d'applications de publication cartographique sur le Web basée sur des données géo-référencées. Une collaboration entre l'Université du Minnesota, la NASA et le Département des Ressources Naturelles du Minnesota est à l'origine de ce projet. MapServer peut être installé sur les principaux serveurs Web et fonctionne sous Windows, Linux et MacOS.

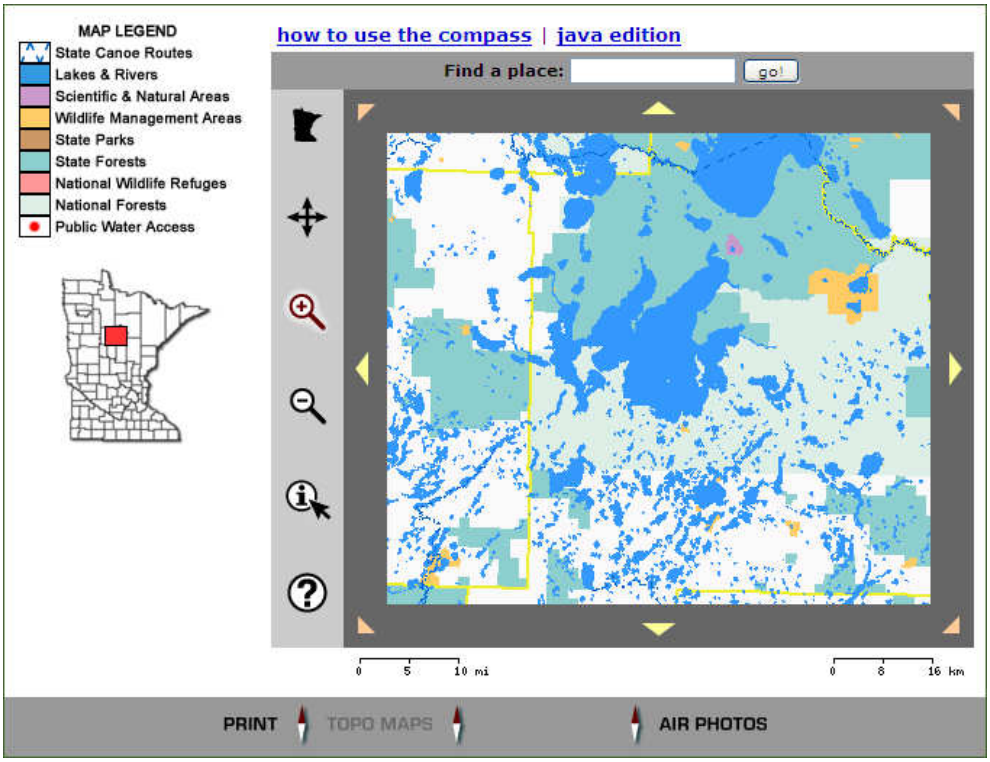


Figure 2.10 : Exemple d'implémentation de MapServer sur le site du Department of Natural Resources of Minnesota

Les formats de données vectorielles supportés en entrée sont les principaux formats propriétaires utilisés par les SIG (MapInfo, ArcView, Oracle Spatial, PostgreSQL / PostGIS, etc.). Les principaux formats d'image sont également supportés. En sortie, MapServer peut produire des documents aux formats : GIF, PNG, JPEG, PDF, SHP (ArcView), SVG, SWF (Flash), etc.

Cette solution est utilisable à travers une application CGI installée sur un serveur Web ou en utilisant MapScript qui permet à de nombreux langages de programmation (C, Java, Perl, PHP, Python, etc.) d'accéder à l'API MapServer écrite en C.

A l'instar d'OpenMap, cette solution repose sur une carte (MapFile) où sont organisées des couches (cf. figure 2.10) pouvant provenir de diverses sources (bases de donnée, fichiers de données, image en mode point, etc.). Ce MapFile peut être enregistré statiquement dans un fichier et être consulté dynamiquement grâce à l'utilisation de MapScript.

En outre, cette solution propose toutes les fonctionnalités classiques d'un SIG : zoom, déplacement, interrogation, numérisation, etc.

2.3.2. JUMP

L'*Unified Mapping Platform* (JUMP) [JUMP, 04], basé sur Java, est une application open source dotée d'une GUI (le *Workbench*) pour représenter et manipuler des données spatiales. Elle inclut de nombreuses fonctions de calcul spatial et géographique.

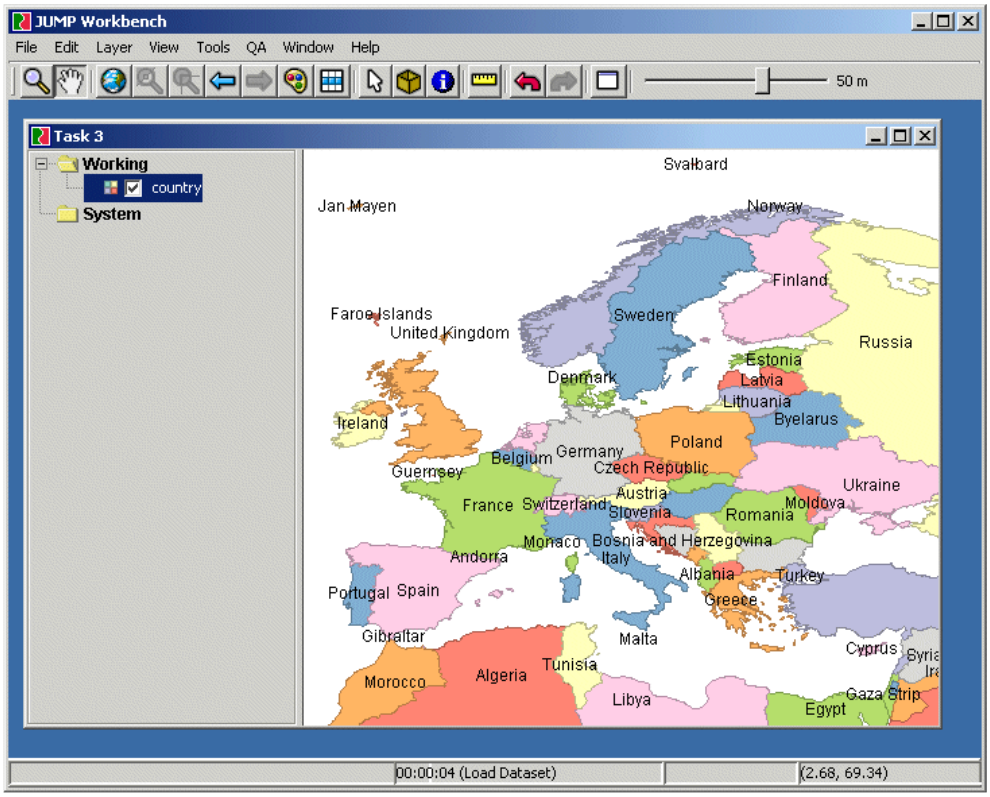


Figure 2.11 : Exemple de coloration attributaire dans le Workbench *de JUMP*

Le *Workbench* (cf. figure 2.11), qui est une interface à base de fenêtres de tâche, est également utilisable pour développer et exécuter des applications personnalisées de calcul de données spatiales. Chaque fenêtre peut afficher plusieurs calques de données qui peuvent provenir d'une multitude de formats de données spatiales[45], dont GML [OGC, 03] et ESRI Shapefile. Les calques peuvent être coupés, copiés et collés. Les données peuvent être manipulées via les vues spatiales ou via les vues tabulaires.

[45] Grâce à l'API I/O (*Input/Output* :).

Les calques sont habillables avec différents styles de couleur, de trait et de remplissage de forme. La coloration des unités territoriales (cf. paragraphe 3.2.1) en fonction de leurs attributs est possible suivant les valeurs discrètes ou suivant des plages de valeurs.

JUMP est un client pour les *Web Map Services*[46] (WMS), notamment grâce à une interface pour créer et modifier les requêtes WMS.

Cette application fournit de nombreux outils d'analyse : calcul de surface et de longueur, opérations spatiales comme l'intersection, l'union, la différence, la différence symétrique[47]. Elle permet les transformations affines (rotation, translation etc.), le voilement (*warping*) et la détection d'erreurs de géométrie (points dupliqués, géométries invalides etc.).

2.4. Synthèse

Les quatre solutions techniques présentées permettent l'implémentation d'applications cartographiques 2D solides et évoluées. Flash n'est pleinement exploitable qu'en utilisant l'environnement payant fournit par Macromedia. MPEG-4, quant à lui, souffre d'un manque d'outils efficaces pour la lecture et la création de documents. Pour les besoins liés à la réécriture du logiciel Hypercarte, deux solutions retiennent particulièrement notre attention : d'une part Java, qui bénéficie du support d'une grande communauté de développeurs et pour lequel de nombreuses bibliothèques open source existent, et d'autre part SVG, qui est un format standard dont les outils de création sont ceux du XML et dont les *plug-ins* de lecture sont gratuits.

Nous verrons par la suite que nos choix pour la version 1.0 – très proche des solutions techniques choisies pour la version 0.9 – ont été guidés par des impératifs liés au temps de développement.

Les environnements de développement cartographique offrent des interfaces graphiques et des outils de transformation géométriques, pour composer des cartes à partir de bases de données statiques. Or le logiciel Hypercarte est basé sur des données produites dynamiquement par calcul.

Dans le chapitre suivant, nous présentons le logiciel Hypercarte.

[46] *Web Map Service* : ce service cartographique basé sur le Web est une spécification de l'OGC [OGC, 01].
[47] JUMP utilise JTS (cf. figure 2.6).

Chapitre 3 – Le logiciel Hypercarte

Nous présentons dans ce chapitre le logiciel développé dans le cadre du projet Hypercarte. Dans une première partie, nous établissons les principes généraux du logiciel. Ensuite, nous définissons les concepts cartographiques utilisés par Hypercarte. Puis, nous décrivons l'utilisation du logiciel. Enfin, nous présentons les différentes cartes générées.

3.1. Principes généraux

Dans le cadre du projet de recherche ORATE 3.1[48], les membres du réseau Hypercarte mettent au point des outils interactifs de prospectives territoriales. Il s'agit de coupler une base de données des régions européennes (Europe des 25 + pays candidats) avec un ensemble de procédures de mesure de la cohésion territoriale des régions et de leur situation à différents niveaux d'analyse (écart à la moyenne européenne, à la moyenne nationale, à la moyenne des régions voisines). Des procédures plus spécifiques d'étude des régions frontalières sont également en cours d'élaboration (cartographie des discontinuités, des gradients).

3.2. Concepts

3.2.1. Unité territoriale – UT

Une unité territoriale (UT) est une zone géographique délimitée par une frontière. Les unités territoriales s'agrègent en unités supérieures (UTS) (cf. figure 3.1). Les UT qui ne sont pas des UTS sont appelées unités territoriales élémentaires (UTE).

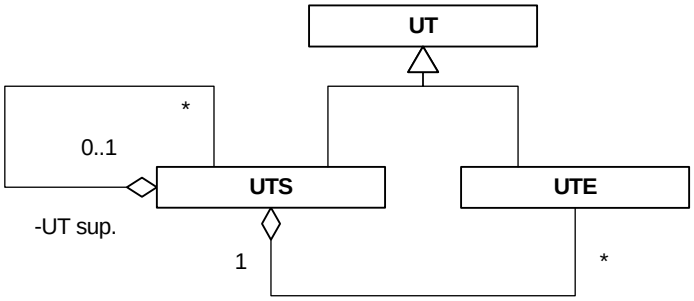


Figure 3.1 : Diagramme de classes : unités territoriales supérieures et élémentaires

3.2.2. Espace

L'espace est une entité géographique de type économique et/ou politique. Cet espace est décomposé en unités territoriales (cf. figure 3.2).

[48] Le volet 3.1 d'ORATE concerne les outils intégrés d'aménagement du territoire européen (*Integrated Tools for European Spatial Planning*)

Figure 3.2 : Diagramme de classes : unités territoriales et espaces d'étude

Hypercarte utilise différents espaces tels que « l'Europe de 25 », « les candidats à l'entrée dans l'Union Européennes », « l'Europe des 25 + les candidats » ou « l'arc atlantique ».

3.2.3. Maillage

Le maillage est une hiérarchisation des découpages de l'espace en UT disjointes les unes des autres (cf. figure 3.3).

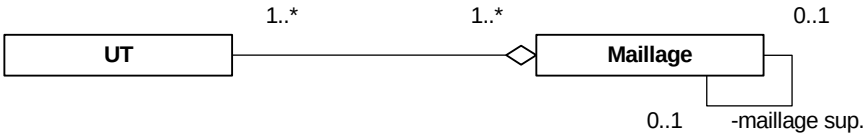


Figure 3.3 : Diagramme de classes : unités territoriales et maillages

Nous pouvons citer comme exemple de maillage la Nomenclature des Unités Territoriales Statistiques (NUTS), utilisée par l'Union Européenne pour ses statistiques et permettant de faire des correspondances entre les découpages administratifs de chacun des membres de l'Union Européenne (cf. figure 3.4). L'équipe PARIS a adopté ce maillage dans les données qu'elle a fournies pour le logiciel Hypercarte.

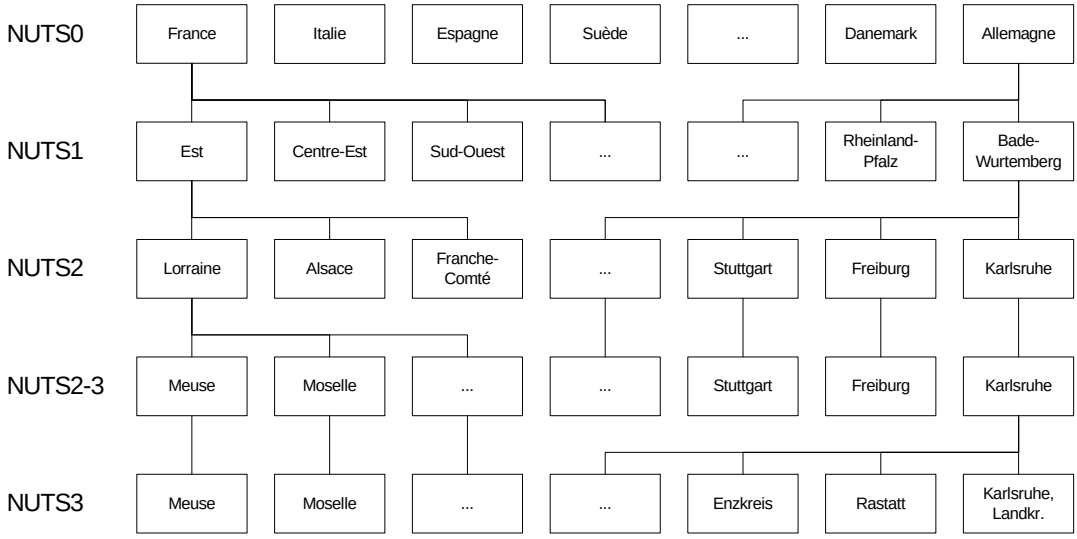


Figure 3.4 : Extrait du maillage NUTS

Les niveaux de maillage NUTS sont les suivants : NUTS0 (pays européens), NUTS1 (*landers* allemands ou ZEAT[49] en France), NUTS2 (régions françaises), NUTS3 (départements français), NUTS4 (*districts* anglais), NUTS5 (communes françaises). Un niveau NUTS2-3

[49] ZEAT : Zones d'Etudes et d'Aménagement du Territoire.

combine les unités de niveau NUTS2 et NUTS3 dans le but d'obtenir un ensemble d'unités plus homogènes (cf. figure 3.5).

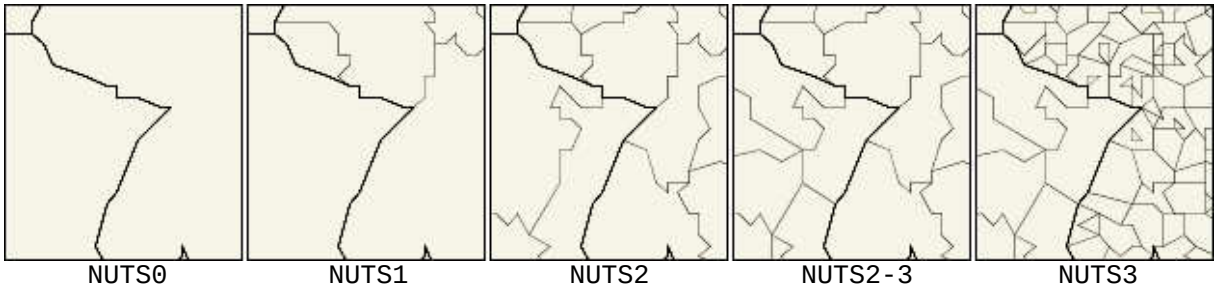


Figure 3.5 : Représentation cartographique des niveaux du maillage NUTS

Cette carte montre les mailles NUTS au niveau de l'est de la France (Alsace, Lorraine) et du sud-ouest de l'Allemagne (Bade-wurtemberg, Rheinland-Pfalz). Les unités de niveau NUTS0 sont tracées en noir, les autres en gris.

3.2.4. Statistiques et stocks

Les statistiques sont des données chiffrées permettant de comparer des entités entre elles. Ces données portent sur la géographie, la démographie, l'économie etc.

Une valeur de stock est associée à chaque UT et à chaque statistique disponible dans l'application (cf. figure 3.6).

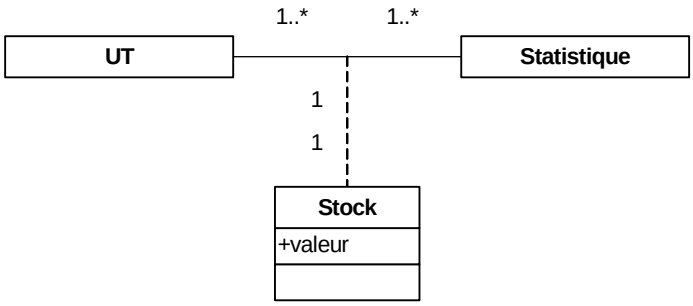


Figure 3.6 : Diagramme de classes : UT et statistiques

Dans le logiciel Hypercarte, nous utilisons par exemple les statistiques suivantes : « PIB en 1999 en millions d'euros », « Population totale moyenne en milliers d'habitants », « Nombre d'hommes au chômage en milliers », « Superficie de l'unité territoriale en Km² ».

3.2.5. Géométries

Les géométries des UT sont décrites par des suites de points formant des polygones (cf. figure 3.7).

L'ensemble des concepts présentés est repris dans un diagramme en annexe (cf. annexe 2).

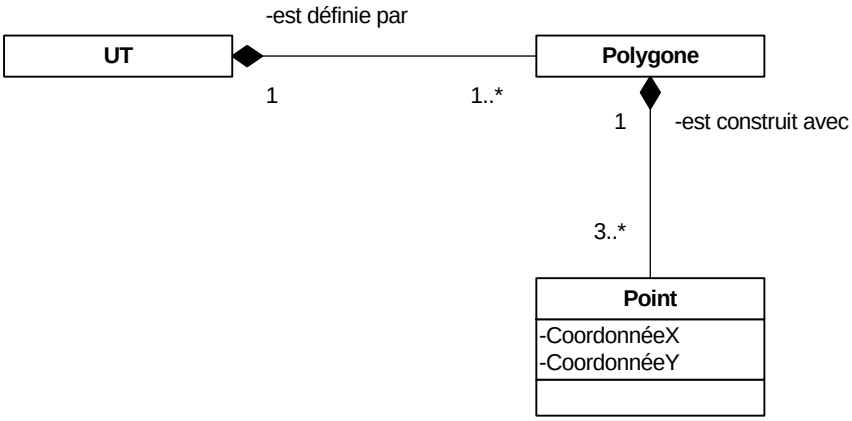


Figure 3.7 : Diagramme de classes : UT et géométrie

3.3. Types de représentation cartographique

Suivant les besoins, de multiples représentations peuvent être utilisées pour afficher des données sur une carte. Dans Hypercarte, deux types de représentations sont utilisés : les cartes à base de symboles proportionnels et les cartes choroplèthes, décrites ci-dessous.

3.3.1. Cartes à base de symboles proportionnels

Les cartes à base de symboles proportionnels sont des cartes affichant des valeurs absolues (population, nombre de naissances, etc.). Nous utilisons généralement le cercle, qui est un symbole simple et lisible (cf. figure 3.8). La surface du cercle est proportionnelle à la donnée représentée.

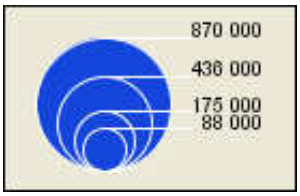


Figure 3.8 : Exemple de légende de carte utilisant des symboles proportionnels

3.3.2. Cartes choroplèthes

Les cartes choroplèthes sont des cartes affichant des valeurs relatives (densité de population, taux de natalité, etc.), basées sur des rapports ou des taux. Elles les représentent en remplissant les zones avec une couleur déterminée à l'aide d'une discrétisation (cf. figure 3.9). Il faut pour cela déterminer les seuils bornant les plages de valeurs discrétisées.

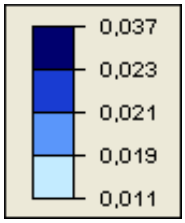


Figure 3.9 : Exemple de légende de carte choroplèthe

3.4. Fonctionnement

Le but du logiciel Hypercarte est de produire à la volée des cartes d'analyse territoriale. Pour chaque UT d'un espace d'étude, un indicateur composé de deux valeurs de stock est comparé :

- à la moyenne de cet indicateur pour un espace donné (carte de déviation globale) ;
- à la valeur de cet indicateur pour une UT supérieure à un niveau de maillage donné (carte de déviation moyenne) ;
- à la moyenne des UT voisines (carte de déviation locale).

Cette application est donc un générateur de cartes d'analyse territoriale. Son fonctionnement est simple : il met en correspondance des paramètres choisis par l'utilisateur avec des cartes. Celles-ci sont toujours à jour par rapport aux paramètres choisis (i.e. lorsque l'utilisateur modifie un paramètre, les cartes sont immédiatement mises à jour).

A l'ouverture de l'application, les paramètres sont positionnés avec des valeurs par défaut et les cartes sont affichées. Il suffit à l'utilisateur de modifier les valeurs des paramètres pour constater immédiatement la mise à jour des cartes. Il peut naviguer à travers l'interface pour aller d'une carte à une autre, pour afficher les détails d'une UT (cf. figure 3.10). L'utilisateur a également la possibilité de faire des traitements cartographiques tels que le changement d'échelle ou la translation (cf. figure 3.11).

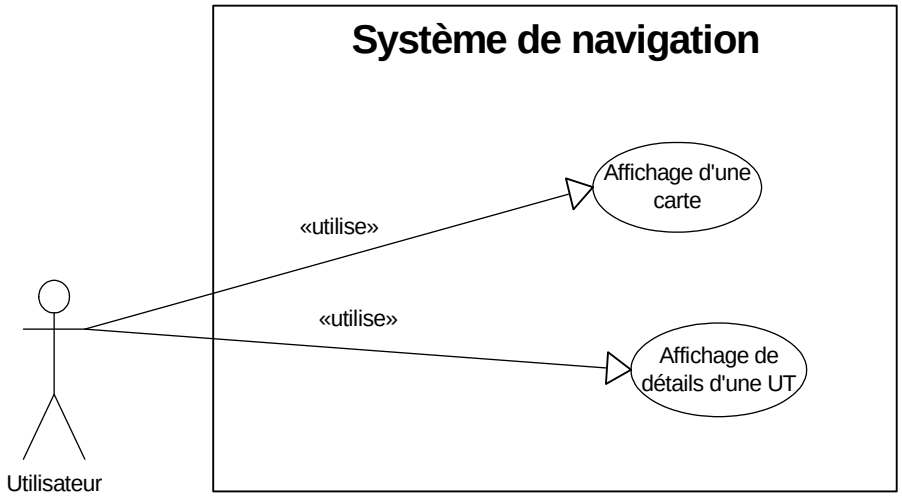


Figure 3.10 : Diagramme des cas d'utilisation de la navigation dans le logiciel

Les étapes de construction d'un jeu de cartes sont les suivantes (cf. figure 3.12) :

1. sélection de l'espace d'étude et du niveau de maillage sur lesquels sont basées les cartes ;

2. choix parmi les statistiques du numérateur et du dénominateur ;

3. sélection de l'espace de référence de la déviation globale, du niveau de maillage utilisé par la déviation moyenne et du type de déviation locale.

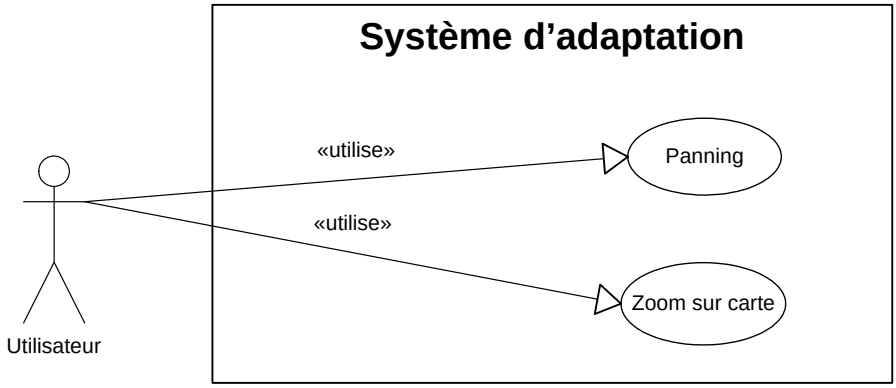


Figure 3.11 : Diagramme des cas d'utilisation des adaptations cartographiques du logiciel

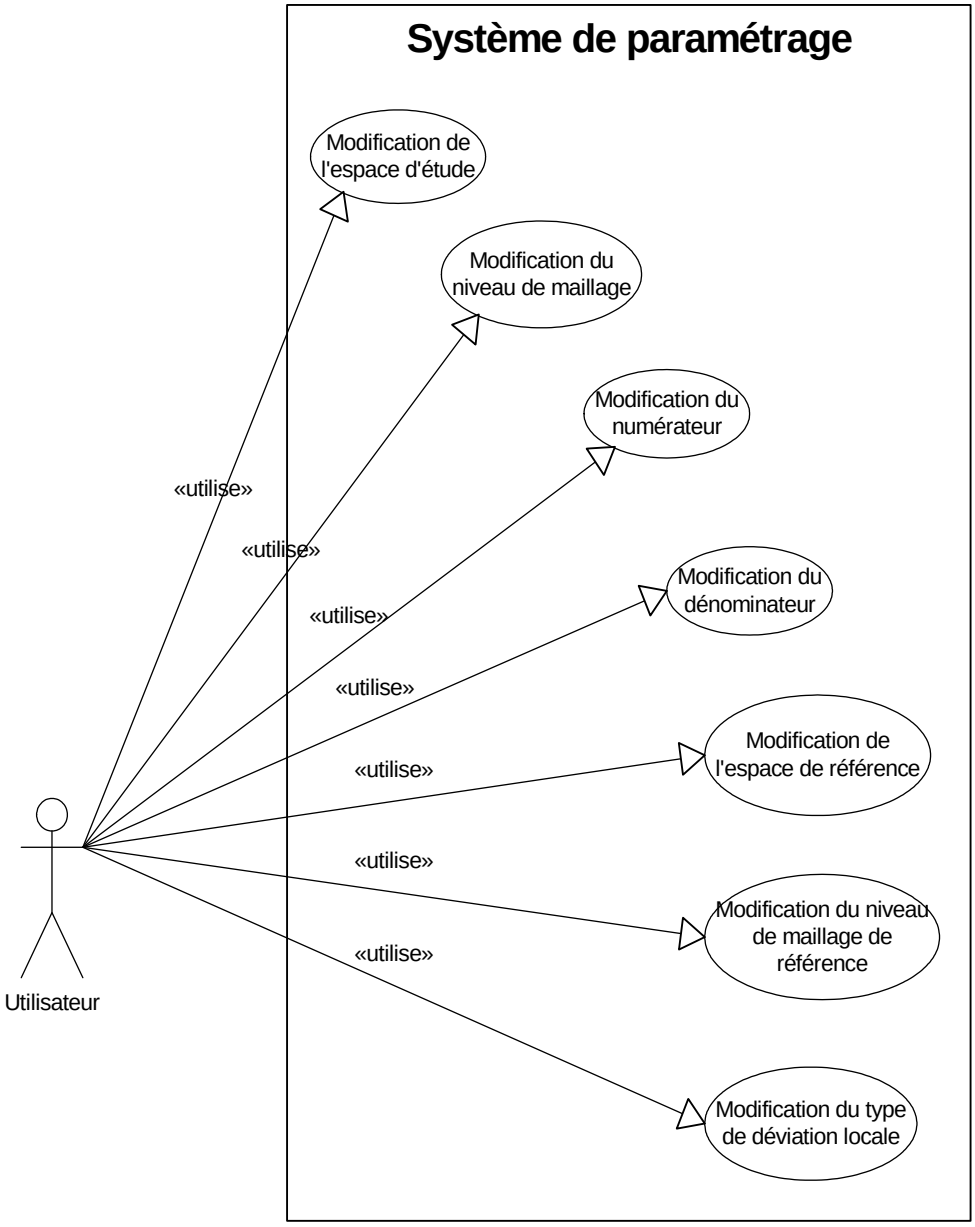


Figure 3.12 : Diagramme des cas d'utilisation des paramètres du logiciel

3.5. Les cartes produites

Hypercarte produit sept cartes différentes, de type choroplèthe ou à base de cercles proportionnels. Nous pouvons distinguer différents groupes de cartes : les cartes de contexte, de stock, de rapport, de déviation et de synthèse.

3.5.1. Carte de contexte

La carte de contexte met en évidence la zone correspondant à l'espace d'étude (cf. figure 3.13). De plus, les UT de l'espace d'étude qui correspondent au niveau de maillage sélectionné sont affichées. Cette carte n'est pas une carte d'analyse, mais elle permet un retour visuel à l'utilisateur sur son choix d'espace d'étude et de niveau de maillage.

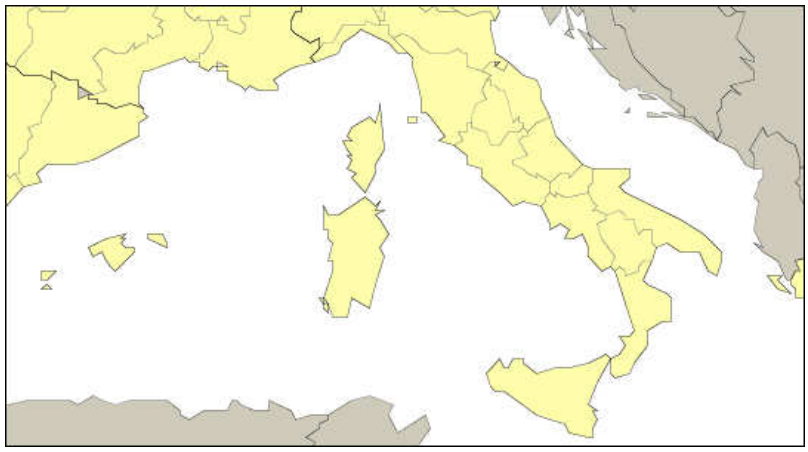


Figure 3.13 : Carte de contexte

Dans cet exemple de carte de contexte, l'espace d'étude est en jaune et les UT du niveau de maillage sélectionné en gris foncé.

3.5.2. Cartes de stock

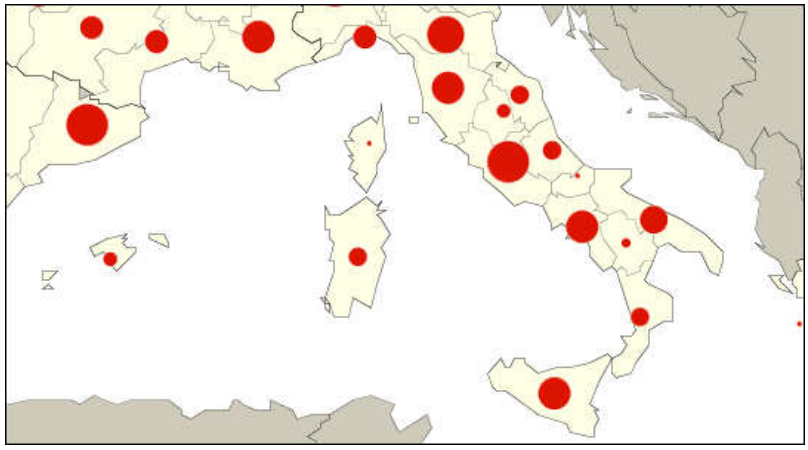


Figure 3.14 : Carte de stock

Le logiciel Hypercarte propose deux cartes, l'une pour visualiser les valeurs de stock du numérateur et l'autre pour visualiser celles du dénominateur. Les valeurs de chaque UT de l'espace d'étude appartenant au niveau de maillage choisi sont représentées par des disques

proportionnels, i.e. la taille d'un disque est proportionnelle à la valeur qu'il représente (cf. figure 3.14).

3.5.3. Carte de rapport

Cette carte présente le rapport entre le numérateur et le dénominateur. Ce rapport, appelé indicateur, est rendu en utilisant pour chaque UT un aplat dont la couleur est déterminée par une palette mettant en correspondance des couleurs et des plages de valeurs (cf. figure 3.15). Les couleurs de l'échelle sont d'une même tonalité et classées selon un dégradé.

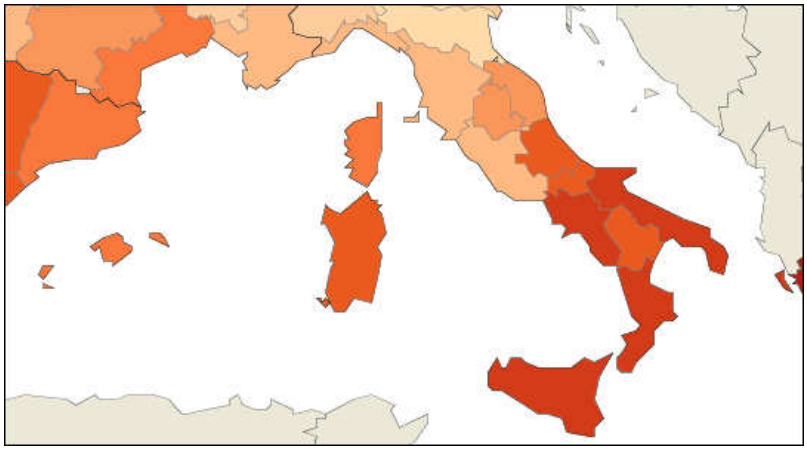


Figure 3.15 : Carte de rapport, choroplèthe avec une palette de couleurs monotonique

3.5.4. Cartes de déviation

La déviation met en rapport la valeur de l'indicateur d'une UT avec la valeur du même indicateur d'une UTS située à un niveau de maillage supérieur. Elle peut aussi comparer la valeur de l'indicateur de cette UT avec la moyenne des valeurs de cet indicateur pour des UT du même niveau de maillage ayant un lien particulier avec l'UT (contiguïté, proximité, etc.).

Le logiciel Hypercarte s'intéresse à trois déviations : la déviation globale (ou macro-déviation) par rapport à un espace, la déviation moyenne (ou méso-déviation) par rapport à un niveau de maillage plus élevé et la déviation locale (ou micro-déviation) par rapport au voisinage.

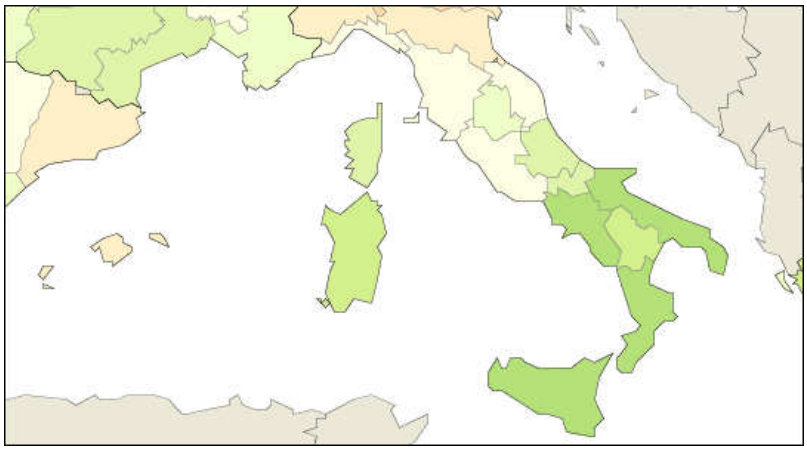


Figure 3.16 : Carte de déviation, choroplèthe avec une palette de couleurs bitonique

Les cartes de déviation (cf. figure 3.16) montrent comment se situe l'indicateur d'une UT par rapport à une valeur. Dans le cas de la carte de déviation globale, cette valeur est la moyenne de l'indicateur des UT d'un espace donné. Dans le cas de la déviation moyenne, nous utilisons l'indicateur d'une UT supérieure par rapport à l'UT en cours (située alors à un autre niveau de maillage). La carte de déviation locale utilise la moyenne de l'indicateur des UT voisines.

Les couleurs utilisées dans les cartes de déviation sont issues d'une palette de couleurs formée par un dégradé entre deux couleurs.

3.5.5. Carte de synthèse

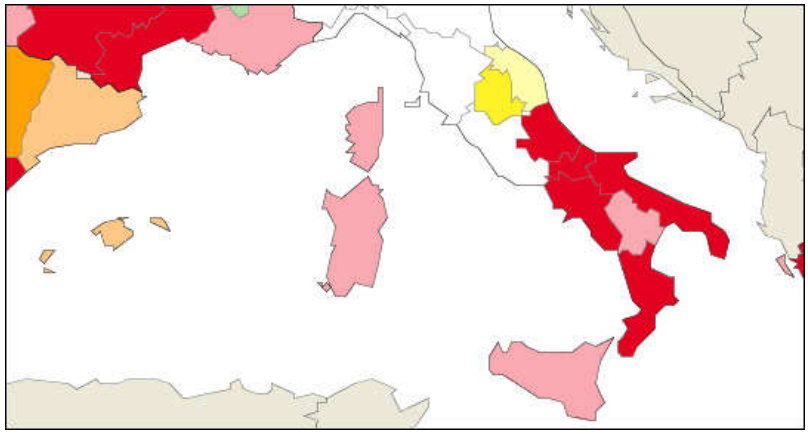


Figure 3.17 : Carte de synthèse des déviations

La dernière carte proposée par Hypercarte est une synthèse des trois déviations (cf. figure 3.17). Elle classe les UT en fonction de la comparaison entre leur indicateur et chacune de leurs trois déviations.

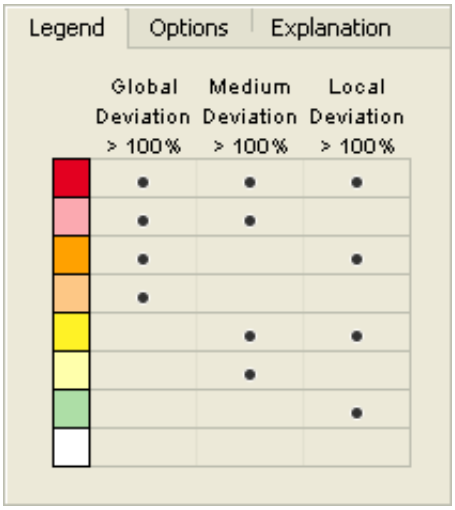


Figure 3.18 : Légende de la carte de synthèse de la version 1.0.

La première couleur de la palette (rouge) est utilisée pour les UT dont la valeur du rapport est supérieure aux trois déviations. La deuxième est utilisée pour celles dont la valeur de rapport est supérieure à la déviation globale et moyenne, et inférieure à la déviation locale. Etc.

Les spécifications de cette carte ont évolué entre les versions 0.9 et 1.0. Toutefois, le principe est le même pour les deux versions [GRAS, 03b]. Cette carte est difficilement lisible sans le support de la légende (cf. figure 3.18).

3.6. Synthèse

Le logiciel Hypercarte, bâti sur des principes originaux, repose sur des concepts élémentaires. Simple de fonctionnement, il produit une série de cartes dont l'analyse croisée permet l'étude territoriale multiscalaire de phénomènes sociaux.

Dans le chapitre suivant, nous présentons et analysons la version 0.9 du logiciel Hypercarte.

Chapitre 4 – Analyse de la version 0.9 d'Hypercarte

Dans ce chapitre, nous faisons une analyse de la version 0.9 d'Hypercarte, sur laquelle se base notre travail.

Dans un premier temps, nous détaillons les techniques choisies. Nous nous intéressons ensuite à la structure et à la composition de l'interface utilisateur. Puis, nous discutons de l'implémentation choisie. Enfin, nous faisons une synthèse des points à améliorer dans cette version.

4.1. Techniques utilisées

4.1.1. Java *vs* SVG

Lors de la phase de maquettage d'Hypercarte, une étude comparative, portant sur différentes techniques de rendu cartographique permettant de construire une nouvelle application, a été réalisée. Trois techniques ont été testées :

- l'utilisation de Java [MOIS, 02] en mode applet ;
- l'utilisation de SVG en natif ;
- une utilisation hybride, mêlant Java et SVG : un fichier SVG est généré sur le serveur et affiché sur le client grâce à une applet Java.

Le manque de maturité de SVG au moment de cette étude a finalement orienté le choix d'implémentation vers une utilisation de Java sans SVG.

Le rendu graphique est obtenu grâce aux classes Java standard[50] `java.awt.Graphics` et `java.awt.Graphics2D`. L'interface est construite avec des composants Swing[51]. Aucune bibliothèque graphique ou géométrique additionnelle n'est utilisée.

4.1.2. Architecture

La version 0.9 d'Hypercarte n'est pas de type client-serveur comme prévu initialement dans les spécifications, du fait de la lenteur des échanges entre le serveur et le client. Les données sont placées côté client et les calculs (agrégations territoriales et agrégations attributaires) sont effectués préalablement.

Des fichiers textuels (cf. figure 4.1) contiennent les définitions géographiques et les attributs des UTE ainsi que les relations entre les UT. A partir de données fournies sous forme de

[50] Cf. paragraphe 2.2.1.3.
[51] Cf. paragraphe 2.2.1.2.2.

fichiers, un programme – le « sérialiseur » – construit une arborescence. Chaque nœud de cette arborescence représente une UT. Chaque UT possède ses propres données territoriales (nom, code), attributaires (valeurs de stock) et géométriques (coordonnées des points permettant son tracé). Pour cette version d'Hypercarte, la sérialisation dure une quinzaine de minutes.

Pendant l'exécution du sérialiseur, l'arborescence d'UT est stockée en mémoire sous forme d'instances de classes. Le sérialiseur utilise la classe `java.io.ObjectOutputStream` qui permet de « sérialiser » des instances en un flux. Le sérialiseur enregistre le flux dans un fichier, puis arrête son exécution. L'arborescence ainsi sérialisée est persistante.

Le client Hypercarte, lorsqu'il se charge en mémoire à son exécution, lit le fichier et désérialise les instances en utilisant la classe `java.io.ObjectInputStream`.

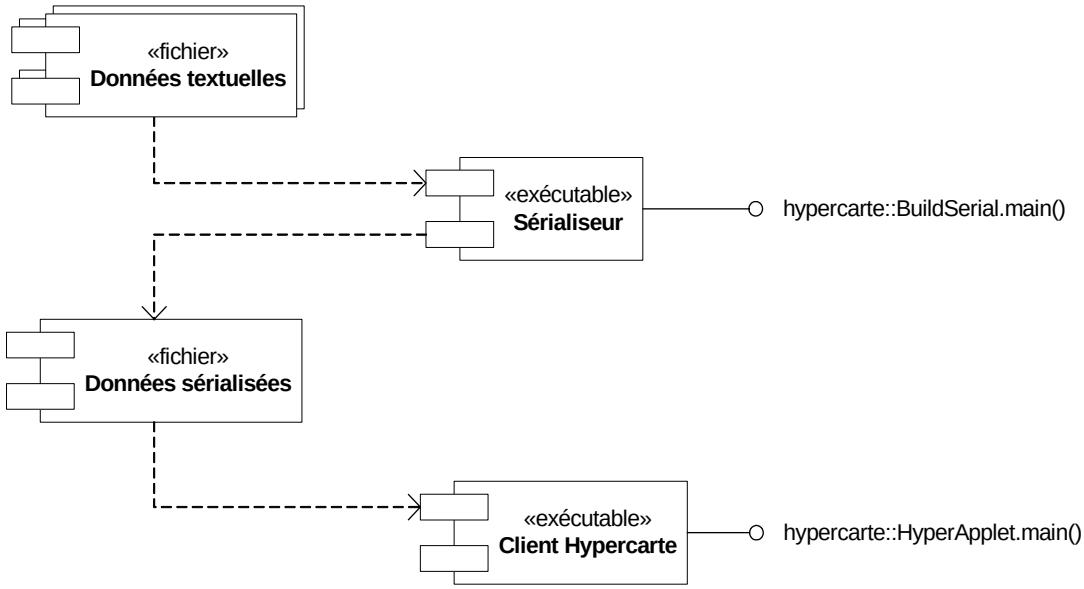


Figure 4.1 : Diagramme de composants : les composants d'Hypercarte (vue macroscopique)

La désérialisation est très rapide : les instances sont rechargées en mémoire telles qu'elles étaient lors de leur sérialisation. De plus, le fichier sérialisé est très compact par rapport à des formats textuels comme XML.

4.2. Interface utilisateur

L'interface utilisateur de la version 0.9 du logiciel Hypercarte (cf. figure 4.3) est issue d'une maquette d'interface réalisée par l'équipe Géographie-Cités (cf. figure 4.2). Les structures des deux interfaces sont très similaires. Cependant, certaines fonctionnalités de la maquette n'apparaissent pas dans la version 0.9, comme :

- la modification manuelle des valeurs de seuil,
- la restriction d'un stock à certains sous-stocks,
- la fonction d'export de carte.

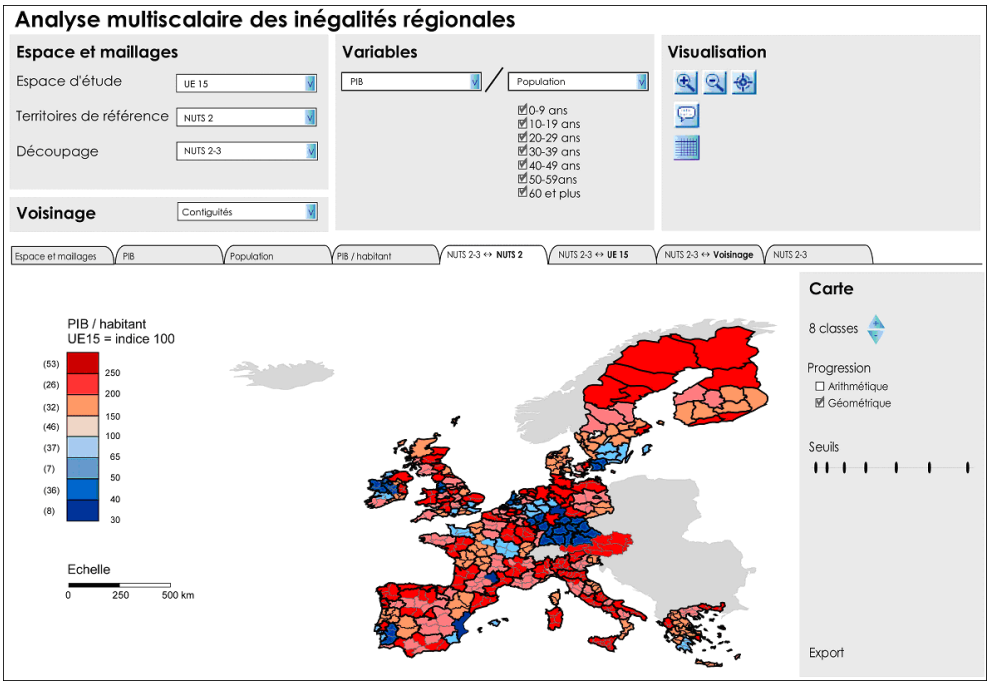


Figure 4.2 : La maquette d'Hypercarte

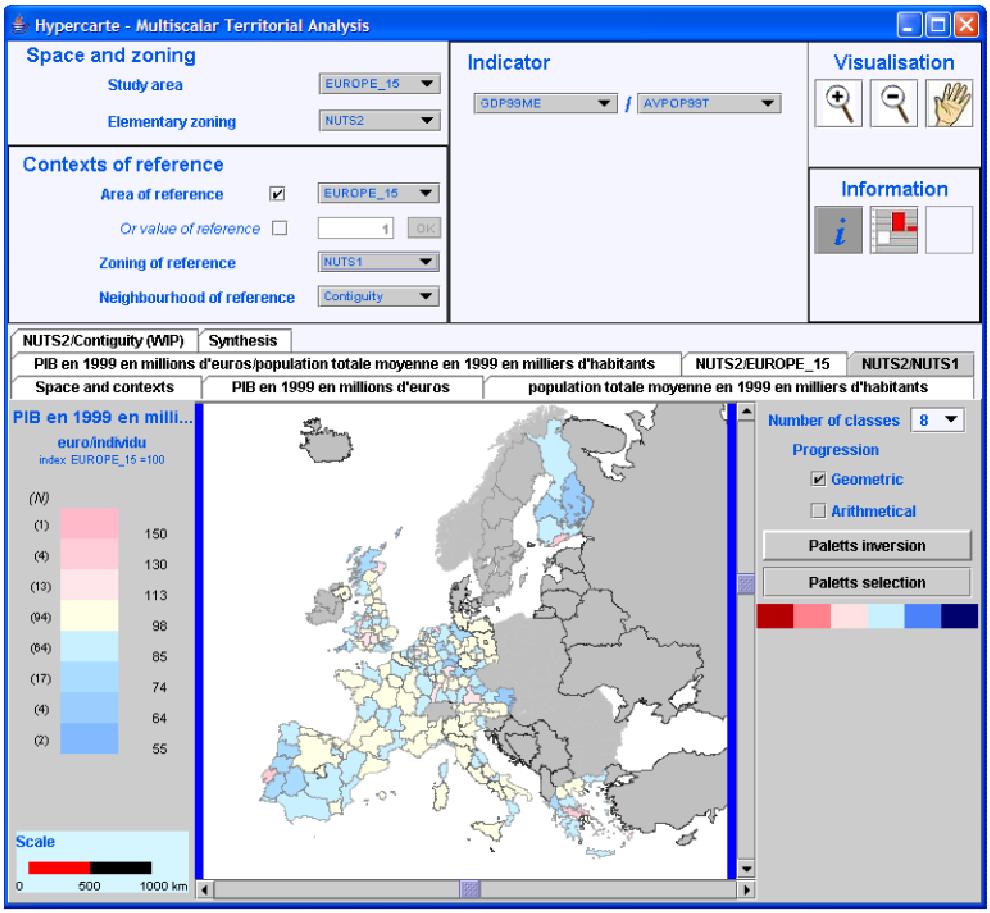


Figure 4.3 : Interface de la version 0.9 d'Hypercarte

Toutefois, la version 0.9 présente des fonctionnalités supplémentaires, comme :

- le choix de l'espace de référence ou d'une valeur de référence spécifiée par l'utilisateur,
- le choix de la palette de couleurs,
- l'inversion de palette.

4.3. Implémentation

4.3.1. Chargement des données dans l'application

Les données en entrée de la version 0.9 sont fournies par les géographes sous forme de fichiers textuels tabulés. Ces fichiers sont lus par la fonction de sérialisation d'Hypercarte (cf. figure 4.1). Ils sont de trois types : territorial, attributaire et géométrique.

4.3.1.1. Données territoriales

Les données territoriales sont stockées dans plusieurs fichiers. Par exemple, le fichier `ut.txt` décrit les codes d'UT et leur intitulé, le fichier `utSup.txt` met en relation les UT avec des UTS (cf. figure 4.4), le fichier `espace.txt` indique pour chaque UT si elle appartient ou non à chacun des espaces.

```
FR1    FR
FR101  FR1
FR102  FR1
FR103  FR1
FR104  FR1
FR105  FR1
```

Figure 4.4 : Extrait du fichier utSup.txt

Les codes situés à gauche représentent des UT, les codes situés à droite représentent leur UTS.

4.3.1.2. Données attributaires

Les données attributaires sont constituées de valeurs pour chaque stock et pour chaque UTE. Ces valeurs sont stockées dans un seul fichier textuel (cf. figure 4.5).

```
CodeUT UT              AREAKM2  GDP99ME  GDP99MP  UNT99  UNM99  UNF99  ACPT99 ACPM99
FR101  Paris           105.4    136621.9 128881.8 120    62.4   60.1   1139.9 571.8
FR102  Seine-et-Marne  5915.3   23529.5  22196.5  47.7   22.4   25.5   520.3  284.3
FR103  Yvelines        2284.4   35834.1  33804    47.2   23.9   23.6   633.3  346
FR104  Essonne         1804.4   27469.8  25913.5  42.9   21.3   21.9   545.6  293
FR105  Hauts-de-Seine  175.6    73880.7  69695.1  62.1   32.2   30.9   713.6  368.5
```

Figure 4.5 : Extrait du fichier de valeur de stock

Ce fichier contient à la fois les intitulés des UT et les valeurs des UT pour tous les stocks.

4.3.1.3. Données géométriques

Chaque UT est représentée sur une carte par un ou plusieurs polygones. Un fichier décrit pour chaque UTE la liste des coordonnées géographiques des points qui constituent ce(s) polygone(s) (cf. figure 4.6).

```
FR101  64134  148018 9119   159655 61873  116790 19409  133456 18308  51115  95124  ...
FR102  4113   62406  121615 129796 63116  133268 120620 36232  68257  8875   65462  ...
FR103  91459  101389 75771  60795  53461  81585  105552 109057 26397  89645  33323  ...
FR104  79060  44988  53052  86721  160252 25509  31655  116628 75805  94924  74821  ...
FR105  26949  162772 107772 130599 104458 64134  75365  35586  128033 72124  56411  ...
```

Figure 4.6 : Extrait du fichier de coordonnées géographiques

Pour dessiner les cartes à base de cercles proportionnels, il faut également connaître le centroïde de chaque UT. Ce centroïde est généralement le barycentre du polygone. Or, dans certains cas, le barycentre est inapproprié. Par exemple, si nous prenions comme centroïde pour la Norvège son barycentre, et compte tenu de la forme générale de ce territoire, le cercle proportionnel de la Norvège serait dessiné sur la Suède (cf. figure 4.7). Il est donc nécessaire de laisser aux géographes la possibilité de déterminer les centroïdes. Ces derniers peuvent être fournis dans le fichier `centroide.txt`.

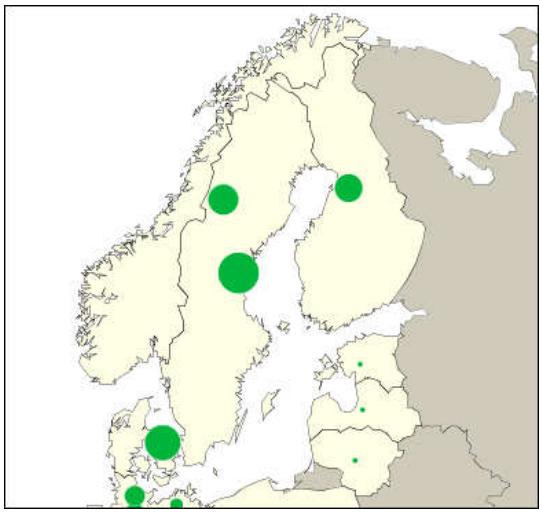


Figure 4.7 : Exemple de centroïdes basés sur le barycentre polygonal

Le centre du disque représentant la valeur associée à une UT est positionné sur le barycentre du polygone définissant ce territoire. La Norvège ayant une forme de croissant, son barycentre est situé à l'extérieur du polygone, sur le polygone représentant la Suède.

4.3.2. Agrégation des géométries et des données attributaires

Les données attributaires et les coordonnées géographiques sont fournies par les géographes uniquement pour les UT de niveau de maillage le plus bas. Ces UTE sont agrégées pour former les UT de niveau supérieur, ces dernières sont à leur tour agrégées, etc.

L'agrégation géométrique consiste à supprimer les arêtes communes aux polygones agrégés (cf. figure 4.8).

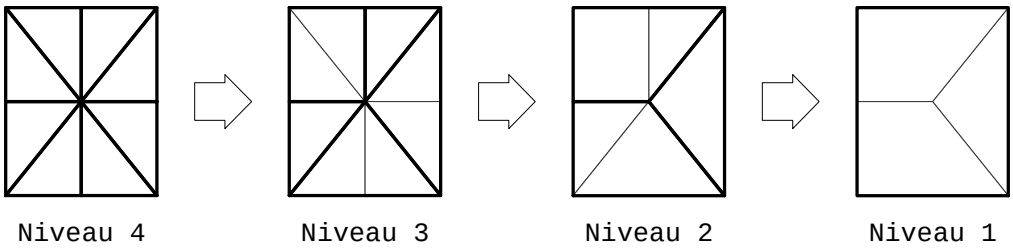


Figure 4.8 : Exemple d'agrégations géométriques

Les traits pleins et épais délimitent les UT. Les traits fins en pointillé représentent les arêtes supprimées par agrégation. Les flèches symbolisent des agrégations.

L'agrégation des données attributaires somme les valeurs de stock des UT agrégées pour fournir les valeurs de stock de l'UT de niveau supérieur.

4.4. Problématique / Champs d'exploration

Les points d'amélioration identifiés dans la version 0.9 concernent la performance, l'évolutivité, l'ergonomie. Nous citons également quelques bogues majeurs.

4.4.1. Performances

Les performances sont médiocres en raison, entre autres, de mauvais choix d'implémentation (une seule instance de carte, absence de cache) et de données cartographiques mal ajustées

4.4.1.1. Une seule instance de carte

La version 0.9 n'instancie qu'une fois la classe `Chart` qui est chargée de faire le rendu de la carte (cf. code 4.1).

Code 4.1 : Code montrant le déplacement de l'instance de carte, d'un onglet à un autre

```
public class HyperApplet extends JApplet implements ...
{
   public void init()
   {
      carte1.add(editeurLegende1,  BorderLayout.EAST);
      carte1.add(panneauLegende1,  BorderLayout.WEST);
      carte1.add(jPanel2,  BorderLayout.CENTER);
      jPanel2.add(jPanel7,  BorderLayout.CENTER);
      jPanel7.add(imageCarte1,  BorderLayout.CENTER);
   }
   private Chart imageCarte1 = new Chart();
   private JTabbedPane jtabbedpane = new JTabbedPane();

   // ...

   protected void ongletAffecte() {
      int indiceSource = jtabbedpane.getSelectedIndex();
      int indiceCarte = jtabbedpane.indexOfComponent(carte1);
```

```
      String titreCarte = jtabbedpane.getTitleAt(indiceCarte);
      if (indiceSource != indiceCarte) {
         jtabbedpane.remove(carte1);
         if (indiceSource > indiceCarte) indiceSource--;
         jtabbedpane.setComponentAt(indiceSource, carte1);
         jtabbedpane.insertTab(titreCarte, null, new JPanel(),"",indiceCarte);
         jtabbedpane.setSelectedComponent(carte1);
      }
   }
}
```

La variable `imageCarte1` qui est l'unique instance de la classe `Chart` (ligne 11) est ajoutée au panneau `carte1`. Lorsque l'utilisateur clique sur un onglet, la méthode `ongletAffecte()` est invoquée. Si l'onglet cliqué n'était pas sélectionné, alors `carte1` est supprimé (ligne 18) du panneau à onglets `jtabbedpane`, remplacé par un panneau vide (ligne 21), puis réinséré sous l'onglet cliqué (ligne 20).

Cette instanciation unique pose plusieurs problèmes.

Le premier est lié à la perception du changement de carte par l'utilisateur. Lors d'un clic sur un onglet, la carte en cours d'affichage disparaît, remplacée par un panneau gris, puis, un moment plus tard, est remplacée par la carte qui correspond à l'onglet cliqué. L'affichage de ce panneau gris est lié à la disparition du panneau qui contient la carte courante. Pendant que la nouvelle carte se reconstruit, la fenêtre est rafraîchie, rendant ce panneau gris visible.

Le second est plus critique en terme de performance. Lorsque l'utilisateur clique sur un onglet, la carte est déplacée mais également recalculée (les géométries sont filtrées et redessinées), avant d'être affichée. Ainsi, lorsque l'utilisateur réaffiche une carte déjà visionnée, la version 0.9 recalcule entièrement la carte.

4.4.1.2. Absence de cache

A chaque affichage ou réaffichage de carte, les actions suivantes sont réalisées :

- les UT de l'espace d'étude, au niveau de maillage choisi, sont obtenues en parcourant la liste complète des UT d'Hypercarte ;
- les calculs basés sur les données attributaires (le calcul de l'indicateur, les calculs des trois déviations) sont refaits.

Cela explique également en partie la lenteur d'affichage des cartes.

4.4.1.3. Définition inadéquate des contours cartographiques

Le fichier de définition des géométries utilisé pour la version 0.9 contient des contours d'UT qui présentent un effet de crénelage rendant les tracés peu lisibles et surtout nécessitant un nombre élevé d'arcs. Cela a pour conséquence d'augmenter le temps nécessaire pour dessiner les cartes.

4.4.2. Evolutivité

Le principe de Java Beans a été choisi pour implémenter les composants d'Hypercarte. Un Java Bean est un composant qui suit un certain nombre de règles de codage afin de pouvoir être utilisé dans les environnements de développement intégré (IDE[52]) de type RAD[53] ou visuel. Le codage avec des branchements conditionnels, la dissémination de paramètres dans différentes classes, et surtout une communication intercomposant lourde posent des problèmes d'évolutivité du code.

4.4.2.1. Codage avec branchement conditionnel

Dans l'application Hypercarte, l'unique classe en charge de l'affichage de carte est le composant Java Bean `Chart`. Elle est capable d'afficher tout type de carte. Elle possède un attribut `type` dont la valeur permet de choisir le type de carte affichée. Le code de cette classe est écrit avec des branchements conditionnels (cf. code 4.2).

Code 4.2 : Structure des branchements conditionnels de la classe `Chart`

```
switch (type)
{
    case 0:
        // traitement spécifique à la carte de type 0
    case 1:
        // traitement spécifique à la carte de type 1
    // etc.
}
```

Un extrait du code de la classe `Chart` de la version 0.9 d'Hypercarte est disponible en annexe (cf. annexe 3).

Ce type de programmation présente deux inconvénients majeurs :

- le code spécifique à un type de carte est éparpillé dans les différents branchements conditionnels de la classe, posant des problèmes de maintenabilité ;
- la classe `Chart` doit être modifiée pour ajouter un type de carte à l'application, posant des problèmes d'évolutivité.

4.4.2.2. Paramètres disséminés

Utilisant des Java Beans, la version 0.9 d'Hypercarte répartit les données dans les classes de graphiques qui les manipulent. Ainsi, les paramètres de l'application sont stockés dans les composants graphiques qui permettent de modifier leur valeur. Par exemple :

[52] IDE (*Integrated Development Environment*) : environnement de développement intégré.
[53] RAD (*Rapid Application Development*) : développement rapide d'applications.

- le facteur de zoom est stocké dans la carte,
- l'espace d'étude est stocké dans le panneau des paramètres,
- le type de progression est stocké dans le panneau d'options.

Or, ces paramètres sont également utilisés par d'autres classes. Par exemple :

- le facteur de zoom est utilisé par le panneau d'affichage de l'échelle cartographique et par le panneau de légende,
- l'espace d'étude est utilisé par la carte et par le panneau de légende.
- le type de progression est utilisé par le panneau de légende et par la carte.

Par exemple, la classe `Chart` (carte) possède un pointeur sur l'instance des composants suivants : `PanelOption` (panneau de paramètres), `EditCaption` (panneau d'options) et `Histogram`. Ainsi, pour obtenir l'espace d'étude sélectionné par l'utilisateur, la carte utilise l'attribut `panelOption`, qui est de type `PanelOption,` et invoque la méthode `panelOption.getChainsSpaceEtude()`.

Ainsi, tous les composants graphiques doivent être instanciés pour que tous les paramètres soient instanciés. Il serait donc très difficile de supprimer un des composants graphiques, comme par exemple le panneau d'options.

De plus, la création de nouveaux composants peut nécessiter de coder l'accès à des paramètres situés dans plusieurs classes.

4.4.2.3. Communication intercomposant

4.4.2.3.1. Principe de l'écouteur

Le prototype utilise le principe d'écouteur (*listener*), également utilisé dans le package `java.awt.event`. Chaque composant réagissant à un événement doit auparavant connaître le composant à l'origine de cet événement et s'enregistrer auprès de lui. Ce principe est décrit dans la figure 4.9 et dans la figure 4.10.

Figure 4.9 : Enregistrement d'un écouteur auprès d'un composant émetteur d'événements

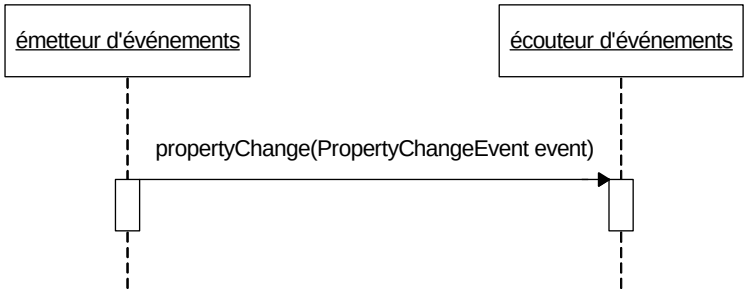


Figure 4.10 : Signalement d'un évènement à l'écouteur par le composant émetteur d'événements

4.4.2.3.2. Implémentation dans la version 0.9

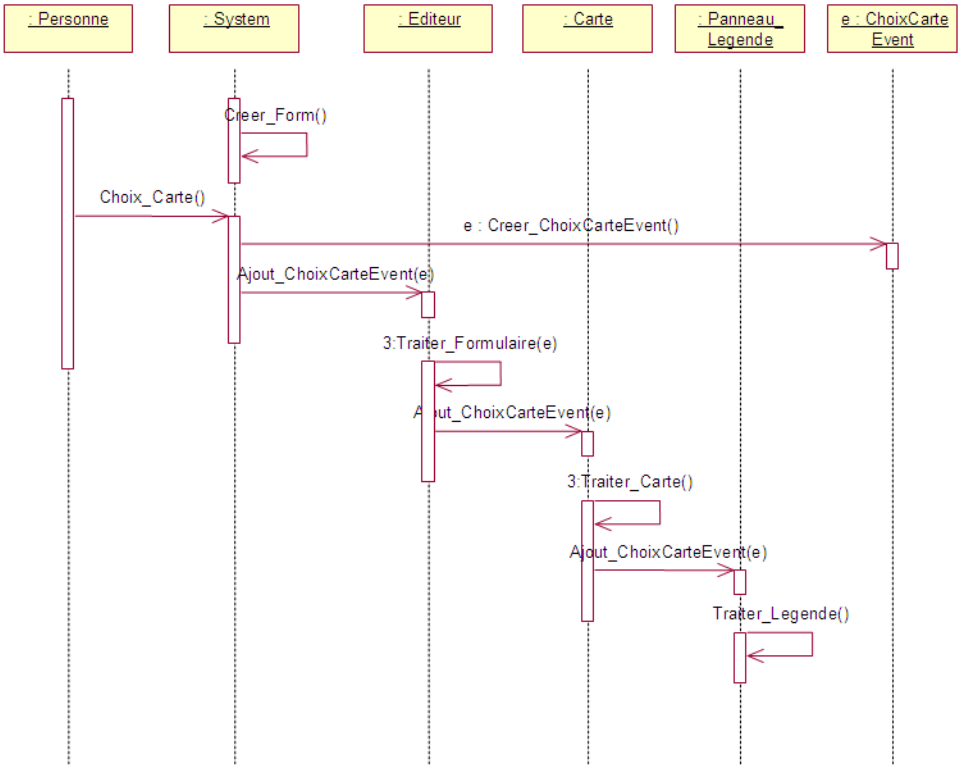


Figure 4.11 : La gestion événementielle en cascade de la version 0.9

L'utilisateur choisit une carte à afficher en cliquant sur l'onglet qui lui correspond et déclenche la procédure `Choix_Carte()` *du panneau d'onglets (*`System`*). Une instance de l'événement* `ChoixCarteEvent` *est créée. Le panneau à onglet déclenche l'événement* `ChoixCarteEvent` *sur le panneau d'options (*`Ajout_ChoixCarteEvent(e)`*). Celui-ci fait un traitement interne (*`Traiter_Formulaire(e)`*) puis déclenche l'événement* `ChoixCarteEvent` *sur le panneau d'options (*`Ajout_ChoixCarteEvent(e)`*). Ce dernier fait un traitement interne (*`Traiter_Carte(e)`*) puis déclenche l'événement* `ChoixCarteEvent` *sur le panneau de légende (*`Ajout_ChoixCarteEvent(e)`*). Le panneau de légende exécute un traitement interne (*`Traiter_Legende(e)`*).*
D'après [MART, 04].

Dans la version 0.9, ce principe a été choisi pour la communication entre les composants. Toutefois, ce principe ne garantit pas l'ordre d'appel des écouteurs lorsqu'un événement survient. Or, les composants dans lesquels sont répartis les paramètres doivent être mis à jour dans un ordre déterminé. Par exemple, le panneau de légende – qui détermine et stocke les valeurs des couleurs utilisées dans une carte – doit être mis à jour avant la carte – qui utilise

ces couleurs. Donc, cela interdit la propagation en masse d'un événement de son émetteur vers ses écouteurs. L'événement doit être transmis de composant à composant, chaque émetteur n'ayant qu'un seul écouteur, et chaque écouteur étant réémetteur de l'événement.

La figure 4.11 montre un exemple d'appel en cascade des composants, suite au choix d'une carte à afficher fait par l'utilisateur.

Nous avons vu dans le paragraphe 4.4.2.2 que les composants possèdent une forte interdépendance en ce qui concerne les données. C'est également le cas en ce qui concerne les événements.

Cela pose des problèmes d'évolutivité. Tous les composants doivent être instanciés, sous peine de rompre la chaîne d'appels en cascade.

4.4.3. Ergonomie

4.4.3.1. Prise en main par l'utilisateur

Hypercarte manipule des concepts cartographiques complexes. Or, la version 0.9 ajoute à cette complexité en offrant à l'utilisateur une interface amphigourique.

4.4.3.1.1. Interface non standard

Comme l'interface implémentée ne respecte pas certaines règles ergonomiques importantes [BAST, 93] comme l'homogénéité ou la lisibilité, l'utilisateur nécessite un long temps d'adaptation pour se familiariser avec l'application.

4.4.3.1.2. Modification dynamique des intitulés d'onglet

Les intitulés d'onglet contiennent les libellés des paramètres choisis (espace d'étude, maillage, déviations, etc.).

Par exemple, supposons que le maillage élémentaire sélectionné soit PECO[54] et que l'espace de référence pour la déviation globale soit l'Europe des Quinze, l'intitulé de l'onglet de la carte de déviation globale est « NUTS2/PECO » (cf. figure 4.12).

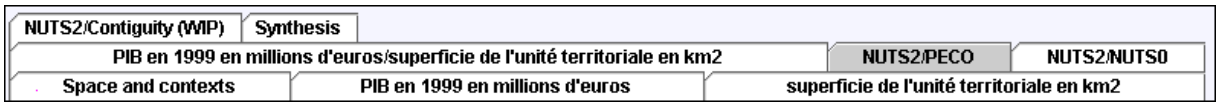


Figure 4.12 : L'onglet sélectionné est celui de la déviation globale

Si l'utilisateur choisit l'Europe des quinze comme nouvel espace de référence, l'intitulé de l'onglet devient alors « NUTS2/EUROPE_15 » et l'onglet change de taille et de position comme le montre la figure 4.13.

[54] PECO : Pays de l'Europe du Centre-Ouest.

Figure 4.13 : L'onglet de la déviation globale change de taille

Cette variation des largeurs d'onglet, les faisant se disposer sur plusieurs rangés (cf. figure 4.14) induit une perte de repères dans la fenêtre pour l'utilisateur. Cela l'oblige à relire les intitulés pour trouver la carte qu'il veut afficher.

4.4.3.1.3. Incohérence des libellés

Les termes employés dans l'interface ne sont pas homogènes et peuvent prêter à confusion. Par exemple, l'onglet « *Space and Contexts* » affiche les paramètres sélectionnés dans le panneau « *Space and zoning* », et non ceux du panneau « *Contexts of reference* ».

4.4.3.2. Taille de la carte

La carte, qui est le composant graphique central de l'application, occupe une surface réduite dans la fenêtre, lorsque cette dernière occupe une surface de 800 pixels par 600.

Notons également qu'avec cette taille de fenêtre, la légende peut être tronquée comme nous le montre la figure 4.14.

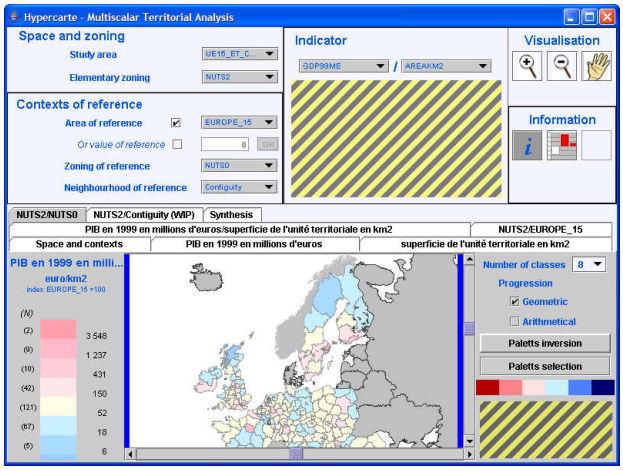


Figure 4.14 : Interface de la version 0.9 en mode 800×600

Les zones hachurées mettent en évidence les surfaces inutilisées.

4.4.3.3. Structure de la fenêtre

La fenêtre de la version 0.9, lorsqu'elle a la taille minimum requise de 800×600 (cf. figure 4.14), présente des zones non utilisées. La figure 4.15 montre qu'avec des résolutions plus grandes, ces zones augmentent sensiblement en taille.

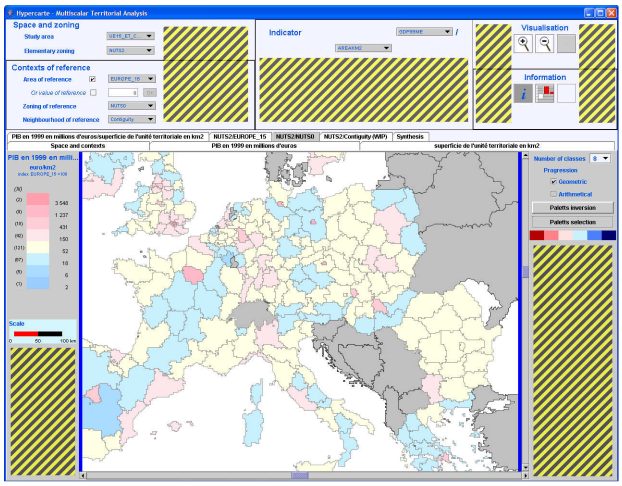


Figure 4.15 : Interface de la version 0.9 en mode 1250×1024

4.4.4. Bogues

4.4.4.1. Problème d'arrondi

Dans la version 0.9, les valeurs de stock et les rapports entre deux valeurs de stock sont tronqués à l'affichage : seule le partie entière est visible (cf. figure 4.16 et figure 4.17). Lorsque le dénominateur est supérieur au numérateur, le rapport entre les deux est compris entre 0 et 1. Or, arrondir un tel rapport à une valeur entière est excessif et produit un résultat faussé.

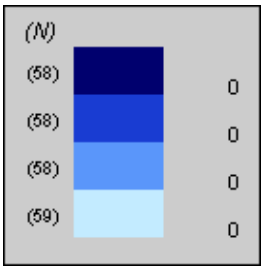


Figure 4.16 : Légende affichant des valeurs arrondies

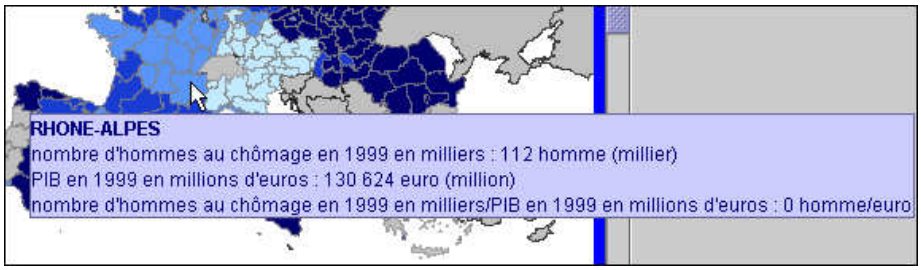


Figure 4.17 : Fenêtre d'information contextuelle affichant une valeur arrondie

Ces deux composants graphiques de la version 0.9 affichent des valeurs erronées. Cette légende et cette fenêtre d'information correspondent à la carte de rapport entre d'une part le nombre (en milliers) d'hommes au chômage en 1999 et d'autre part le produit intérieur brut en 1999 en millions d'euros. La valeur maximum du numérateur est de l'ordre de 300, celle du dénominateur de 400 000. La plage de valeur du rapport est comprise entre 0 et 1, et la valeur arrondie à 0.

4.4.4.2. Mauvaise implémentation de la gestion événementielle

Les appels entre composants par déclenchement d'événements sont mal implémentés dans la version 0.9. La mise en place d'une trace révèle que, après une action de l'utilisateur, certains traitements sont effectués plusieurs fois (cf. figure 4.18).

```
Scale.MapChosen()
```

```
Chart.MapChosen()
HyperApplet.fireChoiceChart()
EditCaption.ChoiceChart()
EditCaption.fireChoiceNbClass()
PanelCaption.ChoiceNbClass(EditCaptionEvent)
PanelCaption.updateColors()
PanelCaption.updateValues()
// Première série d'appels -----------------------------
PanelCaption.fireModification(DistributionUpdateEvent)
Chart.modification(DistributionUpdateEvent e)
Chart.calculateDistribution()
Chart.fireModification(DistributionUpdateEvent e)
PanelCaption.modification(DistributionUpdateEvent)
PanelCaption.UpdateBriques()
// Fin de la première série d'appels --------------------
EditCaption.ajoutPalettsSimple()
EditCaption.fireChoiceChart()
Chart.ChoiceChart(ChoiceChartEvent e)
Chart.fireChoiceChart()
PanelCaption.ChoiceChart()
PanelCaption.updateColors()
// Seconde série d'appels ------------------------------
PanelCaption.fireModification(DistributionUpdateEvent)
Chart.modification(DistributionUpdateEvent e)
Chart.calculateDistribution()
Chart.fireModification(DistributionUpdateEvent e)
PanelCaption.modification(DistributionUpdateEvent)
PanelCaption.UpdateBriques()
// Fin de la seconde série d'appels --------------------
Chart.paint()
```

Figure 4.18 : Trace générée par la version 0.9 après un clic de l'utilisateur sur un onglet.

Après un clic de l'utilisateur sur le troisième onglet, les appels en cascade visualisés grâce à cette trace montre que six méthodes sont invoquées chacune deux fois, avant que l'application ne rende la main à l'utilisateur.

4.4.4.3. Carte de synthèse incorrect

La carte de synthèse est incorrecte en raison de problèmes algorithmiques dans le calcul des contiguïtés.

4.5. Synthèse

La version 0.9 répond aux spécifications et aux principales problématiques adressées par les géographes du projet Hypercarte. Toutefois, un certain nombre de points d'amélioration, dont certains très critiques, sont identifiés. Ils touchent aux performances de l'application, à son évolutivité, ainsi qu'à son ergonomie. Ils constituent les principales évolutions apportées dans la version 1.0.

Le tableau 4.1 met en relation les points d'amélioration les plus importants identifiés dans la version 0.9, avec les solutions proposées et implémentées dans la version 1.0 pour y répondre.

Type	Points d'amélioration identifiés dans la version 0.9		Solutions apportées dans la version 1.0	
	Section	**Description**	**Section**	**Description**
	4.4.4.1	Problème d'arrondi	5.5.4	Méthode d'arrondi adaptatif
	4.4.4.2	Mauvaise implémentation de la gestion événementielle	5.2.4	Isolement de la logique applicative
	4.4.2.3	Communication intercomposant	5.2.2	Communication inter-composant événementielle anonyme
	4.4.2.2	Paramètres disséminés	5.2.3	Regroupement des paramètres
	4.4.2.1	Codage avec branchement conditionnel	5.4.1.2.2	Spécialisations de classe
	4.4.1.1	Une seule instance de carte	5.4.1.2.2	Spécialisations de classe
	4.4.1.2	Absence de cache	5.6.2.1 5.6.2.2	Mise en cache des UT filtrées Mise en cache des calculs intermédiaires
	4.4.1.3	Définition inadéquate des contours cartographiques	5.6.1.2	Adaptation de la définition des contours cartographiques
	4.4.3.1	Prise en main par l'utilisateur	5.7.2.3	Critère « Prise en compte de l'expérience de l'utilisateur »
	4.4.3.3	Structure de la fenêtre	5.7.1	Restructuration
	4.4.3.2	Taille de la carte	5.7.1.4	Amélioration du rapport surfacique

Tableau 4.1 : Points d'amélioration identifiés dans la version 0.9 et solutions apportées dans la version 1.0

Les types de problèmes sont les suivants : bogue ( ), problème d'évolutivité ( ), problème de performance ( ), problème d'ergonomie ( ).

Chapitre 5 – Réalisation de la version 1.0 d'Hypercarte

Nous présentons dans ce chapitre les corrections, évolutions, améliorations et nouveautés apportées au logiciel Hypercarte. Les tâches sont réparties entre les acteurs du projet de la manière suivante :

- L'analyse et la conception de la partie de l'application traitant des données – format des fichiers textuels, format de stockage interne, sérialisation, agrégation, désérialisation, calcul des contiguïtés – sont menées par l'équipe du projet Apache. Nous apportons ponctuellement notre concours à la réalisation de l'analyse de cette partie de l'application.

- Les données brutes – les contours et les valeurs de stock des UTE – et l'expertise géographique sont fournies par l'équipe PARIS.

- La conception et la réalisation de l'interface utilisateur sont de notre responsabilité.

5.1. Conception

5.1.1. Choix de la technique

Etant donnés l'existant et les contraintes techniques, les deux solutions techniques envisagées sont l'utilisation de SVG d'une part, et des bibliothèques graphiques standard de Java d'autre part.

Lors de tests réalisés sur des volumes de données équivalents à ceux utilisés dans la version 0.9, la première solution affiche des temps de chargement relativement long. Or, les besoins de l'utilisateur sont de pouvoir faire beaucoup de rechargements de cartes dans le but de mettre au point ces dernières.

Bien que standard – et donc plus pérenne, et malgré les attraits de la bibliothèque Batik, SVG n'a pas été retenu. En effet, l'amélioration de la solution éprouvée dans la version 0.9 semblait plus prometteuse.

5.1.2. Choix de la méthode de travail

Notre problématique est la suivante : faut-il tout réécrire de A à Z en s'inspirant de la version 0.9, ou doit-on adapter le code de la version 0.9 en le faisant évoluer en douceur ?

Nous avons opté pour la seconde solution. En effet, avec des échéances en cours d'année, où nous devons être capable de présenter des versions intermédiaires du logiciel, la seconde alternative nous semble plus prudente, bien que plus coûteuse en temps de développement.

5.1.3. Architecture

L'architecture globale du logiciel reste inchangée par rapport à la version 0.9 d'Hypercarte (cf. figure 4.1).

L'architecture du code, entièrement remaniée, suit un découpage trois tiers (figure 5.1) avec trois types de classes :

- Les classes graphiques, correspondant aux composants graphiques visibles dans l'interface.
- Les classes techniques, qui sont des outils qui permettent l'accès aux données, mais également le stockage des paramètres, les échanges entre les composants, etc.
- Les classes métier, qui regroupe la logique de l'application et ses règles métiers, pour connaître, par exemple, les cartes à recalculer suite à la modification d'un paramètre donné.

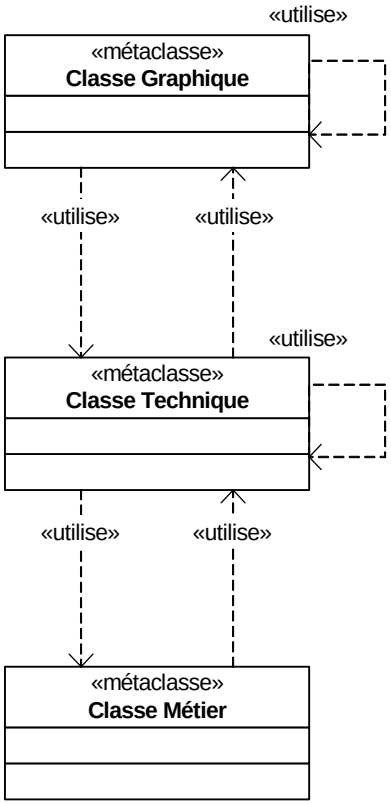


Figure 5.1 : Diagramme de classes des l'architecture logicielle

5.2. Principes mis en œuvre

5.2.1. Principes généraux

5.2.1.1. Indices

La version 0.9 n'instancie qu'une carte, qu'une légende et qu'un panneau d'options. L'attribut `type` de chaque instance est modifié en fonction de la carte à afficher.

La version 1.0 possède autant d'instances de ces classes que de cartes à afficher. La valeur de l'attribut `mapIndex` de chaque instance est fixée à la création de l'instance. Lorsque l'utilisateur clique sur un onglet pour demander une nouvelle carte, seules les instances dont l'attribut `mapIndex` est égal à l'indice de l'onglet, sont affichées.

5.2.1.2. Portée des paramétrages

Comme dans la version 0.9, la portée des paramétrages peut être de deux types :

- soit globale à l'application, comme pour le facteur de zoom de carte, le vecteur de déplacement de carte, les stocks sélectionnés, l'espace d'étude, etc.
- soit limitée à une carte, comme pour la taille et la couleur des disques, les couleurs et le nombre d'éléments de la palette, le type de progression, etc.

5.2.1.3. Instance unique

Les classes graphiques de l'application partagent des données, des règles, des comportements ou encore des événements, par l'intermédiaire de classes dédiées (les classes techniques et les classes métiers).

Ces classes dédiées ne sont instanciées qu'une seule fois. De plus, leurs instances sont persistantes. Nous utilisons pour cela une instance privée et persistante de la classe. Cette instance est créée lors du premier appel à la méthode persistante `getInstance()` et est retournée à chaque appel à cette méthode (cf. code 5.1 et code 5.2).

Code 5.1 : Extrait de la classe `hypercarte.Logic` utilisant le principe d'instance unique

```
public class Logic implements IGlobalEventListener, IIndexedEventListener {

    private static Logic logic = null;

    public static Logic getInstance() {
        if (logic == null) {
            logic = new Logic();
        }
        return logic;
    }

    // ...
}
```

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 réinstancier ces classes très utilisées dans l'application.

Code 5.2 : Exemple d'appel à des méthodes des classes dédiées `Logic` et `Settings`

```
Vector unitsAtElementaryZoningLevelInStudyArea =
            Logic.getInstance().getUnits(
                        Zoning.getZoningMax(),
                        hypercarte.data.Area.getArea(
                                    Settings.getInstance().getStudyAreaName()
                                    )
                        );
```

5.2.2. Communication inter-composant événementielle anonyme

5.2.2.1. Principe

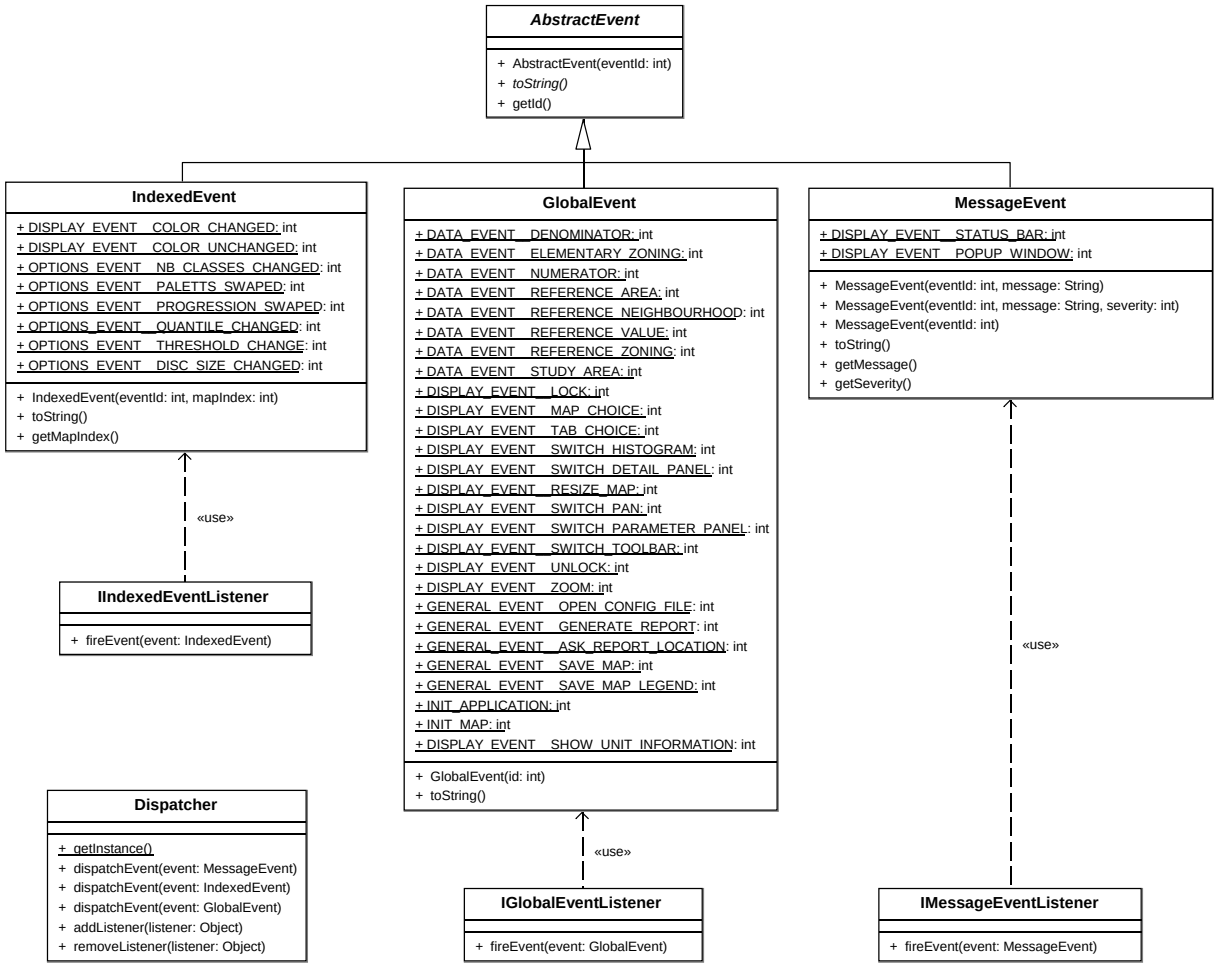


Figure 5.2 : Diagramme des classes et des interfaces implémentant le principe de dispatcher *d'événements*

Nous définissons le dialogue inter-composant comme étant un ensemble d'échanges entre des instances de classes d'Hypercarte. Nous excluons du dialogue inter-composant les instances

de classes étant définies à l'intérieur de classes d'Hypercarte. Par exemple, les instances des classes `JCombobox` ou `JPanel` – classes Java standard – et les instances de la classe Java `hypercarte.ui.Legend.Row` – classe `Row` définie dans la classe `Legend` – ne sont pas concernées.

La communication inter-composant événementielle vise à établir une méthode de propagation des événements entre un émetteur et des écouteurs.

Le but de la communication inter-composant événementielle anonyme (CICEA) est de rendre impersonnelle la propagation des événements, l'émetteur ne connaissant pas son audience et l'écouteur ignorant les sources des événements

Le paquetage `hypercarte.event` comprend tous les objets de la CICEA (cf. figure 5.2).

5.2.2.2. Trois types d'événements

Nous distinguons trois types d'événements : les événements indicés, les événements de messagerie et les événements globaux.

5.2.2.2.1. Evénements indicés

Comme nous l'avons vu dans le paragraphe 5.1.2, la version 1.0 d'Hypercarte instancie certains composants graphiques – légendes, panneau d'options de carte, zones d'affichage de carte, etc. – autant de fois qu'elle affiche de cartes différentes. Lorsqu'une option est modifiée pour une carte, seules les composants graphiques liés à cette dernière doivent être notifiés de cet événement. Il faudrait alors autant de types d'événement qu'il existe de cartes, ce qui rendrait notre implémentation dépendante du nombre de cartes affichées.

En associant un indice à chaque carte, nous ne gérons qu'un seul type d'événement pour les cartes : un événement indicé. Ainsi, toutes les cartes et toutes les légendes sont notifiées et seules celles qui ont un indice correspondant à celui de l'événement réagissent.

Les événements indicés sont créés lorsque l'utilisateur modifie la palette de couleurs d'une carte, la taille des disques d'une carte, le type de progression cartographique, la valeur seuil de la carte de synthèse, etc.

Les événements indicés sont des instances de la classe `IndexedEvent`.

5.2.2.2.2. Evénements de messagerie

Il existe deux canaux de communication pour informer l'utilisateur du résultat de ses actions : la barre de statut et les boîtes de dialogue. Nous utilisons donc deux événements de messagerie : le premier pour afficher des messages dans la barre de statut, le deuxième pour les afficher dans des boîtes de dialogue. Le premier événement est écouté par la barre de statut. Les boîtes de dialogue (composant `javax.swing.JDialog`) sont par défaut centrées par rapport au composant graphique qui les appelle. Nous décidons donc que l'écouteur du deuxième événement est l'élément graphique de plus haut niveau de l'application (composant étendant la classe `javax.swing.JApplet`).

Un événement de messagerie possède trois attributs spécifiques : le message, le type de message (message d'erreur, message d'alerte ou message informatif) et la criticité du message.

Les événements de messagerie sont des instances de la classe `MessageEvent`.

5.2.2.2.3. Evénements globaux

Les événements qui ne sont pas des événements de messagerie et dont les écouteurs peuvent ne pas être indicés, sont globaux. Ils s'adressent à tout type de composant d'Hypercarte.

Les événements globaux sont déclenchés lorsque l'utilisateur : modifie la valeur d'un paramètre, change de carte, navigue entre les onglets de légende, d'options et d'explication, masque un composant (barre d'outils, panneau de paramètres), restaure un espace de travail, génère le rapport, etc.

Les événements globaux sont des instances de la classe `GlobalEvent`.

5.2.2.3. Interfaces

Une interface peut être vue comme un cahier des charges minimum pour une classe. Nous pouvons dire d'une classe qu'elle implémente une interface lorsqu'elle implémente toutes les méthodes et tous les attributs décrits dans cette interface.

A chaque type d'événement correspond une interface Java : `IIndexedEventListener`, `IMessageEventListener`, `IGlobalEventListener`.

Ainsi, un écouteur doit implémenter une des trois interfaces correspondant aux types d'événements précédemment cités. Les écouteurs seront avertis d'un événement par l'invocation de la méthode `fireEvent()` définie par chaque interface.

5.2.2.4. Dispatcher d'événements

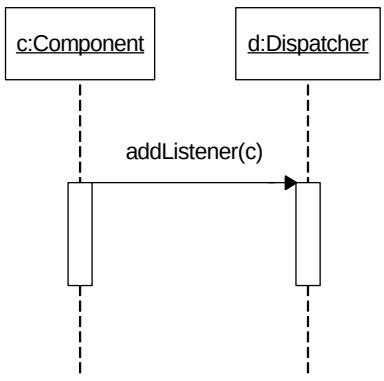


Figure 5.3 : Diagramme de séquence : enregistrement d'un écouteur

Les écouteurs – instances de classes d'Hypercarte souhaitant être notifiées lorsqu'un événement survient – s'abonnent au préalable auprès du dispatcher par la méthode `addListener()`.

Le *dispatcher* d'événements [BISS, 04] est le composant central de la gestion événementielle d'Hypercarte : il collecte et redistribue les événements. Il utilise le principe d'instance unique défini dans le paragraphe 5.2.1.3. Il est connu par tous les composants d'Hypercarte. Pour être notifiés d'un événement, les écouteurs doivent s'enregistrer auprès de lui (cf. figure 5.3).

Le *dispatcher* d'événements gère une liste d'écouteurs par interface (cf. code 5.3).

Code 5.3 : Les listes d'écouteurs de la classe `hypercarte.event.Dispatcher`

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

L'invocation de la méthode `addListener(Object)` enregistre le composant passé en paramètre, dans les listes correspondant aux interfaces qu'il implémente. Par exemple, les instances de la classe `DiscLegend`, à l'écoute des événements globaux et indicés, n'appellent qu'une fois cette méthode pour s'enregistrer (cf. code 5.4). La méthode `removeListener(Object)` ôte le composant des listes correspondant aux interfaces qu'il implémente.

Code 5.4 : La méthode `addListener()` de la classe `Dispatcher`

```
/**
 * 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);
      }
}
```

Lorsqu'un composant veut propager un événement, il invoque la méthode `dispatchEvent(Event)` – où le paramètre `Event` est une instance d'une des trois classes d'événements. Le *dispatcher* va invoquer la méthode `fireEvent(Event)` de chaque composant de la liste correspondant au type d'événement passé en paramètre (cf. figure 5.4 et code 5.5).

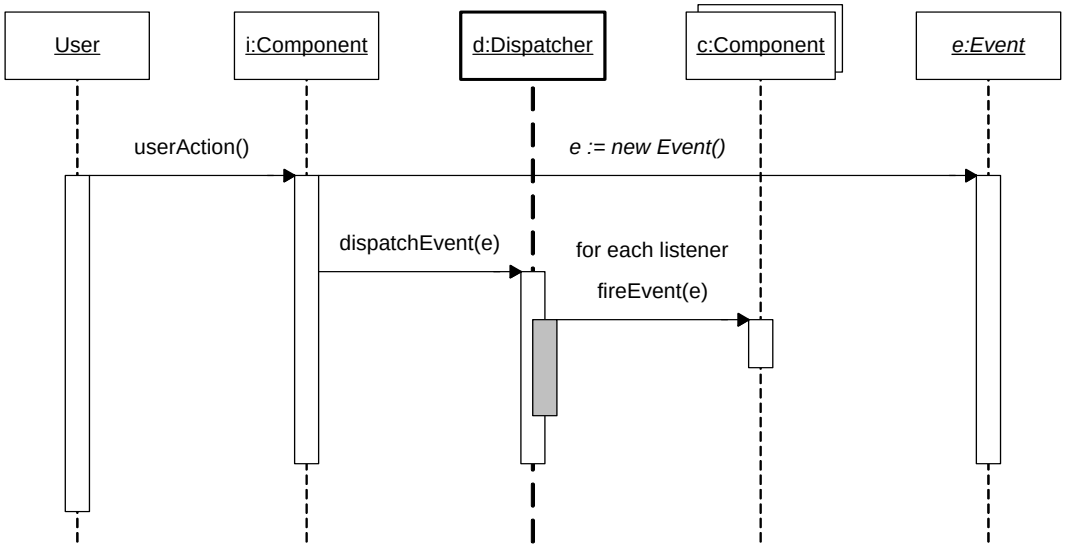


Figure 5.4 : Diagramme de séquence : propagation d'un événement

L'instance `i` d'une classe d'Hypercarte est prévenue d'une action de l'utilisateur lorsqu'une méthode `userAction()` est invoquée. Cette instance `i` crée un événement `e`, instance de la classe `IndexedEvent`, `MessageEvent` ou `GlobalEvent`. L'instance `i` invoque ensuite la méthode `dispatchEvent(e)` du dispatcher `d`. Le dispatcher propage alors cet événement vers chaque écouteur enregistré grâce à la méthode `fireEvent(e)`.

Code 5.5 : La méthode `dispatchEvent()` de la classe `Dispatcher`

```
/**
 * An action occured, warn all the registered listeners.
 *
 * @param event
 *            An event constant number
 */
public void dispatchEvent(IndexedEvent event) {

	// Warn the HC logic that an event was fired
	this.logic.fireEvent(event);

	Iterator listenerIterator = this.indexedEventListeners.iterator();
	IIndexedEventListener eventListener;

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

Le dispatcher d'événements est l'instance de la classe `Dispatcher`.

5.2.3. Regroupement des paramètres

Dans la version 0.9, les paramètres sont stockés dans les composants qui les représentent graphiquement. Par exemple, la variable contenant le facteur de zoom est stockée dans le

composant carte, la variable contenant l'espace d'étude est stockée dans le panneau de paramètres. Or les variables sont généralement utilisées par plusieurs composants, ce qui implique une communication entre le composant qui contient une variable et les composants qui utilisent cette variable.

Dans le paragraphe 0, nous décrivons la CICEA mise en place dans la version 1.0. Cette communication est anonyme, ce qui signifie que les composants n'ont pas connaissance les uns des autres. Pour partager les paramètres, il existe deux solutions :

- soit l'événement porte le paramètre dont il propage le changement de valeur,
- soit les composants partagent les paramètres.

Nous avons opté pour la seconde solution qui semble plus souple et plus évolutive (cf. figure 5.5).

Les paramètres sont stockés dans les deux classes `hypercarte.config.Settings` et `hypercarte.config.Map` : la première contient les paramètres globaux comme le facteur de zoom ou l'espace d'étude ; la seconde stocke les paramètres spécifiques à chaque carte comme la taille des disques ou la palette de couleurs. Il existe autant d'instances de la classe `hypercarte.config.Map` que de cartes.

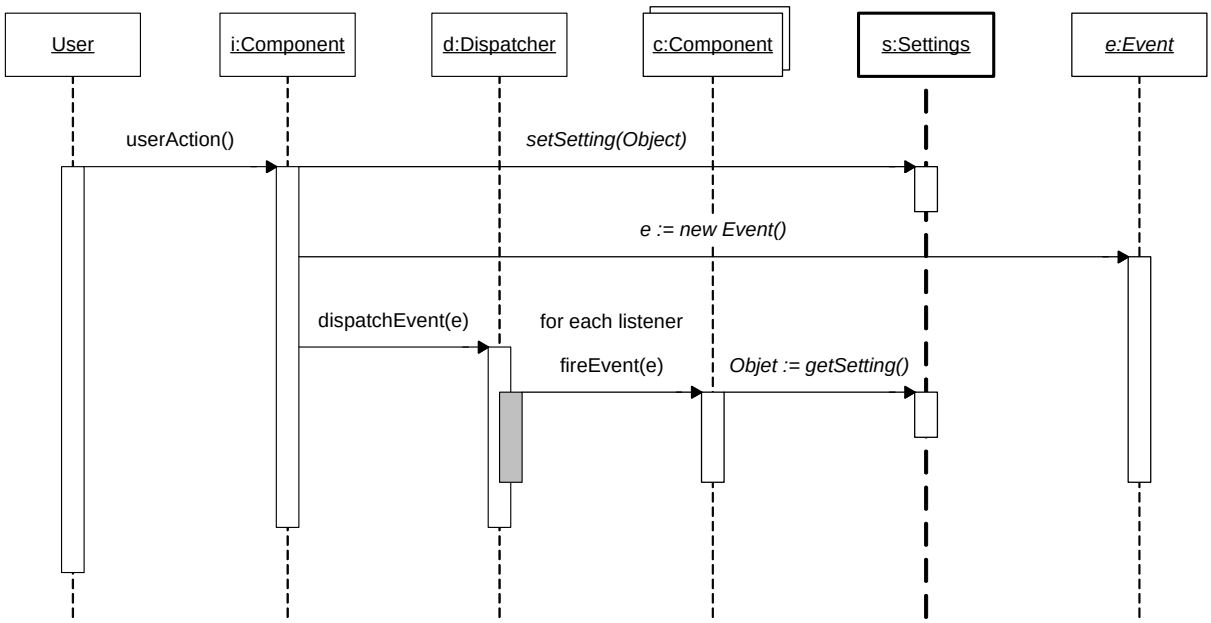


Figure 5.5 : Diagramme de séquences : intégration de la classe `Settings`

5.2.4. Isolement de la logique applicative

Dans Hypercarte, les mêmes calculs doivent être effectués en réaction à différents événements et pour différentes cartes.

Dans le but de centraliser cette logique applicative, nous avons créé la classe métier `hypercarte.Logic`. Son instance est systématiquement prévenue par le dispatcher lorsqu'un événement survient, en invoquant la méthode `fireEvent()` correspondant au type d'événement (cf. figure 5.6).

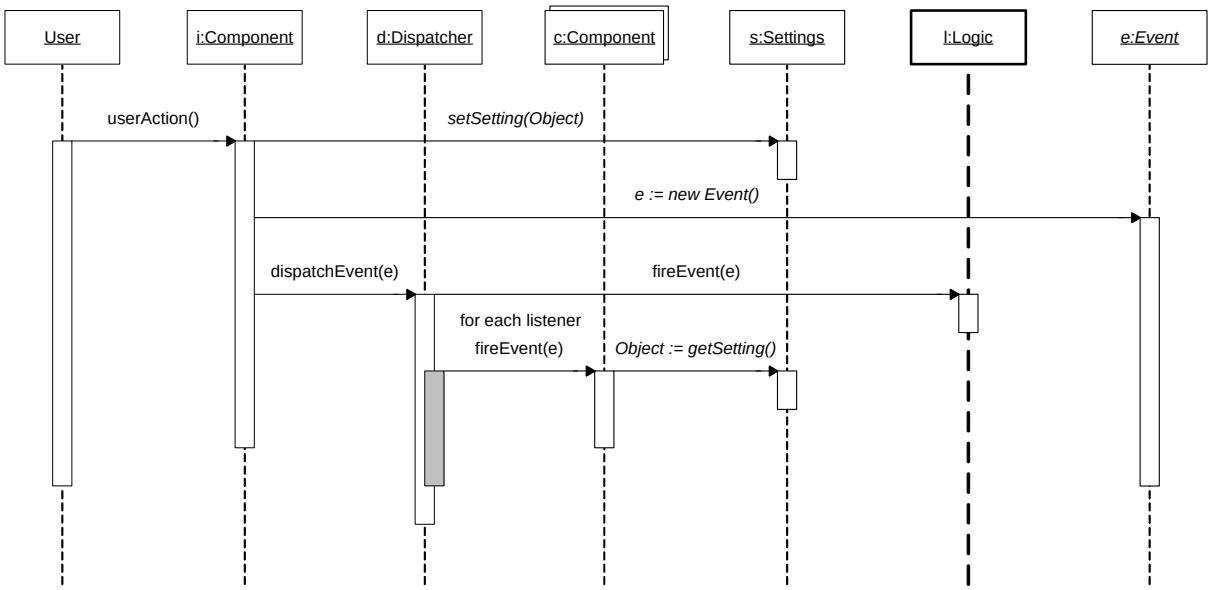


Figure 5.6 : Diagramme de séquences : intégration de la logique métier

La classe `Logic` se charge alors d'effectuer les calculs nécessaires (cf. code 5.6), évitant les calculs redondants et rendant plus facile l'utilisation de caches (cf. paragraphes 5.6.2 et 5.6.2.2).

Code 5.6 : Extrait de la méthode `hypercarte.Logic.fireEvent(GlobalEvent e)`

```
    public void fireEvent(GlobalEvent event) {

        if (Settings.getInstance().isBubbleCancelled())
            return;

        switch (event.getId()) {

        case GlobalEvent.INIT_APPLICATION:

            // Update unit list
            UnitRepository.getInstance().updateContextUnitList();
            break;

        case GlobalEvent.GENERAL_EVENT__OPEN_CONFIG_FILE:

            // Update data lists
            UnitRepository.getInstance().updateContextUnitList();

            // All maps are not up-to-date anymore
            setUpToDateAll(false);

            // Compute current map's values
            computeIfNeeded(Settings.MAP_CONTEXT);
            computeIfNeeded(Settings.MAP_NUMERATOR);
            computeIfNeeded(Settings.MAP_DENOMINATOR);
            computeIfNeeded(Settings.MAP_INDICATOR);
            computeIfNeeded(Settings.MAP_GLOBAL_DEVIATION);
```

```
            computeIfNeeded(Settings.MAP_MEDIUM_DEVIATION);
            computeIfNeeded(Settings.MAP_LOCAL_DEVIATION);
            break;

        // ...
}
```

Si l'événement, dont l'instance de la classe `Logic` est notifiée, correspond à l'initialisation de l'application, alors le cache des unités est mis à jour en prenant les UT de l'espace d'étude qui sont au niveau de maillage (cf. paragraphe 5.6.2.1). Si l'événement est la restauration d'un espace de travail (cf. paragraphe 5.5.2), le cache des unités est mis à jour, les cartes sont marquées comme non à jour, puis recalculées si nécessaire.

5.2.5. Exemple détaillé

Dans la figure 5.7, nous présentons un exemple partiel détaillé des principes mis en œuvre dans la version 1.0.

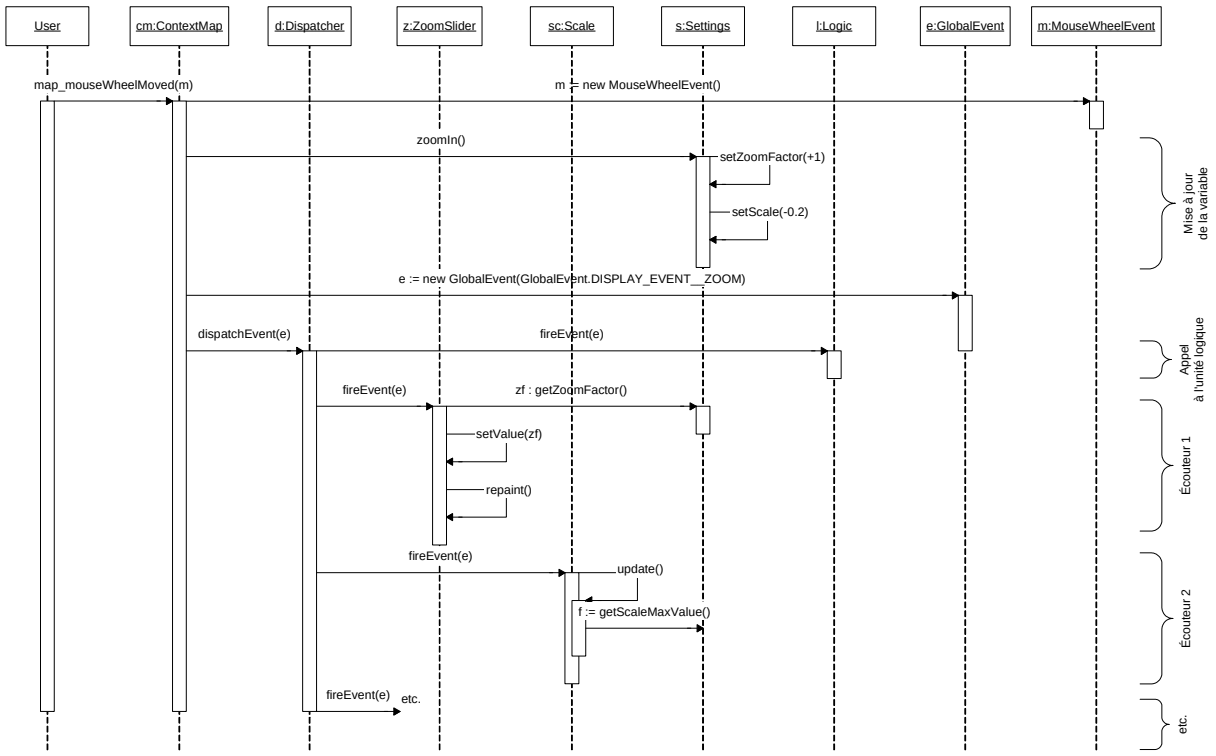


Figure 5.7 : Diagramme de séquence : zoom avant par la molette de la souris de l'utilisateur

Dans le cas d'un zoom, l'unité logique ne fait aucune action spécifique.
L'exemple est tronqué. En effet, en plus du curseur de zoom (classe `ZoomSlider`), quatre classes graphiques d'Hypercarte réagissent à un zoom : la légende de carte à disque proportionnel (`DiscLegend`), la carte (`DiscMap`), la barre de menus (`MenuBar`) et l'échelle (`Scale`).

5.3. Structure de données

La structure des données territoriales et attributaires est identique à celle de la version 0.9.

Dans la version 0.9, les données géométriques sont fournies dans un fichier qui met en relation le code de l'UT et la liste des coordonnées géographiques. Dans le version 1.0, nous

conservons ce principe. De plus, dans le but de rendre l'application accessible, nous adoptons un format très répandu, constitué d'une paire de fichiers MIF/MID [MAPI, 03].

5.4. Restructuration du code

Pour nous aider à restructurer le code de la version 0.9, nous nous sommes appuyé sur un IDE. Après avoir testé JBuilder® de Borland® et NetBeans, nous avons opté pour l'utilisation d'Eclipse [ECLI, 04]. Eclipse est un IDE open source complet, ergonomique, relativement léger et très ouvert grâce au support de *plug-ins*. Nous avons utilisé le *plug-in* EclipseUML Studio de Omondo [OMON, 04].

5.4.1. Réorganisation des classes

En programmation orientée objet, les modules de programmation élémentaires sont appelés des classes. Nous avons réorganisé les classes de la version 0.9 en les regroupant en paquetage ou en les redéfinissant grâce au mécanisme d'héritage.

5.4.1.1. Création de paquetages

Les paquetages sont des regroupements logiques de classes. Dans la version 0.9, les classes sont réparties dans deux paquetages : `hypercarte.data` et `hypercarte.interfac`. Les classes de la version 1.0 étant plus nombreuses que dans la version précédente, nous avons créé de nouveaux paquetages.

Le paquetage `hypercarte.data` a été conservé. Il contient les classes de stockages et d'agrégation des données géographiques et attributaires, les classes permettant le sérialisation et la désérialisation et les classes d'accès aux fichiers textuels et au fichier sérialisé.

Le paquetage `hypercarte` possède les deux classes les plus importantes de l'application : la classe `Starter` qui contient la méthode `main()`, et la classe `Logic` qui contient les règles métiers (cf. paragraphe 5.2.4 - Isolement de la logique applicative).

Le paquetage `hypercarte.config` contient les classes de stockage des paramètres de l'application (cf. paragraphe 5.2.3 - Regroupement des paramètres).

Le paquetage `hypercarte.event` comprend les classes et les interfaces liées à la gestion des événements dans Hypercarte.

Le paquetage `hypercarte.io` contient les classes de manipulation de fichiers XML permettant de sauvegarder et de restaurer l'espace de travail et les classes de création de fichiers XHTML utilisées pour la génération de rapport.

Le paquetage `hypercarte.ui` regroupe toutes les classes de composants graphiques comme les différentes cartes, les légendes, les différents types de panneaux d'options, l'histogramme, la barre de menus, etc.

Les classes de composants graphiques élémentaires comme les boutons, les listes déroulantes, les panneaux indexés (cf. paragraphe 5.2.1.1 - Indices), les panneaux à titre, etc. sont rassemblées dans le paquetage `hypercarte.ui.components`.

Le paquetage `hypercarte.misc` contient les classes inclassables dans les paquetages précédemment cités.

5.4.1.2. Redéfinition de classes

Les classes sont constituées d'attributs (les donnés) et de méthodes (les fonctions d'accès aux données). Le mécanisme d'héritage permet de lier des classes. Par exemple, une classe B qui hérite d'une classe A possède les mêmes attributs et les mêmes méthodes que A sans qu'il soit nécessaire de les redéfinir. Des attributs et des méthodes supplémentaires peuvent être définis dans B. Des méthodes de A peuvent être redéfinies dans B : nous parlons alors de surcharge de méthodes. La relation d'héritage est bijective. La classe B est une spécialisation de la classe A ; la classe A est une abstraction de la classe B.

5.4.1.2.1. Abstractions de classe

Pour la version 1.0 d'Hypercarte, nous réalisons deux abstractions de classes : l'une pour les cartes, l'autre pour les composants indicés.

Abstraction de la carte

Il nous semble intéressant de donner à l'utilisateur la possibilité de modifier l'agrégation territoriale dans un but prospectif : par exemple, changer la région d'appartenance d'un département français, créer des régions transnationales, etc. Une telle agrégation peut ensuite être étudiée à l'aide des fonctionnalités d'analyse multiscalaire pour valider son bien-fondé.

Cette modification de l'agrégation peut être obtenue à l'aide de deux cartes. La première présente à l'utilisateur les UT à un niveau de maillage *m* et la seconde à un niveau de maillages *m-1*. La première carte permet de sélectionner l'UT à déplacer, la seconde carte de choisir la nouvelle UTS de l'UT sélectionnée.

Ces deux cartes diffèrent des cartes d'analyse multiscalaire d'Hypercarte. Ces dernières sont issues de la classe `hypercarte.ui.Map`. Cette classe possède des méthodes autorisant la capture d'événements liés à la souris comme :

- le déplacement de la souris, qui permet de connaître l'UT survolée par le pointeur et d'afficher des informations relatives à cette UT ;
- l'utilisation des boutons (clic), dans le but de connaître l'UT sur laquelle l'utilisateur a cliqué et d'afficher un histogramme de synthèse des déviations de cette UT ;
- le glissement (ou « cliquer-déplacer »), pour mesurer le vecteur de déplacement lors d'un glissement de souris et pour translater la carte en conséquence ;
- l'utilisation de la molette, afin de connaître le sens de rotation de la molette de la souris afin d'agrandir ou de réduire le facteur de zoom de la carte.

A partir de la classe `Map`, nous avons construit une classe abstraite `AbstractMap` possédant les attributs et méthodes nécessaires à l'affichage d'une carte (cf. figure 5.9). Elle possède plusieurs méthodes abstraites de capture des événements souris comme `map_mouseClick()` ou `map_mouseWheelMoved()`. Nous n'avons abstrait de la classe `Map` que le mécanisme de gestion événementiel de la souris, mais pour être plus complète, cette classe nécessiterait également une abstraction des méthodes de dessin des UT.

Ainsi, la classe `Map` hérite de la classe abstraite `AbstractMap` et implémente les méthodes de capture d'événements souris.

Faute de temps, la fonctionnalité de modification de l'agrégation territoriale n'a pas été ajoutée à la version 1.0 d'Hypercarte. Toutefois, les deux cartes nécessaires à sa réalisation peuvent être obtenues en créant une classe qui spécialise `Map` et qui implémente la méthode de capture des clics de souris.

Abstraction des composants indicés

Plusieurs classes héritant de la classe `javax.swing.JPanel` utilisaient un indice lié au numéro de carte. Nous avons créé une classe, héritant de `JPanel`, et possédant un indice comme attribut. Cette classe s'appelle `IndexedPanel`. Les classes utilisant un indice héritent désormais de la classe `hypercarte.ui.IndexedPanel` (cf. figure 5.8).

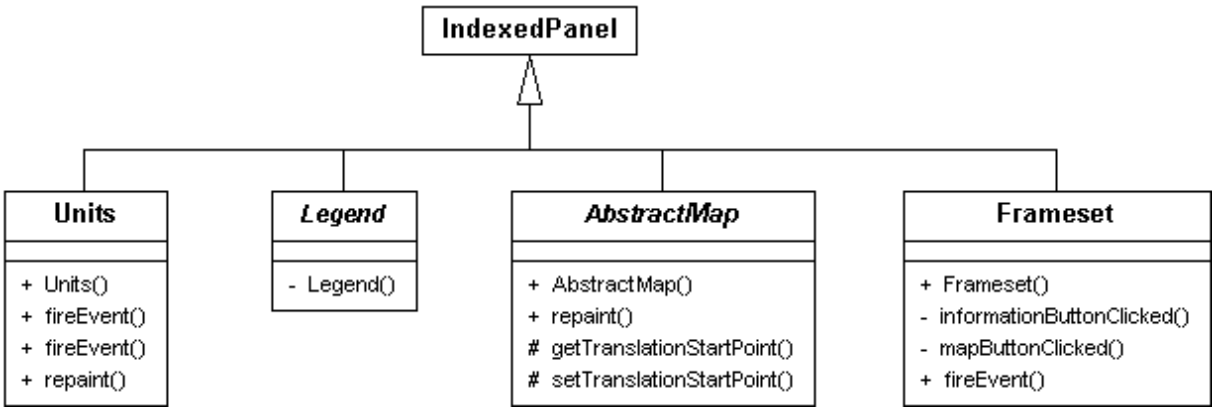


Figure 5.8 : Diagramme de classes : abstraction de la classe `IndexedPanel`

5.4.1.2.2. Spécialisations de classe

Deux nombreuses spécialisations sont faites dans la version 1.0 d'Hypercarte. Nous citons les plus significatives concernant les cartes, les légendes de carte et les options de carte. Nous ne détaillons que les spécialisations de cartes.

Spécialisations de carte

Pour pallier au problème adressé dans le paragraphe 4.4.2.1 (Codage avec branchement conditionnel), nous avons créé plusieurs classes spécialisant la classe `Map` : les classes `ContextMap` et `IndicatorMap` et les classes abstraites `DiscMap` et `DeviationMap`. `DiscMap` est spécialisée par les classes `NumeratorMap` et `DenominatorMap` ; `DeviationMap` est spécialisée par `GlobalDeviationMap`, `MediumDeviation`, `LocalDeviationMap` et `SynthesisMap` (cf. figure 5.9).

Un branchement conditionnel est nécessaire pour instancier la classe correspondant aux besoins (cf. code 5.7).

Code 5.7 : Utilisation de branchement conditionnel pour l'instanciation de classes spécialisées

```
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_DENOMINATOR:
        map = new DenominatorMap(this.mapIndex);
        break;
    case Settings.MAP_INDICATOR:
        map = new IndicatorMap(this.mapIndex);
        break;
    case Settings.MAP_GLOBAL_DEVIATION:
        map = new GlobalDeviationMap(this.mapIndex);
        break;
    case Settings.MAP_MEDIUM_DEVIATION:
        map = new MediumDeviationMap(this.mapIndex);
        break;
    case Settings.MAP_LOCAL_DEVIATION:
        map = new LocalDeviationMap(this.mapIndex);
        break;
    case Settings.MAP_SYNTHESIS:
        map = new SynthesisMap(this.mapIndex);
        break;
    default:
        map = new ContextMap(this.mapIndex);
        break;
}
```

La classe `Map` possède les méthodes permettant de dessiner la frontière et la surface des UT. Elle contient également des méthodes ensemblistes pour dessiner les UT de l'espace d'étude. La méthode `paintMap()`, invoquée à l'affichage d'une instance de `Map`, dessine le fond de carte (i.e. toutes les UT ne faisant pas parties de l'espace d'étude) et les UT de l'espace d'étude.

La méthode `paintMap()` de la classe `ContextMap`[55] surcharge `Map.paintMap()` en ajoutant le dessin des frontières des UT de plus haut niveau (avec le jeux de données que nous utilisons, cela correspond aux pays).

[55] Notée `ContextMap.paintMap()`.

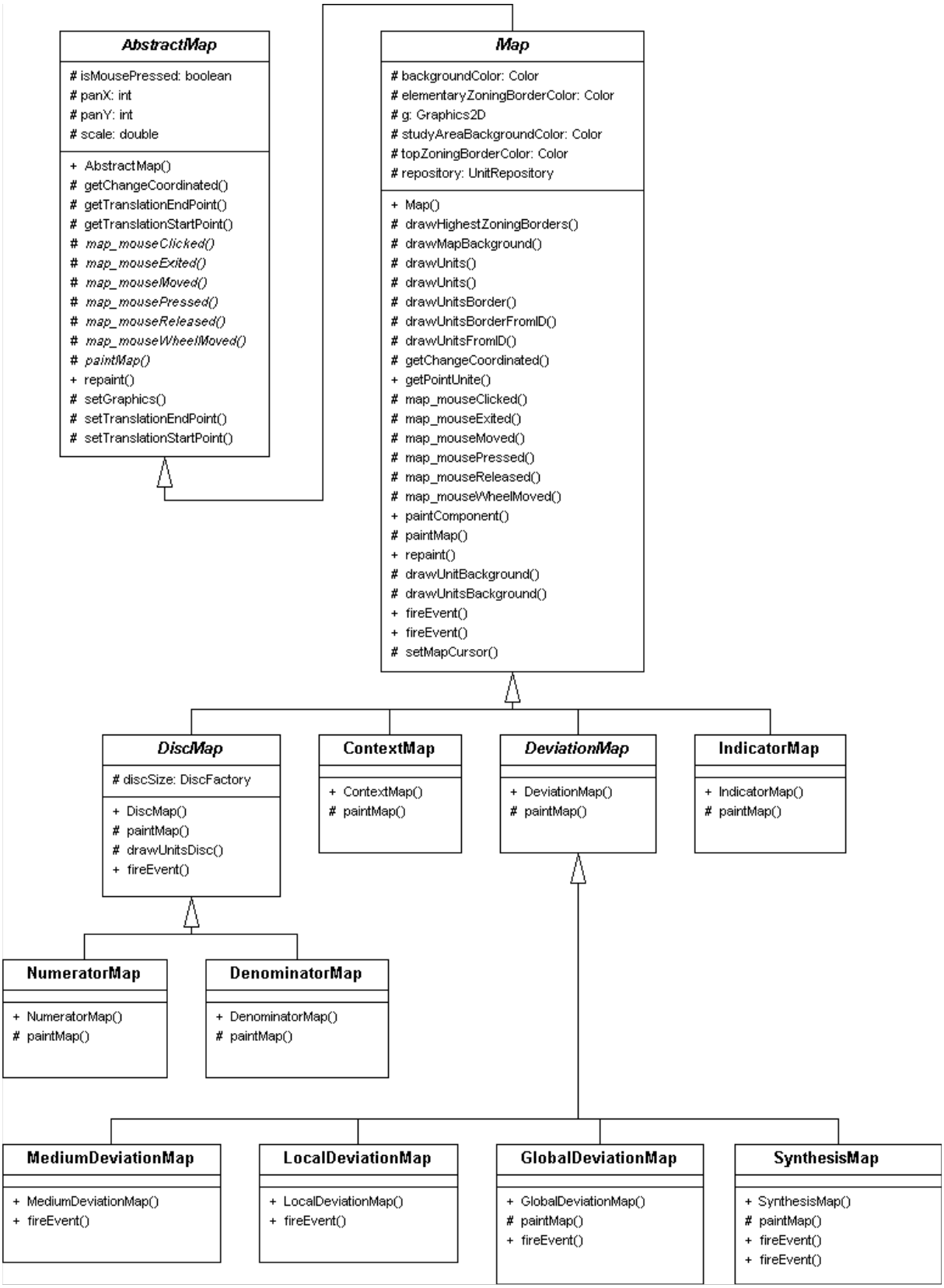


Figure 5.9 : Diagramme de classes : héritage des classes de carte

La classe `DiscMap` surcharge la classe `Map` avec des attributs (pour stocker la taille du plus petit disque affiché sur la carte et la taille du plus grand) et une méthode (pour dessiner des disques). Elle surcharge `Map.fireEvent()` pour réagir lorsque l'option de la taille des

disques est modifiée. A l'instar de la classe `ContextMap`, la classe `DiscMap` surcharge `Map.paintMap()` en ajoutant le dessin des frontières des UT de plus haut niveau.

Les méthodes `NumeratorMap.paintMap()` et `DenominatorMap.paintMap()` surchargent `DiscMap.paintMap()` en ajoutant le dessin des disques proportionnels pour chaque UT de l'espace d'étude.

Les méthodes `IndicatorMap.paintMap()` et `DeviationMap.paintMap()` surchargent `Map.paintMap()` en dessinant chaque UT de l'espace d'étude de la couleur correspondant à sa valeur. Elle dessine également les bordures de ces UT, et les bordures des UT de plus haut niveau.

Les classes `GlobalDeviationMap`, `MediumDeviationMap` et `LocalDeviationMap` surchargent la méthode `fireEvent()` pour réagir en cas de modification du paramètre de déviation utilisé (respectivement déviation globale, moyenne et locale).

La classe `SynthesisMap` surcharge les méthodes :

- `fireEvent()` pour réagir lors de la modification d'un des trois paramètres de déviation ou lors de la modification du seuil ;
- `setMapCursor()` pour faire apparaître un pointeur de souris en forme de main lorsque le paramètre *Histogram* est activé ;
- `drawUnitsBackground()` pour dessiner chaque UT avec la couleur correspondant aux valeurs des trois déviations par rapport au seuil choisi par l'utilisateur ;
- `map_mouseClicked()` pour afficher l'histogramme lorsque l'utilisateur clique sur une UT.

Des extraits du code de certaines de ces classes sont disponibles en annexe (cf. annexe 4).

Spécialisations de légende de carte

Les classes de légende de carte d'Hypercarte héritent de la classe `AbstractLegend`. Nous distinguons trois classes principales de légende (cf. figure 5.10) :

- `ContextLegend` pour la légende de la carte de contexte qui présente la couleur utilisée pour colorier les UT de l'espace d'étude et celle utilisée pour les UT n'y appartenant pas ;
- `PalettesLegend` pour les légendes basées sur des palettes de couleurs qui peuvent être :
  - associées aux plages de valeurs numériques des cartes choroplèthes (cf. paragraphe 3.3.2) (`SimplePalettsLegend`),

  - associées à la logique booléenne multi-attributaire de la synthèse des déviations (`SynthesisPalettsLegend`) ;

- `DiscLegend` pour les légendes de cartes basées sur les symboles proportionnels (cf. paragraphe 3.3.1).

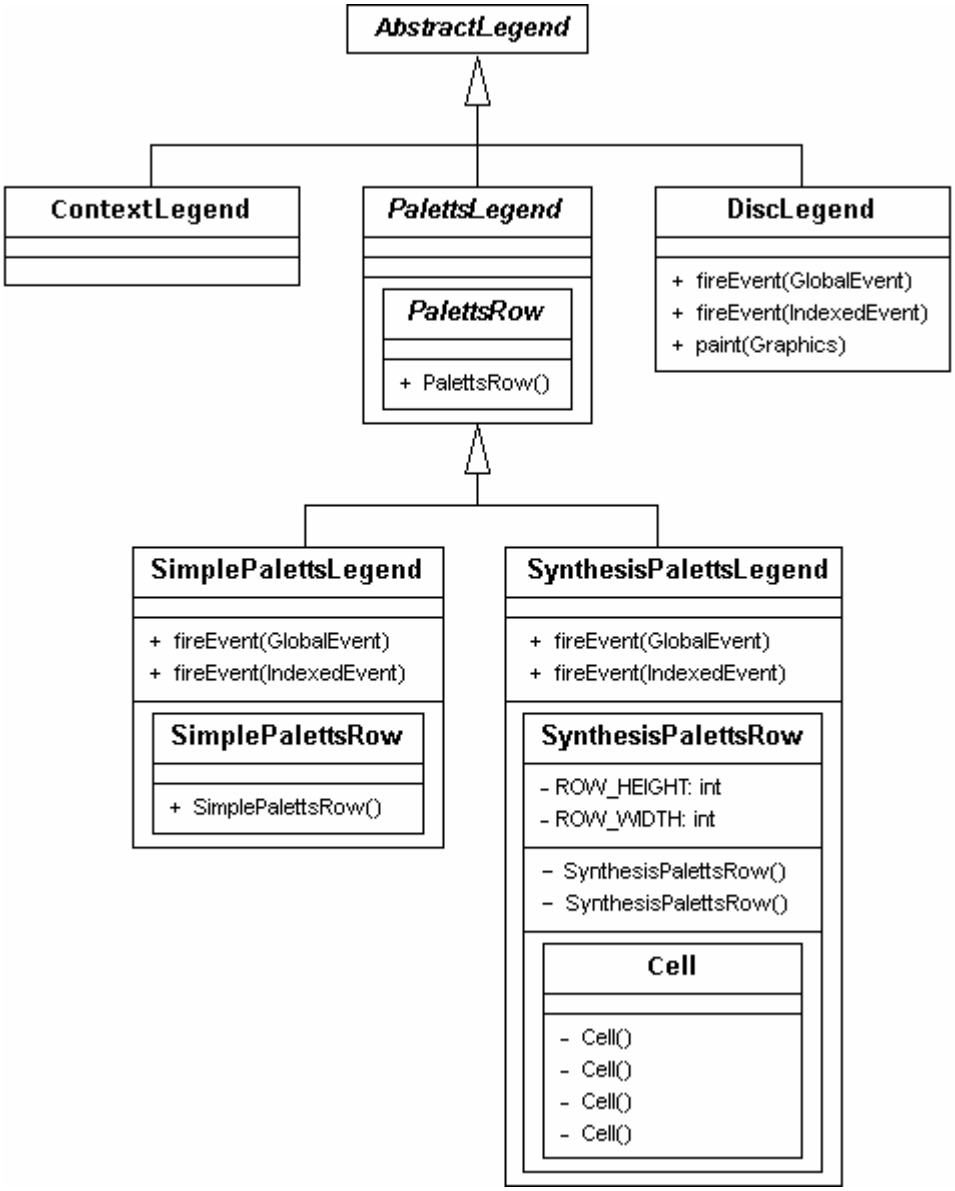


Figure 5.10 : Diagramme de classes : héritage des classes de légende

Spécialisations d'options de carte

Les classes d'options de carte d'Hypercarte héritent de la classe `AbstractOptions` (cf. figure 5.11). Cette dernière est spécialisée par deux classes : `ColorOptions`, qui gère une liste déroulante permettant de sélectionner une couleur, et `SynthesisOptions`, qui permet de gérer les options de la carte de synthèse des déviations.

Deux classes spécialisent la classe abstraite `ColorOptions` : d'une part, `HueOptions`, qui spécialise la gestion de la liste déroulante pour permettre de sélectionner une palette de couleurs, et d'autre part `DiscOptions`, qui permet de choisir la taille de disque.

La classe abstraite `HueOptions` est spécialisée par les deux classes `SingleHueOptions` et `DoubleHueOptions`. Elles permettent la gestion de la liste déroulante pour permettre de sélectionner une palette de couleurs respectivement monotonique et bitonique.

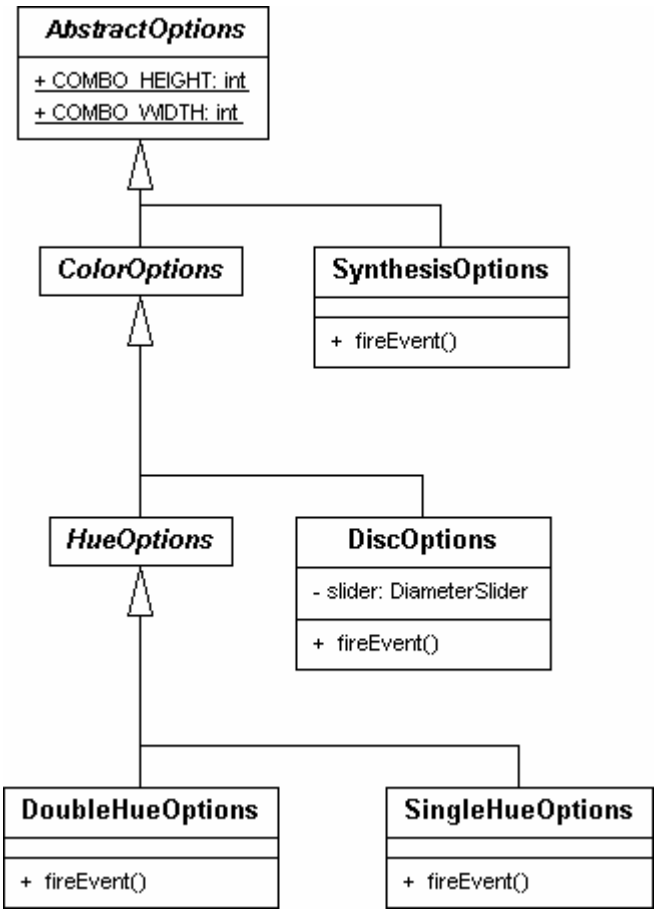


Figure 5.11 : Diagramme de classes : héritage des classes d'options

5.4.2. Composants graphiques

Les classes principales de la version 1.0 d'Hypercarte (comme les classes spécialisant `AbstractMap`, `AbstractLegend` ou `AbstractOptions`) sont complexes. Elles utilisent des briques élémentaires graphiques fournies par Swing, comme les boutons, les listes déroulantes, etc.

Nous avons créé une série de composants, qui sont des classes spécialisant ces briques élémentaires, dans deux buts :

- mutualiser les améliorations apportées aux classes de Swing,
- uniformiser et rendre cohérentes les différentes instances d'une même classe de Swing, dans leur comportement et dans leur présentation.

Le nom des classes issues d'AWT (exemple : `Button`, `Panel`, `CheckBox`, etc.) et redéfinies dans Swing, est basé sur le nom de la classe d'AWT, préfixé par la lettre « J » (`JButton`, `JPanel`, `JCheckBox`, etc.). Le nom des classes issues de Swing et redéfinies dans Hypercarte, est basé sur le nom de la classe AWT équivalente, préfixé par les lettres « HC » (`HCButton`, `HCPanel`, `HCCheckBox`, etc.).

Nous avons créer des composants comme `HCIndexedPanel`, qui spécialise `HCPanel` en ajoutant un attribut `index`, comme `HCTwoStateButton`, pour la création de boutons possédant deux états distincts, ou comme `HCTitledPanel`, qui surcharge la classe `HCPanel` pour obtenir des panneaux dotés d'une barre de titre.

Les classes de composants sont regroupées dans le paquetage `hypercarte.ui.components`.

5.4.3. Réécriture

5.4.3.1. Simplification et nettoyage du code

La version 0.9 est issue d'un prototype dont le code a été surchargé au cours de développements successifs. Pour la version 1.0, le code a été nettoyé afin de le rendre plus clair. Par exemple, pour que l'utilisateur puisse utiliser son propre jeu de données, nous avons rendu le code indépendant des données, en supprimant par exemple toutes les références aux NUTS codées « en dur » dans la version 0.9.

Le code a été remis à plat afin d'être allégé. Par exemple, l'histogramme de la version 0.9 (cf. figure 5.12) nécessite trois classes qui totalisent 440 lignes de code. Dans la version 1.0 (cf. figure 5.13), l'histogramme est composé de deux classes pour 360 lignes.

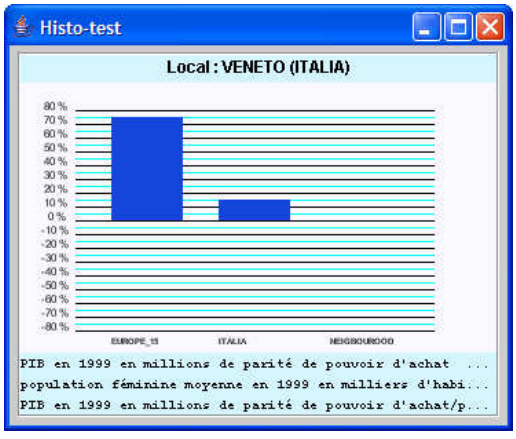


Figure 5.12 : Histogramme de la version 0.9

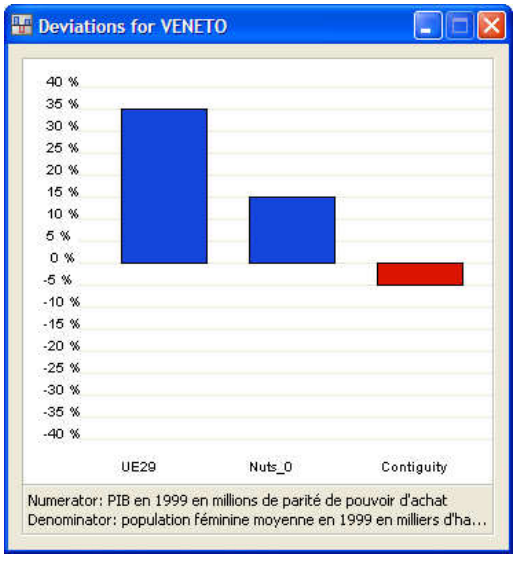


Figure 5.13 : Histogramme de la version 1.0

5.4.3.2. Anglicisation du code

Le projet Hypercarte participant à des projets européens, les sources de financement du projet étant parfois étrangères, il semble intéressant d'avoir un code source en anglais. De plus, si ce code devient open source, il faut impérativement qu'il puisse être facilement partagé et compris par les développeurs, quelle que soit leur langue. Pour une meilleure réutilisabilité, nous avons donc entièrement anglicisé le code source.

5.4.3.3. Etablissement de règles de codage et de nommage

Pour rendre le code plus lisible, nous avons respecté les règles suivantes.

- Les attributs de classe sont soit publics, statiques et constants (`public static final`), soit privés (`private`), soit protégés (`protected`). Les attributs privés et protégés peuvent également être constants (`final`).

- Les attributs statiques sont intégralement écrits en majuscules (avec comme séparateur de mots le caractère de soulignement « _ »), alors que tous les autres attributs et toutes les méthodes sont écrits en mode chameau[56] (cf. code 5.8).

- Les méthodes d'accès aux attributs de classe respectent les règles de nommage des Java Beans – bien que nos classes n'en soient pas. Ainsi, pour un attribut d'un type non booléen s'appelant `aperture`, les méthodes d'accès se nommeraient `getAperture()` et `setAperture(aperture)`. Pour un attribut booléen `opened`, les méthodes d'accès seraient `isOpened()` et `setOpened(opened)`.

- Les méthodes locales invoquées par les écouteurs sont composées : du nom de la classe à laquelle elles appartiennent, du nom de la méthode de l'écouteur, séparés par le caractère de soulignement (cf. code 5.9).

- Les attributs de classe sont systématiquement préfixés par `this` (ou `super` si l'on souhaite utiliser l'attribut d'une classe héritée – et non l'attribut hérité d'une classe).

Code 5.8 : Exemple de formatage de nom de variables

```
public static final int MAX_ZOOM_FACTOR = 25;
public static final int MIN_ZOOM_FACTOR = 0;
private final int ZOOM_FACTOR_DEFAULT_VALUE = 0;
private int zoomFactor = this.ZOOM_FACTOR_DEFAULT_VALUE;
```

Code 5.9 : Exemple de formatage de nom de méthode locale d'écouteur

```
public Map(int mapIndex) {
```

[56] Mode chameau : une variable ou une méthode, composée d'un ou plusieurs mot(s), suit la règle du mode chameau lorsqu'elle commence par une lettre minuscule et utilise une majuscule au début de chaque mot la composant, sans utiliser de séparateur entre les mots. Exemples d'utilisation du mode chameau : `sample`, `camelSample`, `myCamelSample`, `setMyCamelSample`, `readAndSetMyCamelSample`.

```
    super(mapIndex);
    // ...
    addMouseMotionListener(new MouseMotionAdapter() {
        public void mouseMoved(MouseEvent e) {
            map_mouseMoved(e);
        }
    });
}
```

5.5. Ajout de fonctionnalités

5.5.1. Création d'un rapport

Hypercarte est un outil d'analyse multiscalaire qui permet de construire des cartes, mais reste au stade de la démo dans sa version 0.9. En effet, pour devenir un véritable outil, il doit proposer une fonction d'export des données calculées et des cartes créées.

Pour cela, nous avons ajouté une fonction de création de rapport. Dans un répertoire choisi par l'utilisateur, la version 1.0 crée un fichier au format XHTML portant l'extension `.html` (cf. code 5.10). Ce fichier contient les paramètres choisis dans Hypercarte par l'utilisateur, comme l'espace d'étude, le maillage ou les deux stocks. Il contient également le tableau de valeurs utilisées (valeurs de stock) ou calculées (valeurs de l'indicateur, ou des trois déviations) pour chaque UT du niveau de maillage choisi et appartenant à l'espace d'étude.

Code 5.10 : Extrait du code XHTML d'un rapport

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
       <head>
              <meta name="author" content="Hypercarte" />
              <meta http-equiv="content-type" content="text/html;
                     charset=iso-8859-1" />
              <title>Hypercarte Report</title>
       </head>
       <body>
              <h1>Hypercarte Report</h1>
              <h2>Parameters</h2>
              <h3>Space and Zoning</h3>
              <ul>
                     <li><b>Study Area:</b> UE29</li>
                     <li><b>Elementary Zoning:</b> Nuts_2</li>
              </ul>
              <h3>Indicator</h3>
              <ul>
                     <li><b>Numerator:</b> PIB en 1999 en millions d'euros</li>
                     <li><b>Denominator:</b> population totale moyenne
                            en 1999 en  milliers d'habitants</li>
              </ul>
              <h3>Contexts of Reference</h3>
```

```
            <ul>
                  <li><b>Global:</b> UE15</li>
                  <li><b>Medium:</b> Nuts_0</li>
                  <li><b>Local:</b> Contiguity</li>
            </ul>
            <h2>Generated maps</h2>
            <div>
                  <img src="map0.png" alt="Map" style="border:solid 1px #000;" />
                  <img src="map1.png" alt="Map" style="border:solid 1px #000;" />
                  <img src="map2.png" alt="Map" style="border:solid 1px #000;" />
                  <img src="map3.png" alt="Map" style="border:solid 1px #000;" />
                  <img src="map4.png" alt="Map" style="border:solid 1px #000;" />
                  <img src="map5.png" alt="Map" style="border:solid 1px #000;" />
                  <img src="map6.png" alt="Map" style="border:solid 1px #000;" />
                  <img src="map7.png" alt="Map" style="border:solid 1px #000;" />
            </div>
            <h2>Table of generated results</h2>
            <table border="1" cellspacing="0" style="border-collapse:collapse;">
                  <tr>
                        <th rowspan="2">Code</th>
                        <th rowspan="2">Name</th>
                        <th>Numerator</th>
                        <th>Denominator</th>
                        <th>Indicator</th>
                        <th>Global Deviation</th>
                        <th>Medium Deviation</th>
                        <th>Local Deviation</th>
                  </tr>
                  <tr>
                        <th>GDP99ME</th>
                        <th>AVPOP99T</th>
                        <th>GDP99ME/AVPOP99T</th>
                        <th>UE15</th>
                        <th>Nuts_0</th>
                        <th>Contiguity</th>
                  </tr>
                  <tr>
                        <td>UKL2</td>
                        <td>EAST WALES</td>
                        <td style="text-align:right">24063</td>
                        <td style="text-align:right">1068</td>
                        <td style="text-align:right">22,53</td>
                        <td style="text-align:right">105,38</td>
                        <td style="text-align:right">97,5</td>
                        <td style="text-align:right">100,7</td>
                  </tr>
                  <!-- etc. -->
            </table>
      </body>
</html>
```

Des images sont créées dans le même répertoire. Elles sont des copies d'écran logicielles de chaque carte. Le code XHTML fait appel à ces images. Les copies d'écran de la figure 5.14 montre le rapport tel qu'il apparaît dans un navigateur Web.

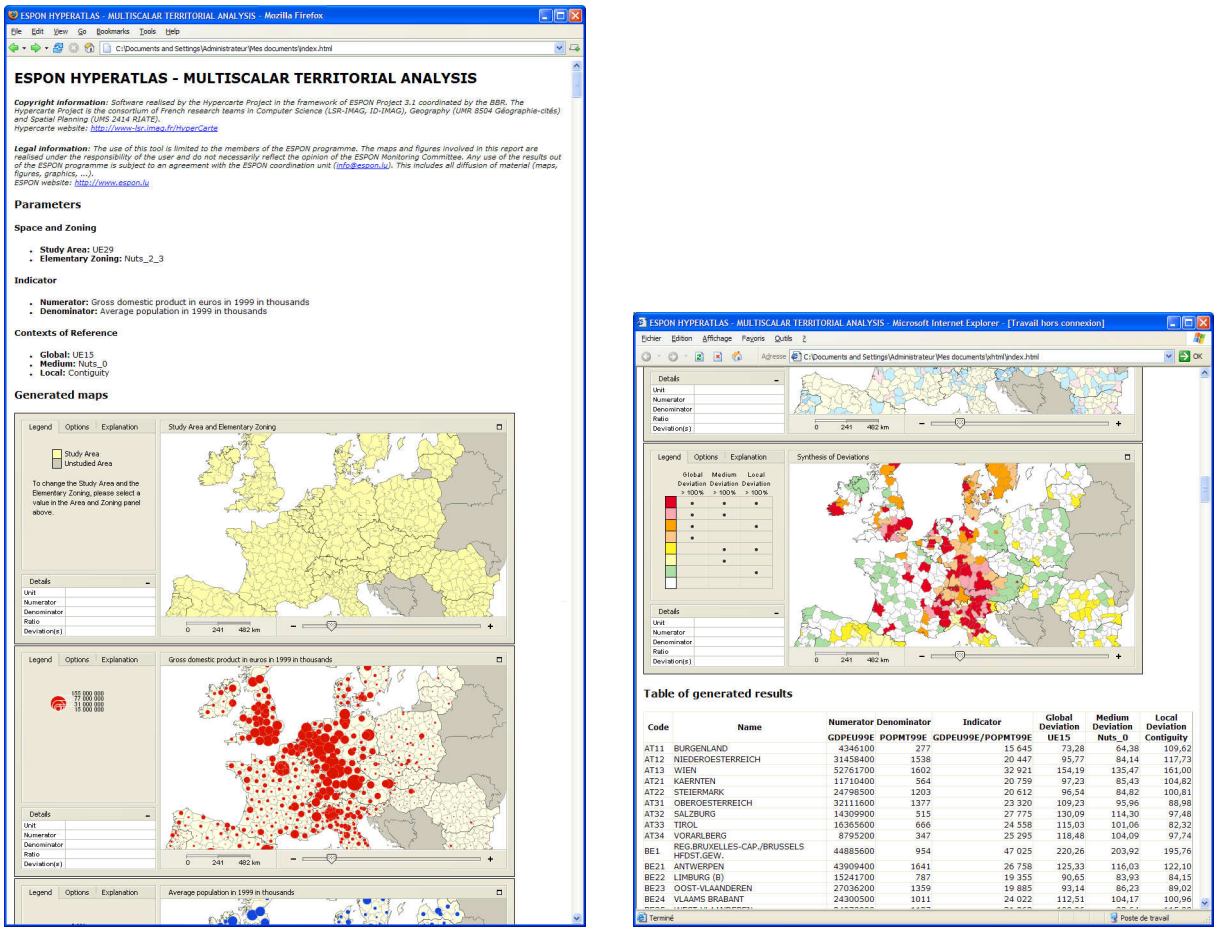


Figure 5.14 : Le rapport XHTML généré par Hypercarte

Une fois les fichiers créés, Hypercarte fait appel aux API de l'OS qui l'exécute, pour ouvrir le rapport dans le logiciel associé à l'extension `.html`. Par manque de temps, nous n'avons implémenté la fonctionnalité d'ouverture automatique de la page XHTML que pour les OS de type Microsoft Windows 32 bits.

5.5.2. Sauvegarde et restauration de l'espace de travail

L'espace de travail représente l'interface d'Hypercarte sur laquelle l'utilisateur peut faire des modifications. L'état de l'espace de travail peut être sauvegardé dans un fichier au format XML (cf. code 5.11). Ce fichier permet de connaître :

- la valeur des paramètres comme l'espace d'étude, le maillage, les stocks choisis, l'espace de déviation global sélectionné, etc.,
- la valeur du facteur de zoom,
- la valeur du vecteur de déplacement de la carte,
- les options sélectionnées pour chaque carte comme la couleur et la taille des disques, les couleurs et le nombre d'éléments de la palette, la valeur de seuil de la carte de

synthèse, etc.,

et de savoir :

- si le panneau de détails est réduit,
- si la barre d'outils et le panneau de paramètres sont affichés,
- si la palette d'une carte est inversée,
- si la progression pour calculer la palette d'une carte est arithmétique ou géométrique,
- etc.

Un fichier de sauvegarde de l'espace de travail peut être restauré dans Hypercarte. L'utilisateur retrouve alors l'espace de travail exactement tel qu'il était lorsque le fichier a été généré.

Code 5.11 : Document XML contenant l'information nécessaire à la restauration d'un espace de travail

```
<?xml version='1.0'?>
<hypercarte>
	<zoom_factor>9.0</zoom_factor>
	<pan_x>309</pan_x>
	<pan_y>77</pan_y>
	<enable_pan>true</enable_pan>
	<enable_histogram>true</enable_histogram>
	<display_parameters>true</display_parameters>
	<display_toolbar>true</display_toolbar>
	<expand_map>false</expand_map>
	<minimize_details>false</minimize_details>
	<indicator_numerator>GDPEU99E</indicator_numerator>
	<indicator_denominator>POPMT99E</indicator_denominator>
	<elementary_zoning>Nuts_2</elementary_zoning>
	<study_area>UE29</study_area>
	<reference_area>UE25</reference_area>
	<reference_zoning>Nuts_0</reference_zoning>
	<reference_neighbourhood>Contiguity</reference_neighbourhood>
	<reference_value>0.0</reference_value>
	<map0/>
	<map1>
		<color>1</color>
		<disc_size>0.5</disc_size>
	</map1>
	<map2>
		<color>0</color>
		<disc_size>0.5</disc_size>
	/map2>
	<map3>
		<color>0</color>
		<class_number>4</class_number>
		<quantile>0</quantile>
	</map3>
	<map4>
		<color>5</color>
```

```
            <class_number>8</class_number>
            <is_arithmetic>false</is_arithmetic>
            <paletts_reversed>false</paletts_reversed>
      </map4>
      <map5>
            <color>5</color>
            <class_number>8</class_number>
            <is_arithmetic>false</is_arithmetic>
            <paletts_reversed>false</paletts_reversed>
      </map5>
      <map6>
            <color>5</color>
            <class_number>8</class_number>
            <is_arithmetic>false</is_arithmetic>
            <paletts_reversed>false</paletts_reversed>
      </map6>
      <map7>
            <threshold_value>100</threshold_value>
            <threshold_operator>0</threshold_operator>
      </map7>
</hypercarte>
```

5.5.3. Inhibition de l'écouteur de liste déroulante et de case à cocher

Des écouteurs scrutent les listes déroulantes. Ainsi, lorsque l'utilisateur modifie l'élément sélectionné de la liste déroulante, l'écouteur permet au programme d'effectuer une action. Cette action peut consister à enregistrer la valeur du nouvel élément sélectionné dans les classes de stockage des paramètres ou à déclencher un événement pour prévenir les autres composants.

Code 5.12 : Méthode de changement silencieux de l'élément sélectionné d'une liste déroulante

```
public abstract class HCComboBox extends JComboBox {

    public void setSelectedIndex(int index, boolean silentMode) {

        // If no change required, return
        if (getSelectedIndex() == index) return;

        // If not in silent mode, call super
        if (!silentMode) super.setSelectedIndex(index);

        // Backup action listeners
        ActionListener[] backupedActionListeners = getActionListeners();

        // Remove action listeners
        removeActionListeners();

        // Set index
        super.setSelectedIndex(index);
```

```
        // Restore action listeners
        addActionListeners(backupedActionListeners);
    }
}
```

Avec la restauration d'un fichier de sauvegarde de l'espace de travail, le programme peut désormais modifier l'élément sélectionné d'une liste déroulante. Pour éviter que cette action ne prévienne inutilement l'écouteur, nous avons doté les listes déroulantes d'Hypercarte d'une méthode de modification de l'indice de l'élément sélectionné, avec une option permettant de choisir le mode silencieux.

Si le mode silencieux est demandé, les écouteurs existants sont sauvegardés, puis supprimés de la liste des écouteurs de la liste déroulante (cf. code 5.12). L'indice de l'élément sélectionné est alors modifié puis les écouteurs sauvegardés sont de nouveau ajoutés à la liste des écouteurs de la liste déroulante. Ce mécanisme a également été mis en place pour les cases à cocher.

5.5.4. Méthode d'arrondi adaptatif

Comme on ne connaît pas a priori la valeur de la distribution d'un rapport, il faut pouvoir afficher avec plus ou moins de chiffres après la virgule. Nous nous sommes inspiré de la préconisation des géographes de l'équipe PARIS. Tout d'abord, nous calculons la valeur minimum Z (1).

$$Z = min(\Sigma \text{ numérateur} / \Sigma \text{ dénominateur}) \qquad (1)$$

Ensuite nous calculons son logarithme décimal et en déduisons le nombre de chiffre à mettre après la virgule (ou le nombre de zéros à placer avant la virgule) (2).

$$Virgule = trunc(log10(z))-2 \qquad (2)$$

Ainsi, pour une valeur minimum Z = 0.054, on trouve Virgule = –3 ce qui indique qu'il faut placer 3 chiffres après la virgule. En revanche, pour une valeur Z = 456, Virgule = 0, ce qui indique que les valeurs seront écrites sans chiffre après la virgule. Enfin, avec Z = 458654 on trouve Virgule = +3 ce qui indique qu'il faut remplacer les chiffres de centaine, dizaine et unité par des zéros.

5.6. Amélioration des performances

5.6.1. Affichage

5.6.1.1. Utilisation du double buffering

Le *double buffering* (cf. paragraphe 2.2.1.2.2) est activé pour chaque composant graphique de la version 1.0, notamment pour les cartes, qui sont des composants dont le temps de calcul nécessaire pour l'affichage est long. Cela est particulièrement intéressant pour le réaffichage d'une carte précédemment affichée. En effet, dans ce cas, la carte n'est pas recalculée : Swing utilise la représentation stockée dans un cache graphique lors du premier affichage.

5.6.1.2. Adaptation de la définition des contours cartographiques

La version 1.0 intègre un fichier de définition des géométries retravaillé par les géographes du projet qui réduit par plus de deux le nombre d'arcs.

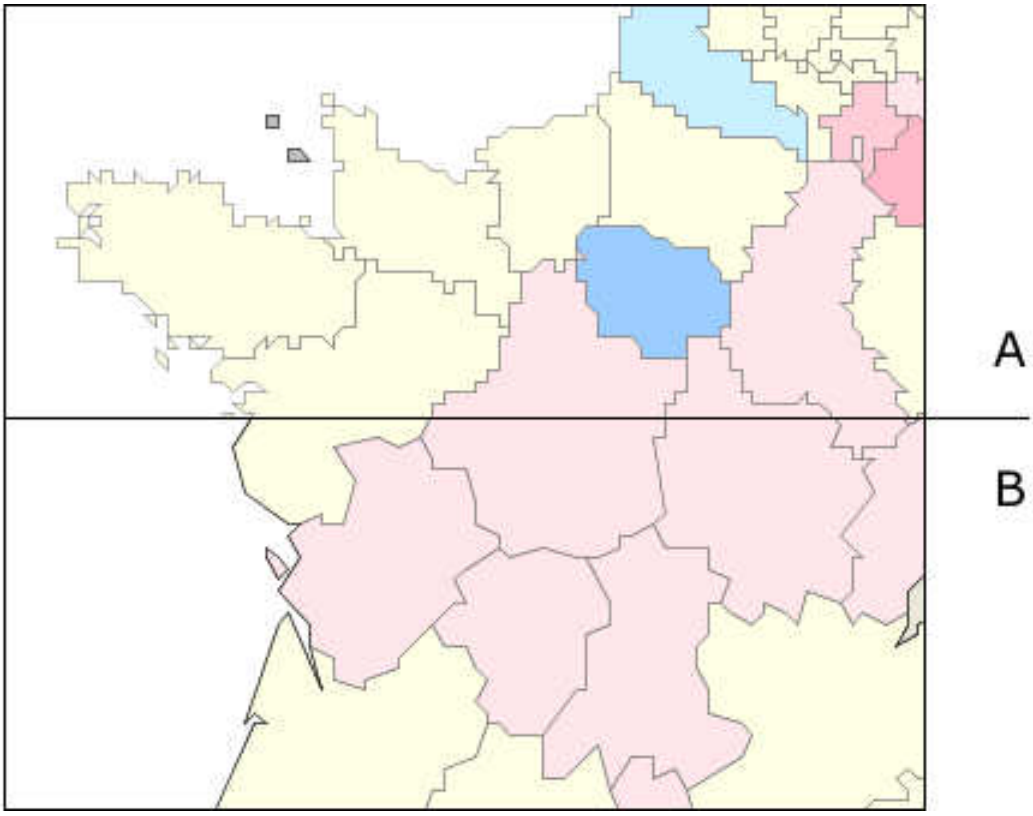


Figure 5.15 : Modifications du contour des unités territoriales

La version 0.9 utilise des contours constitués de segments orientés à 0°, 45°, 90° de l'horizontale (A). La version 1.0 utilise des contours constitués de segments d'orientation libre (B), ce qui divise par plus de deux le nombre de points nécessaires pour définir les unités territoriales.

Les cartes ainsi redéfinies sont plus lisibles, notamment à de grandes échelles (cf. figure 5.15). De plus, cette baisse de la résolution cartographique permet de réduire :

- la taille des fichiers texte lus par le sérialiseur,
- le temps de sérialisation des données,
- la taille du fichier sérialisé,
- le temps de désérialisation des données et donc le temps de chargement de l'application,
- le temps d'affichage d'une carte.

5.6.2. Données

5.6.2.1. Mise en cache des UT filtrées

L'association de l'espace d'étude et du niveau de maillage élémentaire définit un ensemble d'UT. Au cours de l'utilisation de l'application, très fréquemment, seules les UT de cet ensemble sont utilisées.

Comme le parcours de liste est une opération coûteuse, nous utilisons un mécanisme de cache de données qui enregistre les UT de cet ensemble dans un espace de stockage dédié (classe `UnitRepository`). La liste des UT stockées dans cet espace est utilisée, de préférence à la liste complète des UT. L'instance de `UnitRepository` est mise à jour lorsque l'espace d'étude ou le niveau de maillage élémentaire est modifié.

5.6.2.2. Mise en cache des calculs intermédiaires

Pour toutes les UT stockées par l'instance de `UnitRepository`, il est nécessaire de calculer :

- le rapport entre les valeurs des deux stocks choisis (l'indicateur),
- le delta entre l'indicateur de l'UT et la moyenne des indicateurs des UT de l'espace de référence choisi,
- le delta entre l'indicateur de l'UT et l'indicateur de l'UTS située au niveau de maillage de référence choisi,
- le delta entre l'indicateur de l'UT et la moyenne des indicateurs de ses UT contiguës,

Ces calculs sont réalisés au fur et à mesure que les cartes sont affichées. Pour éviter de recalculer ces valeurs, nous les enregistrons dans un espace de stockage dédié (classe `DeltaRepository`) jusqu'à ce que l'espace d'étude ou le niveau de maillage élémentaire soit modifié.

5.6.2.3. Recherche de l'UT survolée par la souris

Afin de réduire le temps de recherche, le corpus est limité aux UT de l'espace d'étude appartenant au niveau de maillage élémentaire (cf. paragraphe 5.6.2.1).

Dans la version 0.9, le résultat s'affiche dans un panneau surgissant (*popup*) qui se superpose à la carte. Il faut donc construire et afficher le panneau. Ensuite, après disparition du panneau, il faut repeindre la partie de la carte masquée par celui-ci.

Dans la version 1.0, le résultat s'affiche dans un panneau visible, placé en permanence à côté de la carte. Il suffit donc de mettre à jour le contenu de ce panneau pour afficher puis effacer les informations concernant une UT survolée.

5.7. Améliorations ergonomiques

5.7.1. Restructuration de l'interface

La restructuration de la fenêtre a pour but de rendre l'interface plus standard en suivant les recommandations pour le développement d'interfaces utilisateurs décrites par des spécialistes [BAST, 93], par des éditeurs d'OS (tels que Microsoft® [MICR, 04], Apple [APPL, 04] ou GNOME [GNOM, 02]) ou encore par des organismes scientifiques [CNRS, 00].

5.7.1.1. Structure

La modification de structure porte sur l'ajout de trois composants fondamentaux d'une fenêtre (les barres de menus, d'outils et de statut) et sur la réorganisation et le réagencement des

composants graphiques dans la fenêtre. La figure 5.16 présente le schéma de la nouvelle structure de la fenêtre d'Hypercarte.

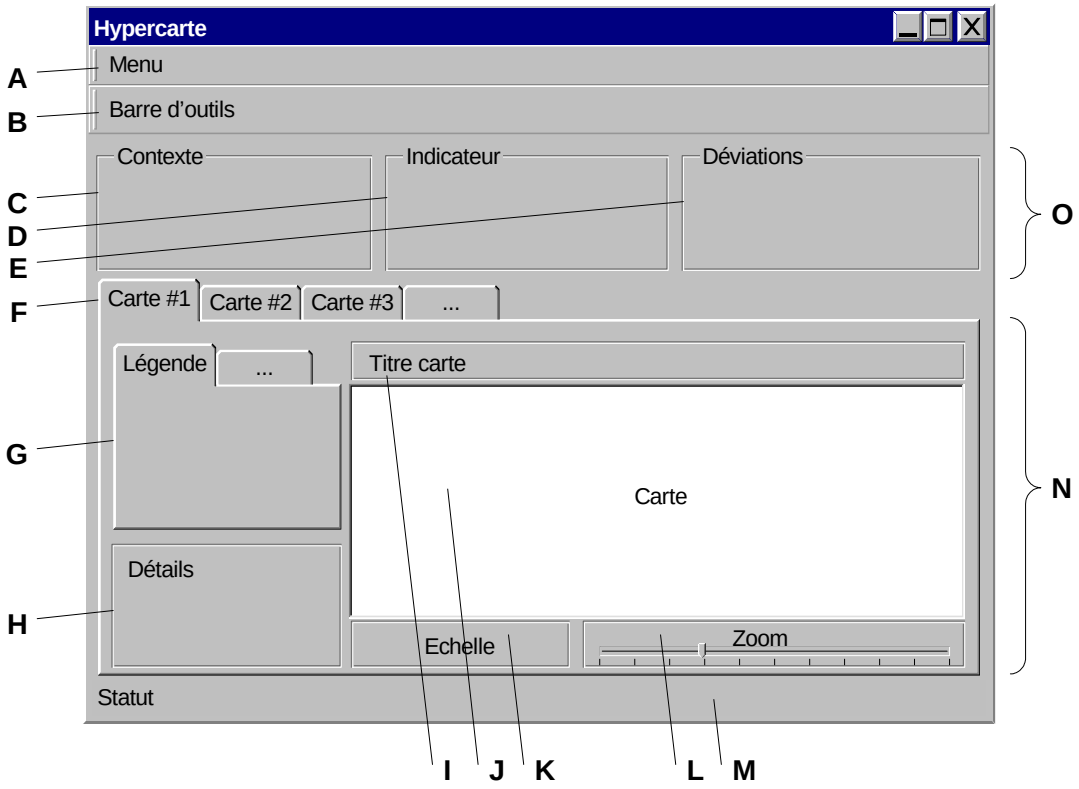


Figure 5.16 : Structure de l'interface de la version 1.0

(A) Barre de menus
(B) Barre d'outils
(C) Zone de saisie des 2 paramètres du contexte
(D) Zone de saisie des 2 champs formant l'indicateur
(E) Zone de saisie des 3 paramètres de déviation
(F) Onglets de choix de la carte à afficher
(G) Panneau à onglets regroupant la légende, les options et la description de chaque carte
(H) Zone d'affichage du détail de l'unité territoriale survolée par la souris
(I) Titre de la carte
(J) Zone d'affichage de la carte
(K) Echelle cartographique
(L) Curseur de zoom
(M) Barre de statut
(N) Les composants de cette zone (de F à J) sont instanciés autant de fois qu'il existe d'onglets (F). L'échelle (K) et le curseur de zoom (L) sont liés au facteur de zoom, qui est identique sur toutes les cartes.
(O) Zone regroupant les paramètres cartographiques. Cette zone, qui est un regroupement logique utilisé par la commande de masquage des paramètres, n'est pas représentée à l'écran.

La figure 5.17 présente une copie d'écran de la version 1.0.

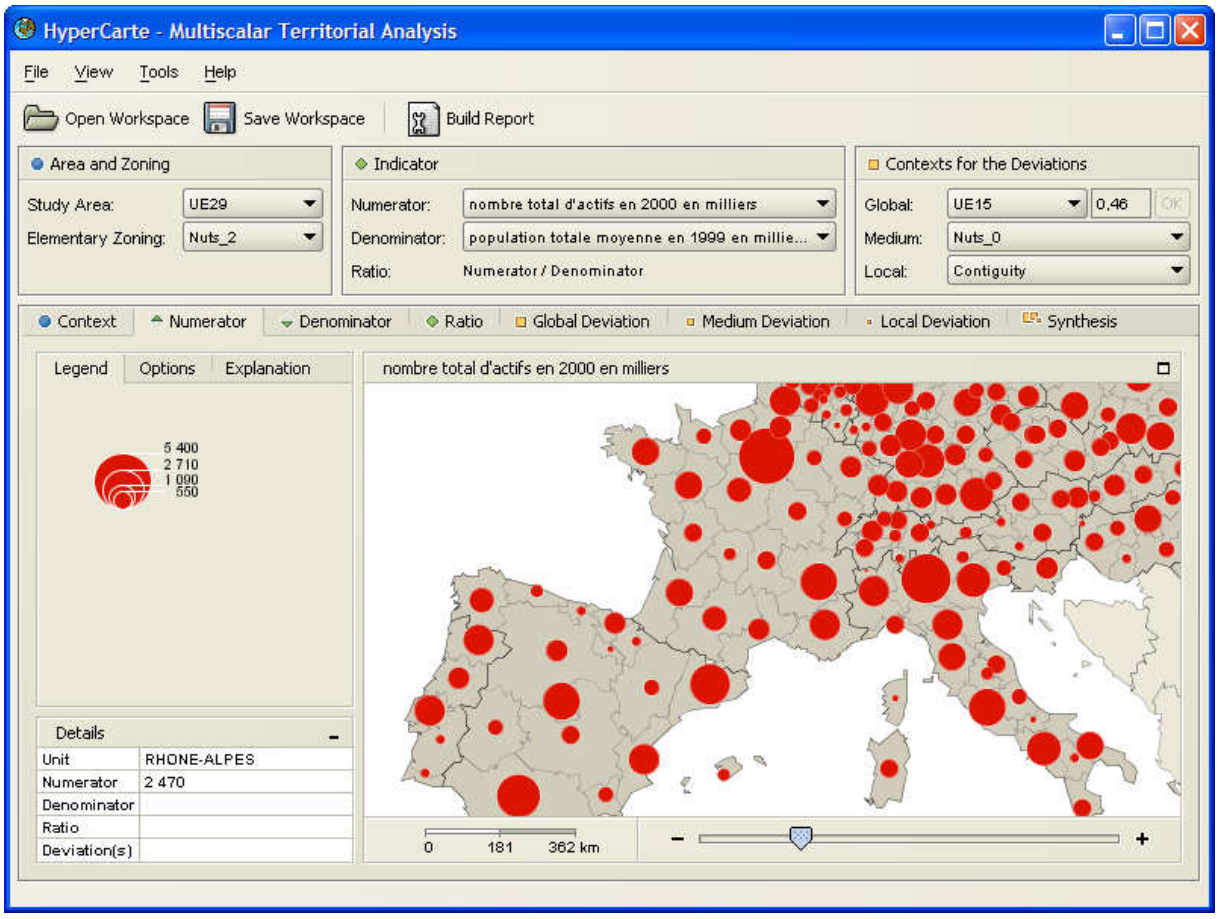


Figure 5.17 : Interface de la version 1.0

5.7.1.1.1. Menu

Toute fonction mise à disposition de l'utilisateur doit être accessible via un menu. La structure du menu (cf. figure 5.18) est simple, logique et surtout standard.

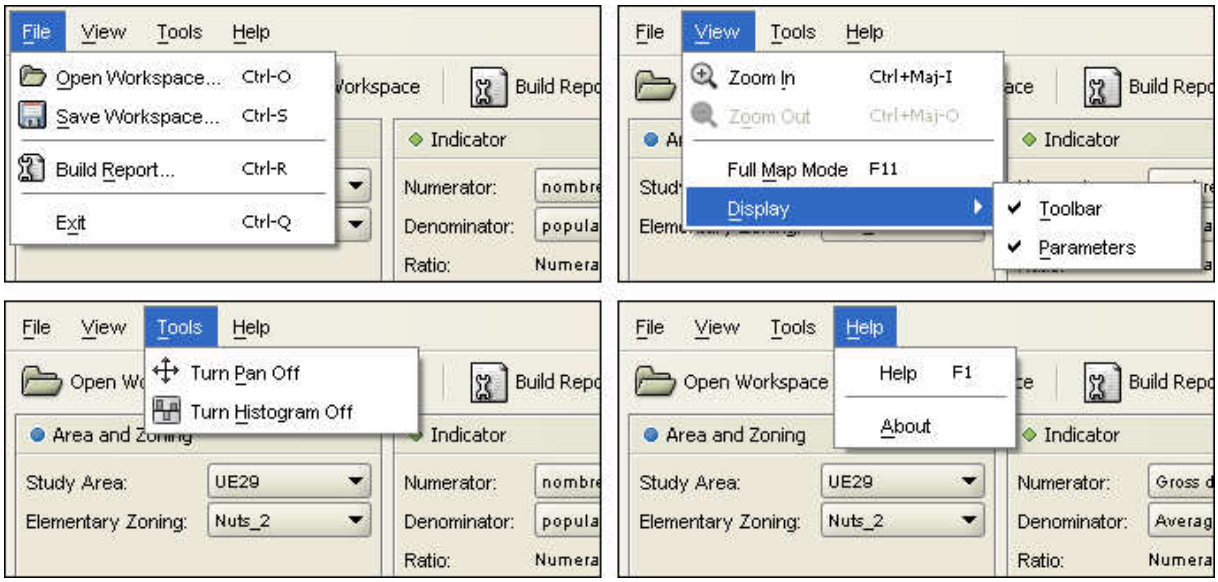


Figure 5.18 : Menus de la version 1.0

Sa structure est la suivante :

- Le menu *File* regroupe les fonctions d'entrée/sortie telles que la sauvegarde et la restauration d'un espace de travail, la génération d'un rapport, et la fermeture de l'application.

- Le menu *View* comprend, d'une part, les fonctions de zoom cartographique et, d'autre part, les fonctions d'affichage/masquage de certains composants graphiques.

- Le menu *Tools* permet à l'utilisateur d'activer ou de désactiver la fonction de déplacement de la carte et la fonction d'histogramme.

- Le menu *Help* possède la fonction d'appel du fichier de l'aide et la fonction d'affichage de la boîte de dialogue *About*.

Un raccourci clavier est affecté aux commandes les plus utilisées. Certains raccourcis s'imposent comme `Ctrl+O`[57] pour l'ouverture d'un fichier, `Ctrl+S` pour sa sauvegarde, `F11` pour le mode plein écran, ou `Ctrl+Q` pour quitter l'application [MICR, 04]. Les lettres `I` et `O` s'imposant également pour les fonctions de *Zoom In* et *Zoom Out*, et `Ctrl+O` étant déjà pris, il a fallu choisir une séquence de caractères différente de `Ctrl`. Nous avons choisi arbitrairement `Ctrl+Maj+I` et `Ctrl+Maj+O`.

5.7.1.1.2. Barre d'outils

La barre d'outils permet de mettre à disposition de l'utilisateur les commandes les plus couramment utilisées. Parmi les commandes disponibles via la barre de menus, nous choisissons les fonctions de zoom et les fonctions d'entrée/sortie.

Toutefois, les fonctions accessibles par un autre composant graphique que la barre de menus ne nécessitent pas de figurer dans la barre d'outils. Par exemple les commandes *Zoom In* et *Zoom Out* sont accessibles très instinctivement par le curseur de zoom (cf. figure 5.16) et ne sont donc pas reprises dans la barre d'outils.

Figure 5.19 : Barre d'outils de le version 1.0

Seules les trois fonctions gérant les flux d'entrée/sortie (sauvegarde et restauration de l'espace de travail, et construction du rapport) apparaissent dans la barre.

Cette dernière peut être masquée et affichée à volonté grâce à la commande `View > Display > Toolbar` de la barre de menus (cf. figure 5.20).

[57] Ctrl+O : la séquence de caractères est composée de la touche Contrôle suivie de la touche O.

Figure 5.20 : Masquage de la barre d'outils

5.7.1.1.3. Barre de statut

La barre de statut permet d'afficher une information contextuelle, comme la description d'un bouton ou d'un élément de menu. Nous l'utilisons de préférence aux infobulles[58]. En effet, lorsqu'une infobulle disparaît, il faut redessiner l'espace qu'elle a occupé. Lorsqu'il y a intersection entre cet espace et la carte, cela peut être coûteux.

5.7.1.1.4. Panneau latéral à onglets

La version 0.9 d'Hypercarte (cf. figure 4.14) est construite avec un panneau de légende de carte. Elle affiche également un panneau d'options permettant de modifier les palettes de couleurs, le nombre d'éléments de la palette, la taille des disques, les valeurs seuils, etc. Chaque carte utilise une légende et un panneau d'options qui lui est propre. Ces deux panneaux sont affichés en permanence, or leur affichage simultané n'est pas nécessaire.

Dans la version 1.0, nous les superposons en utilisant un panneau à onglets, implémenté par le composant `javax.swing.JTabbedPane`[59].

Le besoin d'explication dans l'utilisation et la signification de certaines cartes ayant été jugés nécessaire par les géographes, nous avons ajouté un troisième onglet qui fournit à l'utilisateur une explication sur chaque carte.

5.7.1.1.5. Panneau d'affichage des informations d'UT

Les informations concernant l'UT survolée par la souris sont affichées dans un nouveau panneau (cf. figure 5.21).

La réduction de la taille de ce panneau permet :

- de ne plus effectuer la recherche des informations relatives aux UT survolées, qui requiert des ressources système. La réduction de la taille du panneau s'effectue en cliquant sur le bouton _ (cf. figure 5.22), l'agrandissement en cliquant sur le bouton 🗗.

[58] Infobulle (*tooltips*) : bulle d'aide contextuelle.
[59] *Tabbed pane* signifie textuellement « panneau tabulé ». Nous le traduisons par « panneau à onglets ».

- de libérer de l'espace dans la fenêtre : la sélection de l'onglet d'explication de carte réduit automatiquement la taille du panneau (cf. figure 5.23) ; la sélection d'un autre onglet restaure la taille du panneau.

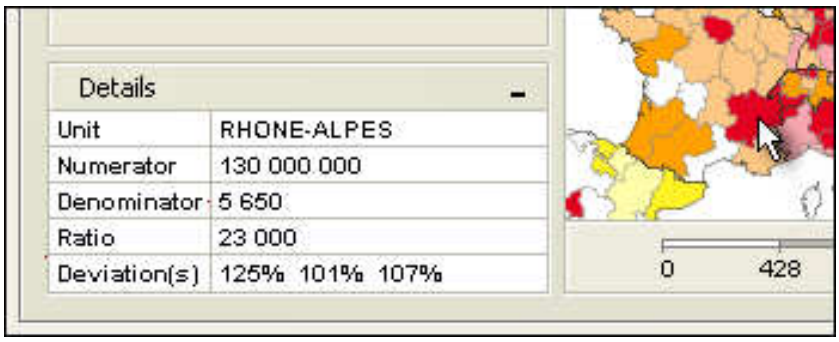


Figure 5.21 : Panneau d'affichage des informations sur l'UT survolée

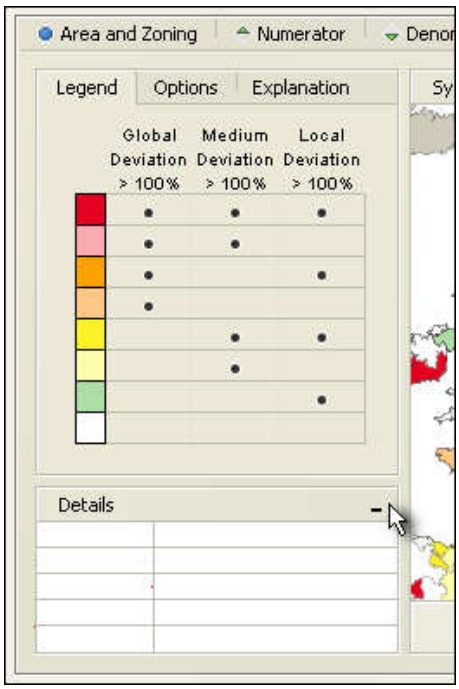


Figure 5.22 : Réduction du panneau par clic sur bouton _

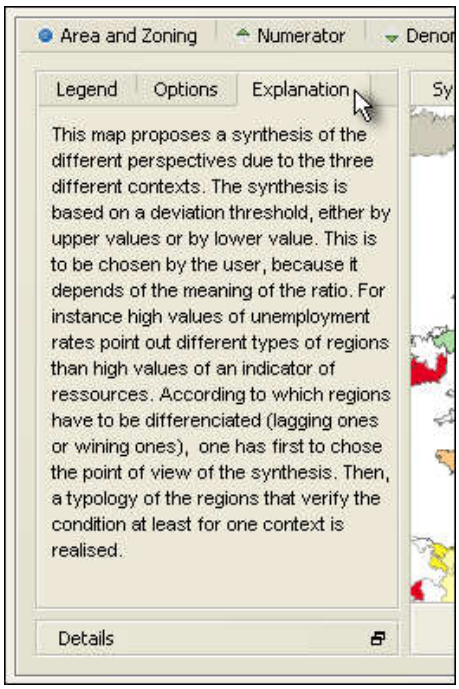


Figure 5.23 : Réduction du panneau par sélection de l'onglet d'explication

5.7.1.2. Taille dynamique

5.7.1.2.1. Taille liée au contenu

Le panneau des paramètres est une zone de formulaire contenant principalement des listes déroulantes. Ces listes contiennent des intitulés qui varient suivant les données utilisées. Ainsi, en utilisant les données fournies pour le développement du logiciel par l'équipe PARIS de Géographie-Cités, la liste des espaces d'étude (*Study Area*) contient des codes relativement courts. Mais ces codes pourraient être remplacés par des intitulés équivalents (par exemple « Union européenne des 29 » au lieu de « UE29 ») dans un autre jeu de données. Cette zone de formulaire doit donc avoir une structure très dynamique.

La taille des listes déroulantes *Study Area*, *Elementary Zoning* est imposée par la longueur du plus long intitulé parmi les deux listes (cf. figure 5.24). Cette taille permet de fixer celle du panneau *Area and Zoning*.

La taille du panneau *Contexts for the Deviations* est liée à la plus grande largeur entre, d'une part, le plus long intitulé parmi les listes Medium et Local, et d'autre part, l'addition des largeurs du plus long intitulé de la liste *Global*, du champ de saisie et du bouton.

Figure 5.24 : Adaptation de la taille des composants du panneau de paramètres

Enfin, le panneau *Indicator* occupe l'espace laissé libre par les panneaux *Context* et *Deviations*, et les listes déroulantes *Numerator* et *Denominator* utilisent toute la largeur du panneau qui les contient.

5.7.1.2.2. Ajustement de la taille lors du redimensionnement de la fenêtre

La résolution minimum requise est de 800 par 600 pixels. Sachant que l'application doit offrir une ergonomie optimale quelle que soit la résolution, l'interface doit répartir de manière optimale les surfaces gagnées en augmentant la taille de la fenêtre. Cette répartition (cf. figure 5.16) doit se faire entre la carte – qui est le composant central de l'application, et les listes déroulantes de sélection du numérateur et du dénominateur – qui peuvent contenir de longs intitulés (cf. figure 5.25).

Ainsi, les composants ayant une largeur adaptable en fonction de la résolution de la fenêtre – outre les barres de menus, d'outils et de statut – sont les suivants : les deux listes déroulantes de choix du numérateur et du dénominateur, la carte et son titre et le curseur de zoom. Les composants s'adaptant en hauteur à la résolution de la fenêtre sont la carte et le panneau à onglets (cf. figure 5.26).

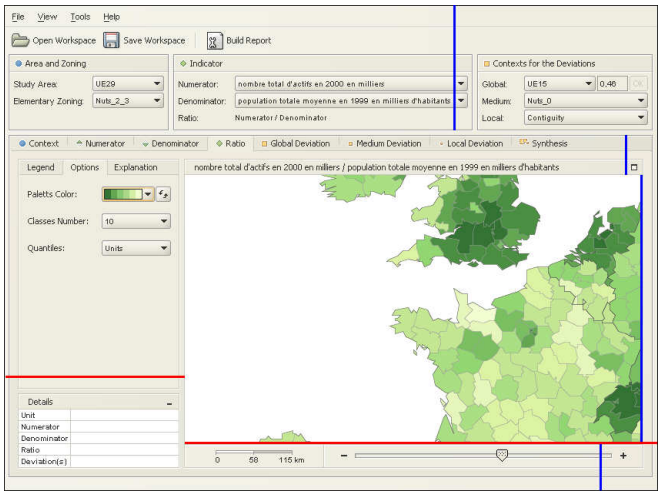


Figure 5.25 : Axes de redimensionnement dynamiquement au changement de taille de fenêtre de l'interface de la version 1.0

En rouge figurent les axes de redimensionnement vertical, en bleu les axes de redimensionnement horizontal.

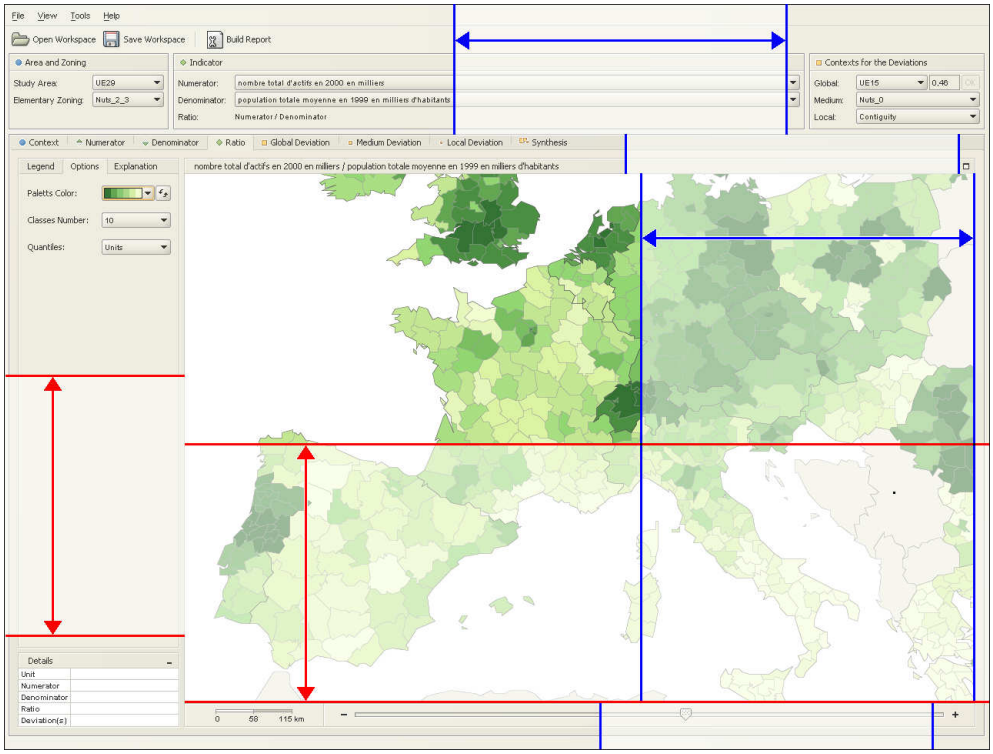


Figure 5.26 : Répartition des zones redimensionnées dynamiquement au changement de taille de fenêtre de l'interface de la version 1.0

En agrandissant la fenêtre, les listes déroulantes situées dans le panneau Indicator*, la carte et son titre ainsi que le curseur de zoom s'adaptent en largeur. La carte et les panneaux* Legend / Options / Explanation *s'adaptent en hauteur.*

5.7.1.3. Découpage de la fenêtre

Les changements dans la structure de l'interface et le choix des composants graphiques dont la taille est dynamique, permettent de réduire l'espace inutilisé lorsque la taille de la fenêtre est de 800 par 600 pixels, par rapport à la version 0.9 (cf. figure 5.27).

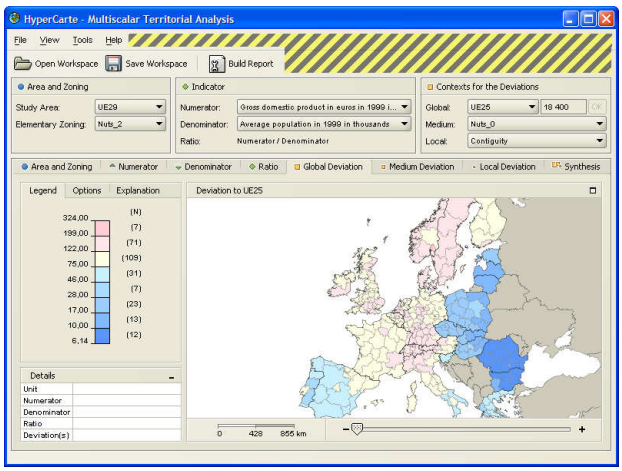


Figure 5.27 : Interface de la version 1.0 en mode 800×600.

Ils permettent également l'optimisation de l'espace inutilisé lorsque la taille de la fenêtre est grande (cf. figure 5.28).

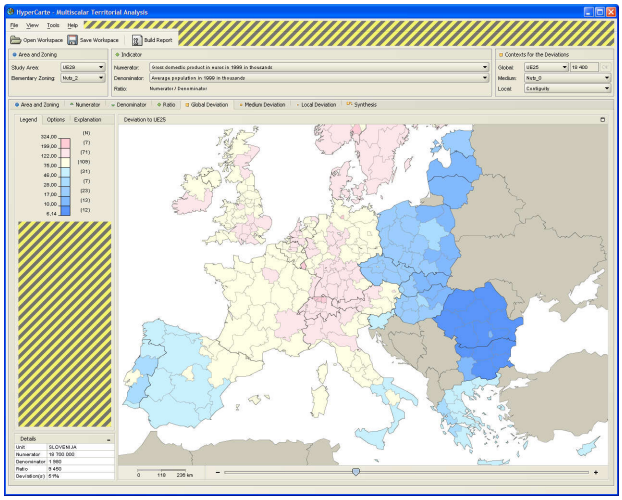


Figure 5.28 : Interface de la version 1.0 en mode 1250×1024

Les zones hachurées mettent en évidence les surfaces inutilisées.

5.7.1.4. Amélioration du rapport surfacique

Le tableau 5.1 présente la taille et la surface de la carte pour les deux versions de l'interface d'Hypercarte et pour deux tailles de fenêtre.

	Fenêtre	Carte	
		Version 0.9	Version 1.0
Largeur	800	425	539
Hauteur	600	250	285
Surface	480 000	**106 250**	**153 615**
Largeur	1 280	914	1 026
Hauteur	1 024	705	723
Surface	1 269 760	**644 370**	**741 798**

Tableau 5.1 : Comparaison des surfaces de la fenêtre et de la carte pour les deux versions d'Hypercarte

Hauteurs et largeurs sont exprimées en pixel. Les surfaces sont exprimées en pixel².

Le tableau 5.2 montre le rapport entre la surface de la carte et celle de la fenêtre pour chaque version d'Hypercarte à deux résolutions de fenêtre différentes. Ce tableau montre également le gain sur la surface de la carte entre la version 0.9 et la version 1.0 pour les deux résolutions.

Fenêtre	Rapport Carte / Fenêtre		Gain surfacique de la carte
	Version 0.9	Version 1.0	
800 × 600	22,1 %	32,1 %	44,6 %
1280 × 1024	49,2 %	56,6 %	15,1 %

Tableau 5.2 : Rapport surfacique carte/fenêtre pour les deux versions d'Hypercarte et gain surfacique de la carte, à différentes résolutions

5.7.2. Utilisation des critères ergonomiques de Bastien et Scapin

Les critères de Bastien et Scapin [BAST, 93] guident nos choix ergonomiques. Tout d'abord, nous nous intéressons à un cas d'utilisation de ces critères. Puis nous détaillons les améliorations apportées à l'interface en se conformant à six d'entres eux.

5.7.2.1. Cas d'utilisation des critères ergonomiques

L'espace de référence et la valeur de référence, utilisés pour le calcul de la carte de déviation globale, sont saisis dans une zone de formulaire comprenant plusieurs contrôles ayant entre eux une interaction forte.

Dans la version 0.9, ces contrôles sont deux étiquettes, deux cases à cocher, une liste déroulante, une zone de saisie et un bouton (cf. figure 5.29). Lorsque la première case est cochée, la zone de saisie et le bouton sont désactivés ; lorsque la deuxième case est cochée, la liste déroulante est verrouillée.

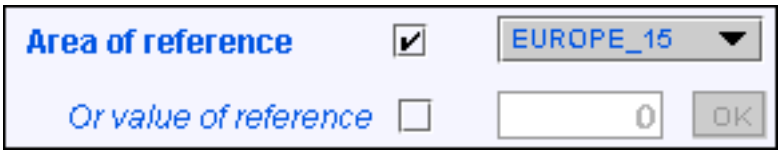


Figure 5.29 : Saisie de l'espace ou de la valeur de référence dans la version 0.9

Dans cette version, certains critères ergonomiques de Bastien et Scapin ne sont pas respectés :

- Lisibilité : les étiquettes et les cases à cocher ne sont pas alignées.
- Homogénéité/cohérence : police du texte de l'étiquette correspondant à la case cochée est en gras, celle du texte de l'autre étiquette étant en italique.
- Concision : l'alternative est visible à la fois dans le libellé de la deuxième étiquette (« *Or* ») et dans la proximité des deux cases à cocher.
- Densité informationnelle : la qualificatif « *of reference* » est utilisé deux fois.
- Groupement/distinction par le format : les cases à cocher ont le comportement de boutons radio exclusifs sans en avoir l'apparence.

La figure 5.30 illustre la manière dont ces points ont été résolus.

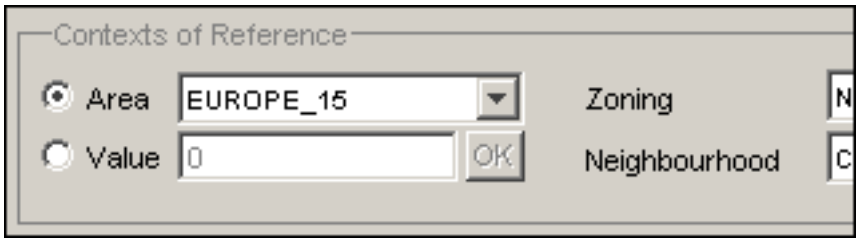


Figure 5.30 : Saisie de l'espace ou de la valeur de référence dans une première mouture de la version 1.0

Lorsqu'un espace de référence est sélectionné, la valeur de référence est la moyenne des UT de cet espace. Comme la valeur de référence de la déviation globale est toujours qualifiée, l'affichage de cette valeur à tout moment a un sens voire une utilité. Dès lors, la zone de saisie ne doit plus être liée à un seul bouton radio et les boutons radio perdent leur raison d'être (cf. figure 5.31).

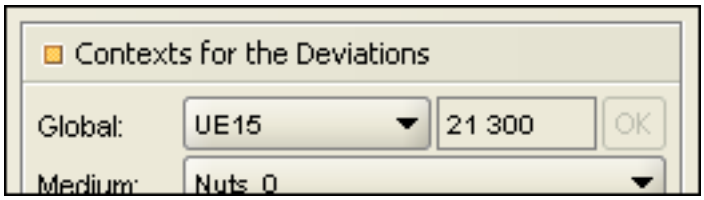


Figure 5.31 : Saisie de l'espace de référence dans le version 1.0

Nous insérons une valeur particulière dans la liste des espaces de référence qui, lorsqu'elle est sélectionnée, déverrouille la zone de saisie (cf. figure 5.32). Ainsi nous réduisons encore la densité informationnelle et nous augmentons la lisibilité et la concision.

Figure 5.32 : Saisie de la valeur de référence dans la version 1.0

5.7.2.2. Critère « Flexibilité »

Le logiciel Hypercarte s'adresse à deux populations d'utilisateurs : les décideurs et les scientifiques. Or, d'après [BAST, 93], « le critère Flexibilité concerne les moyens mis à la disposition des utilisateurs pour personnaliser l'interface afin de rendre compte de leurs stratégies ou habitudes de travail ».

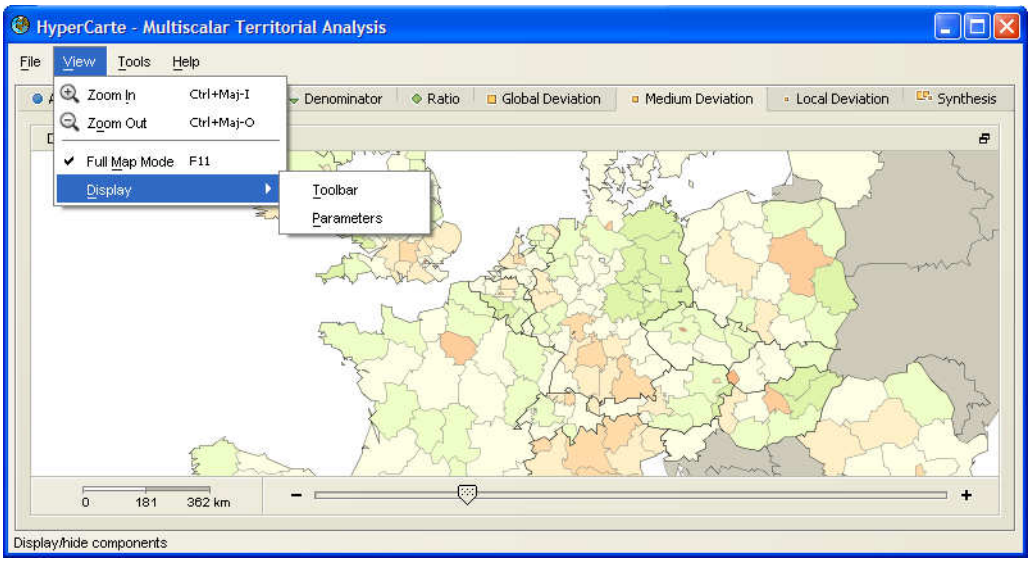


Figure 5.33 : Mode « carte en pleine fenêtre »

Ainsi, certains composants graphiques sont masquables – comme la barre d'outils, le panneau de paramètres, le panneau d'affichage des informations d'UT, etc., ou agrandissables – comme le panneau d'affichage des informations d'UT ou la carte (cf. figure 5.33).

5.7.2.3. Critère « Prise en compte de l'expérience de l'utilisateur »

Hypercarte valide le critère « Prise en compte de l'expérience de l'utilisateur » en permettant à l'utilisateur d'effectuer une action de différentes manières.

Par exemple, le facteur de zoom d'une carte peut être modifié de trois manières différentes :

- en choisissant `View > Zoom In` ou `View > Zoom Out`, dans la barre de menus,
- en utilisant les raccourcis `Ctrl+Maj+I` ou `Ctrl+Maj+O`,
- en actionnant la molette de la souris.

Par exemple, le panneau de paramètre et la barre d'outils peuvent être masqués et la carte peut être élargie de trois façons :

- en agissant individuellement sur chaque composant (clic sur ❐ pour agrandir la carte, choix de `View > Display > Toolbar` dans la barre de menus, etc.),

- en choisissant `View > Full Map Mode` dans la barre de menus,
- en pressant la touche F11.

5.7.2.4. Critère « Groupement/distinction par la localisation »

« Les utilisateurs auront plus de facilité à repérer les différents items s'ils sont présentés de façon organisée » [BAST, 93]. Or, dans l'interface d'Hypercarte, le panneau de paramètres et les onglets de choix de la carte présentent les mêmes types d'information (espace et maillage, indicateur, déviations). Par conséquent, le panneau de paramètres est réorganisé selon un axe horizontal, où les champs sont regroupés par type d'information, et où les types d'information sont ordonnés de la même manière que pour les onglets.

5.7.2.5. Critère « Homogénéité/cohérence »

Dans la version 0.9, les intitulés de champs de saisie et les intitulés d'onglets ne sont pas cohérents. Par exemple, un premier groupe de paramètres s'intitule « *Contexts of reference* », un second groupe se nomme « *Space and zoning* ». Or, l'onglet de la carte qui présente ce dernier s'intitule « *Space and contexts* ». De plus, certains intitulés d'onglets reflètent la valeur de paramètres. Ainsi, lorsqu'une valeur change, ces intitulés d'onglets sont modifiés. Cette instabilité est source de confusion pour l'utilisateur.

Type	Intitulés	
	Version 0.9	**Version 1.0**
🗐	*Space and zoning*	*Area and Zoning*
□	*Study Area*	*Study Area*
□	*Elementary zoning*	*Elementary Zoning*
🌍	*Space and contexts*	*Area and Zoning*
🗐	*Indicator*	*Indicator*
□	–	*Numerator*
□	–	*Denominator*
🌍	L'intitulé du numérateur choisi	*Numerator*
🌍	L'intitulé du dénominateur choisi	*Denominator*
🌍	L'intitulé du numérateur choisi / L'intitulé du dénominateur choisi	*Ratio*
🗐	*Contexts of reference*	*Contexts for the Deviations*
□	*Area of reference*	*Global*
□	*Zoning of reference*	*Medium*
□	*Neighbourhood of reference*	*Local*
🌍	Le code du maillage choisi / L'intitulé de l'espace de référence choisi	*Global*
🌍	Le code du maillage choisi / Le code du maillage de référence choisi	*Medium*
🌍	Le code du maillage choisi / L'intitulé du voisinage de référence choisi	*Local*
🌍	*Synthesis*	*Synthesis*

Tableau 5.3 : Table de correspondance des intitulés entre les deux versions d'Hypercarte

Les types d'intitulés sont les suivants : groupe de paramètres (🗐), paramètre (□) et carte (🌍).

Dans la version 1.0, nous partons du principe que le libellé d'un champ doit indiquer la fonction plutôt que le type de données du champ. De même, l'intitulé d'un onglet doit indiquer le type de la carte plutôt que les valeurs des champs qui lui servent de paramètres (cf. tableau 5.3).

5.7.2.6. Critères « Groupement/distinction par le format » et « Signifiance des codes et dénominations »

Pour chacun des trois groupes de paramètres, nous avons défini arbitrairement une icône. Ces icônes sont de formes et de couleurs différentes. Nous avons évité les formes triangulaires et les couleurs rouges ou orangés qui, par convention [GNOM, 02], ont une signification forte de danger ou d'erreur. Le contexte est symbolisé par un disque bleu, l'indicateur par un losange vert et la déviation par un carré jaune (cf. tableau 5.4).

En reprenant la symbolique de la fraction, l'icône de l'indicateur a été coupée horizontalement pour construire les icônes du numérateur (partie haute) et du dénominateur (partie basse).

Nous avons redimensionné l'icône du groupe de paramètres des déviations pour fournir une icône pour chaque type de déviation : un grand carré pour la déviation globale, un carré de taille moyenne pour la déviation moyenne et un petit carré pour la déviation locale. La synthèse des déviations est le regroupement des carrés de trois tailles.

Groupe de paramètres		**Carte**	
Icône	**Description**	**Icône**	**Description**
	Contexte		Contexte
	Indicateur		Numérateur
			Dénominateur
			Rapport
	Déviations		Déviation globale
			Déviation moyenne
			Déviation locale
			Synthèse des déviations

Tableau 5.4 : Description des icônes thématiques

5.7.3. Nouveau *look and feel*

Une application est généralement construite exclusivement avec des composants graphiques standard comme les boutons, les listes déroulantes, les tableaux de valeurs, etc. Dans le cas d'Hypercarte, certains composants spécifiques telles les cartes et leur légende sont très colorés. L'utilisation des PLAF fournis en standard dans Java, trop contrastés, ne permet pas une bonne lisibilité de l'application. Nous avons mis en place différentes méthodes pour rendre les composants graphiques standard plus discrets tout en améliorant la lisibilité

générale de l'application : utilisation de dégradés, affinage des bordures, utilisation d'un PLAF, modification du rendu des onglets. En outre, le choix du thème utilisé par le PLAF est laissé à l'utilisateur.

5.7.3.1. Utilisation du PLAF `PlasticXPLookAndFeel`

Nous utilisons le PLAF `PlasticXPLookAndFeel`[60] de JGoodies [JGOO, 04]. Ce PLAF est épuré et discret. Il permet d'obtenir un rendu nuancé des contrôles (cf. figure 5.34). Il est libre d'utilisation.

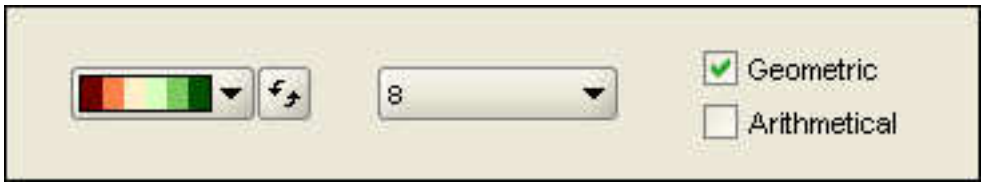


Figure 5.34 : Exemple de contrôles de formulaire avec le PLAF `PlasticXPLookAndFeel`

5.7.3.2. Bordures

5.7.3.2.1. Suppression de bordures

Les bordures sont habituellement utilisées pour encadrer les composants graphiques et non pas pour faire des séparations. Deux composants contigus possédant des bordures sont donc séparés par deux bordures. Ces dernières sont redondantes et réduisent la lisibilité de la fenêtre. Ainsi, lorsque cela est visuellement acceptable et techniquement possible, nous supprimons les bordures redondantes.

Figure 5.35 : Barre de statut de la version 1.0

Par exemple la bordure basse de la barre de menus n'est pas nécessaire puisqu'elle fait double usage avec les bordures hautes des panneaux *Area and Zoning*, *Indicator* et *Contexts for the Deviations* (cf. figure 5.19). La bordure haute de la barre de statut est également redondante avec la bordure du panneau à onglets qui contient les cartes (cf. figure 5.35).

5.7.3.2.2. Affinages des bordures

Les encadrements de composants sont habituellement faits avec des bordures en relief de deux ou trois pixels, comme pour le PLAF `WindowsLookAndFeel`[61] de Swing (cf. figure 5.36). Elles sont constituées d'une bordure 2D, bordée d'une ombre et/ou d'un éclairage. Les bordures 2D, les ombres et les éclairages sont constitués d'un trait d'un pixel de largeur

[60] `com.jgoodies.plaf.plastic.PlasticXPLookAndFeel`
[61] `com.sun.java.swing.plaf.windows.WindowsLookAndFeel`

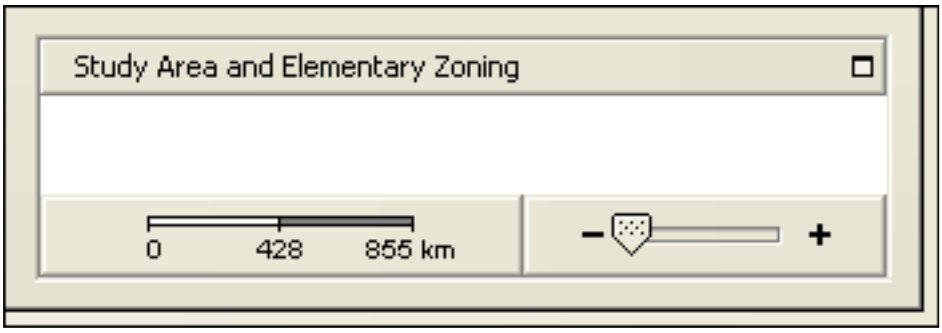


Figure 5.36 : Exemple de bordures 3D avec le PLAF `WindowsLookAndFeel`

Les bordures en relief proposées par le PLAF `PlasticXPLookAndFeel` de JGoodies pour le panneau à onglets (composant `JTabbedPane`) sont fines – entre un et deux pixels. Elles sont graphiquement fausses en ce sens que les bordures 2D et les ombres ne forment qu'un seul trait d'un pixel : nous appelons cet effet « pseudo-3D ». Mais elles sont plus légères et requièrent moins d'espace. Les juxtapositions complexes de panneaux (avec des bordures 3D ou non, en creux ou en relief) ont été harmonisées avec le rendu graphique du `JTabbedPane` de ce PLAF (cf. figure 5.37). La figure 5.38 montre un certain nombre d'écueils à éviter lors de l'utilisation des bordures pseudo-3D.

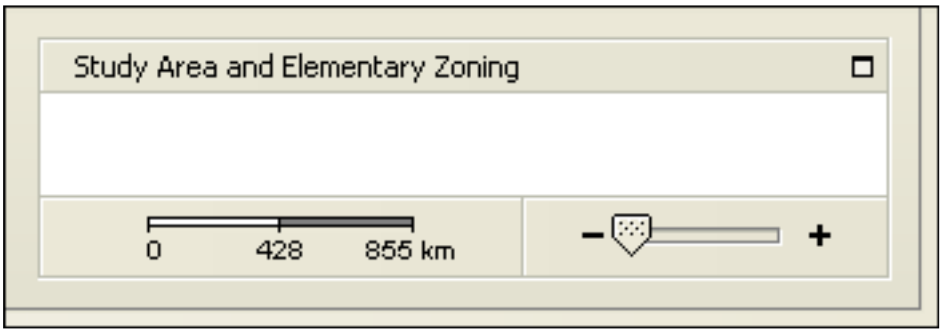


Figure 5.37 : Exemple de bordures pseudo-3D associées au PLAF Plastic

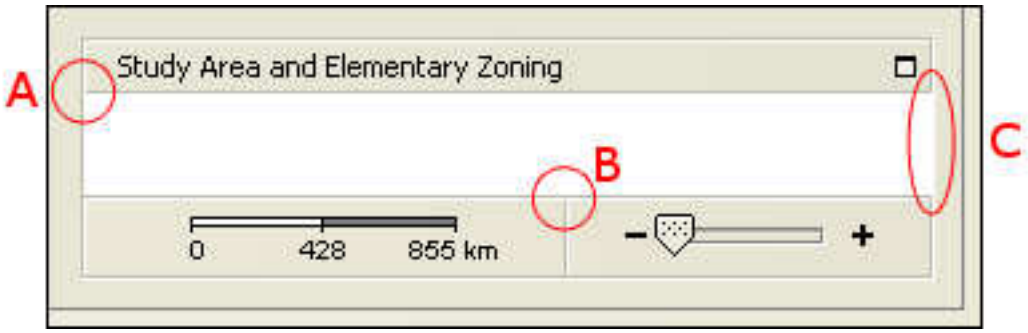


Figure 5.38 : Exemple d'erreurs communes avec les bordures pseudo-3D

Ces erreurs sont de deux types : d'une part, la rupture de la bordure 2D dans un angle (A, B) et, d'autre part, l'absence de bordure 2D sur un côté du composant graphique (C).

5.7.3.3. Modification des onglets

En adoptant le PLAF `PlasticXPLookAndFeel`, les composants de l'interface ont un rendu différent, plus moderne que les PLAF de base de Java. Cependant, certains rendus ne

nous ont pas apporté entière satisfaction. Les onglets des composants `JTabbedPane`, par exemple, sont peu marqués lorsqu'ils sont sélectionnés. Puisque le sens donné à chaque carte est fortement lié à l'onglet sélectionné, nous avons créé une classe `HCPlasticXPLookAndFeel` qui hérite de la classe `PlasticXPLookAndFeel`. Ce nouveau PLAF utilise une nouvelle classe, qui s'appelle `HCPlasticTabbedPaneUI` et qui hérite de `MetalTabbedPaneUI` – la classe de rendu du composant `JTabbedPane` pour le PLAF `Metal`.

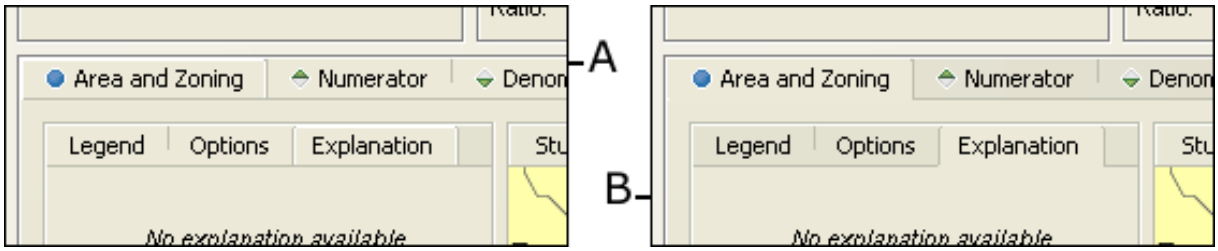


Figure 5.39 : Modification du rendu des onglets

Les onglets sélectionnés ont un rendu peu lisible avec `PlasticXPLookAndFeel` (A), alors que le rendu est plus marqué avec `HCPlasticXPLookAndFeel`, le PLAF que nous avons écrit (B).

Quatre modifications ont été apportées pour améliorer la lisibilité :

- les onglets sont transparents ;
- un dégradé est dessiné derrière l'ensemble des onglets ;
- l'onglet sélectionné n'est plus entouré d'une bordure intérieure blanche ;
- la bordure située en bas de l'onglet sélectionné n'est plus dessinée.

Les effets obtenus sont les suivants (cf. figure 5.39) :

- les onglets non sélectionnés sont plus sombres et sont liés par un dégradé ;
- l'onglet sélectionné est de la même couleur que son panneau et semble faire partie de ce dernier.

5.7.3.4. Possibilité de modifier le thème

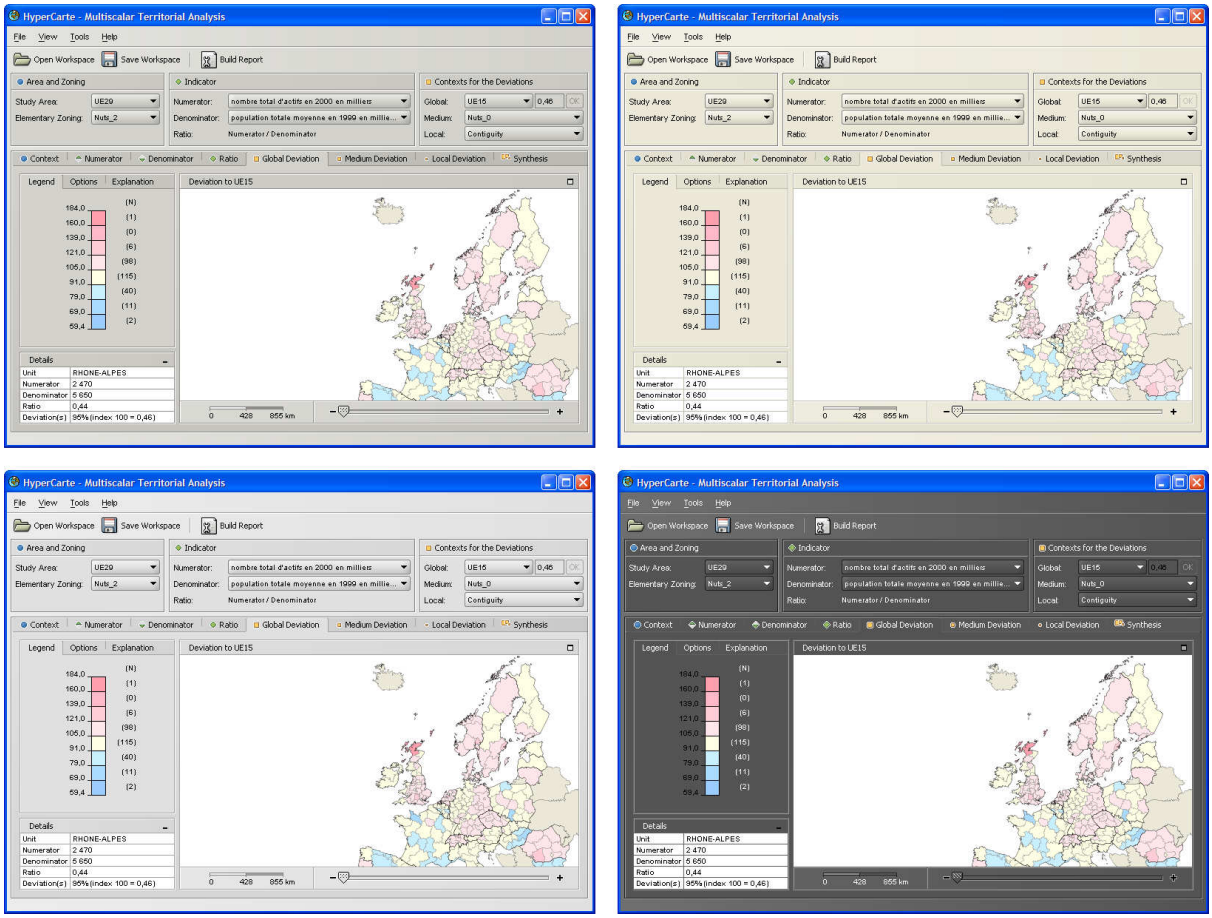


Figure 5.40 : Hypercarte avec différents thèmes du PLAF `PlasticLookAndFeel`

Les thèmes utilisés ici sont de gauche à droite puis de haut en bas : Sky Blue, Experience Blue, Desert Blue *et* Dark Star.

Le PLAF de JGoodies met plusieurs thèmes à notre disposition. La figure 5.40 montre un échantillon des thèmes disponibles. La version 1.0 d'Hypercarte permet de démarrer l'application avec un des thèmes disponibles. Par exemple, la ligne de commande de la figure 5.41 lance Hypercarte en utilisant le thème « *Sky Blue* ».

```
>java -jar hypercarte.jar -SkyBlue
-----------------------
Current theme is SkyBlue.
To change theme, start applet by giving the desired theme as argument.
Available themes are ExperienceBlue, ExperienceGreen, DesertBlue, DesertGreen, Dark
Star and SkyBlue.
-----------------------
```

Figure 5.41 : Ligne de commande pour démarrer Hypercarte

Nous récupérons le paramètre passé à la méthode *main* (cf. code 5.13).

Code 5.13 : Le thème choisi par l'utilisateur est passé en paramètre au PLAF

```
package hypercarte;
// ...
import com.jgoodies.plaf.plastic.PlasticLookAndFeel;
public class Starter extends JApplet
                    implements EventListener, MessageEventListener {
    public static void main(String[] args) {
        // ...
        for (int i = 0; i < args.length; i++) {
            if (args[i].equalsIgnoreCase("-standard")) {
                Settings.getInstance().setCustomer(Settings.CUSTOMER_STANDARD);
            } else if (args[i].equalsIgnoreCase("-ESPON")) {
                Settings.getInstance().setCustomer(Settings.CUSTOMER_ESPON);
            } else {
                plafTheme = args[i];
            }
        }
        // ...
    }
    // ...
}
```

5.7.3.5. Dégradés

Pour mettre en évidence les groupes de contrôles sans ajouter de traits, de bordures ou de couleurs, nous utilisons un dégradé léger. Chaque dégradé se distingue par sa nuance de couleurs et par son orientation (cf. figure 5.42).

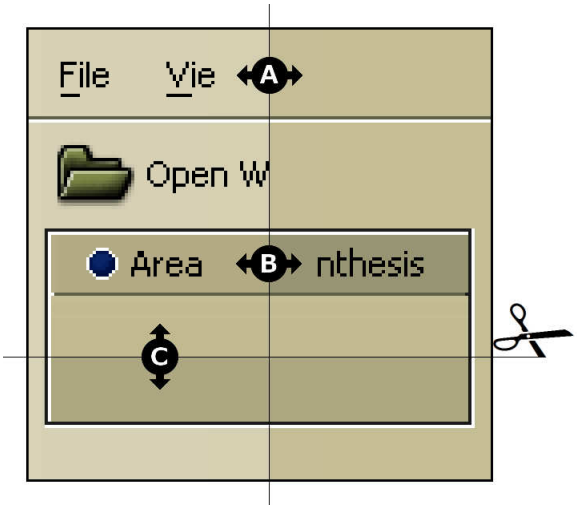


Figure 5.42 : Mise en évidence des dégradés

Les quatre coins de la fenêtre de l'interface d'Hypercarte sont mis côte à côte afin de mettre en évidence les dégradés de couleurs. Nous pouvons voir deux dégradés du clair vers le sombre orientés tous deux de gauche à droite : le premier (A) lie le fond de la fenêtre, la barre de menus, la barre d'outils et la barre de statut ; le second (B) lie les onglets non sélectionnés. Nous pouvons également voir un dégradé du clair vers le sombre orienté du haut vers le bas dans le panneau (C) correspondant à l'onglet sélectionné.

5.7.3.6. Habillage spécial pour ESPON

Un habillage spécial correspondant à la charte graphique d'ESPON a été demandé pour la livraison de la version 1.0 (cf. figure 5.43).

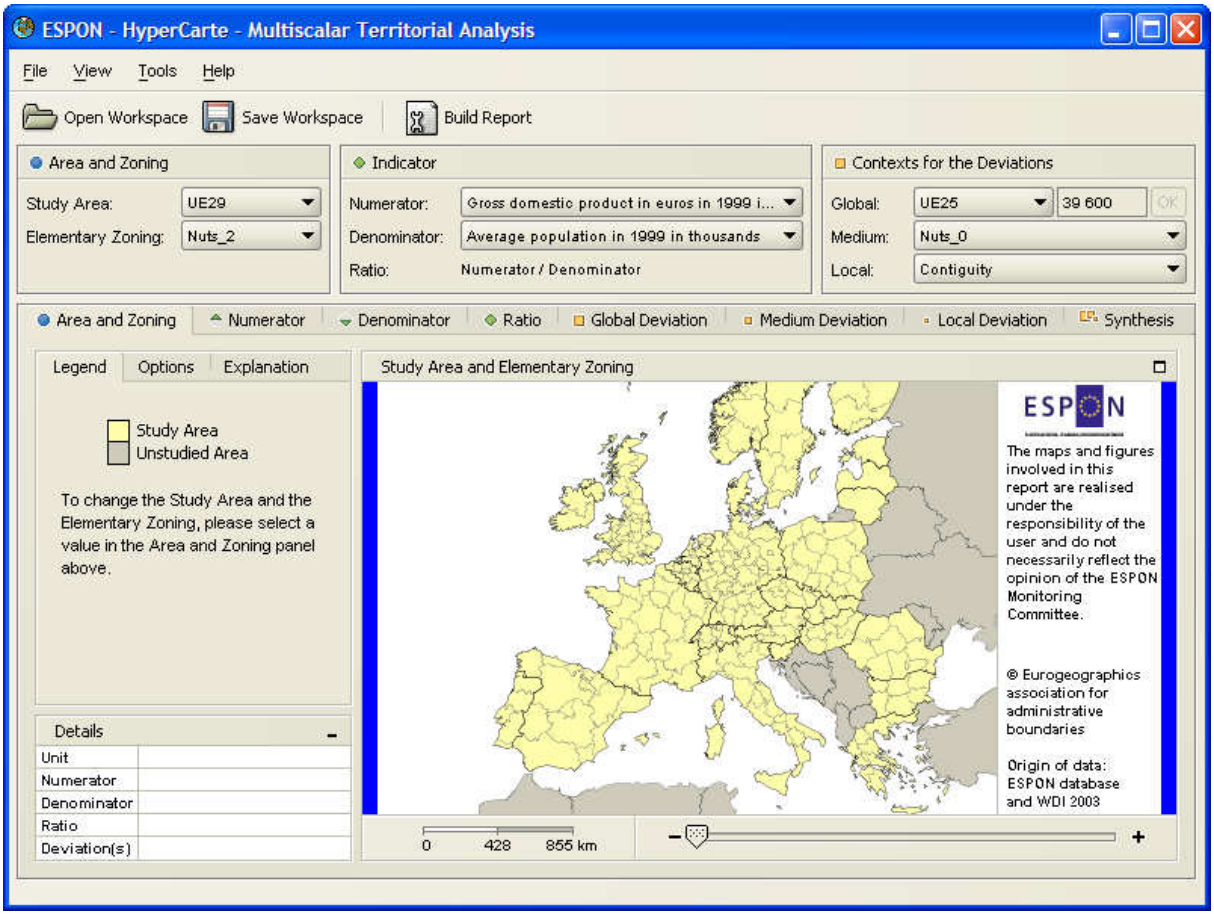


Figure 5.43 : Habillage d'Hypercarte selon la charte graphique d'ESPON

Le choix d'afficher ou non cet habillage est laissé à l'utilisateur. S'il désire une application avec la charte graphique et les mentions de copyright d'ESPON, il doit lancer l'application avec la ligne de commande décrite dans la figure 5.44.

```
>java -jar hypercarte.jar -ESPON
```

Figure 5.44 : Ligne de commande permettant d'utiliser l'habillage d'ESPON

5.8. Synthèse

5.8.1. Bilan

Les deux maîtres mots de notre action sont « simplification » et « homogénéisation » :

- simplification et homogénéisation de l'application, en réécrivant le mécanisme de communication entre les composants, en regroupant la gestion des paramètres et en isolant la logique applicative ;

- simplification du code en supprimant les branchements conditionnels grâce à l'héritage de classes, ce qui a le double effet d'augmenter le nombre de classes et de réduire leur complexité ;

- simplification et homogénéisation de l'interface en améliorant l'ergonomie : l'utilisateur doit comprendre du premier coup d'œil le rôle de chaque élément de l'interface et appréhender rapidement le fonctionnement de l'application.

En outre, les performances en terme d'affichage et en terme de manipulation de données ont été améliorées, notamment par l'utilisation de caches.

Enfin, nous avons ajouté des fonctionnalités périphériques comme la sauvegarde et la restauration du contexte ou la génération de rapport.

5.8.2. Quelques chiffres

La version 0.9 est constituée de 54 classes Java pour 11265 lignes de code source. Le *bytecode* généré est enregistré dans un fichier JAR[62] d'une taille de 182 Ko. Cette version d'Hypercarte se lance en 27 secondes.

La version 1.0 est constituée de 139 classes Java[63] pour 24005 lignes de code source. Le *bytecode* fait 624 Ko. L'application se lance en 9 secondes.

[62] JAR : format de fichier Java Archive.
[63] Auxquelles il faut ajouter les 111 classes de la bibliothèque JGoodies.

Conclusion et perspectives

Rappel des objectifs

Le prototype du logiciel Hypercarte répond aux attentes des géographes en ce qui concerne les fonctionnalités minimales requises. Toutefois, il présente encore des insuffisances au niveau de son architecture, des temps de réponse et de l'ergonomie. Nos trois objectifs principaux sont d'une part de réarchitecturer l'application et de restructurer son code, d'autre part d'améliorer les performances en terme d'affichage et de calcul, et enfin de finaliser l'interface.

Conclusion

En l'absence de spécifications détaillées, et en présence d'un prototype dont le code est peu documenté, la réalisation de la version 1.0 du logiciel Hypercarte a nécessité un gros travail de rétro-ingénierie préalable à sa réécriture.

Une fois le fonctionnement du prototype assimilé et le fonctionnel de l'application acquis, la réécriture de notre outil s'est déroulée en trois phases successives :

- La phase de construction comporte deux tâches : la première consiste à réécrire unitairement chaque classe en fonction de règles de codage et de nommage ; la seconde tâche concerne la réorganisation des classes, à travers la spécialisation et la généralisation, grâce au mécanisme d'héritage.

- La phase d'affûtage consiste à améliorer les performances en terme de rapidité d'affichage, de vitesse de calcul ou de réduction du temps de chargement de données.

- La phase de façonnage, par l'application de règles ergonomiques, permet d'assurer un prise en main aisée de l'outil, de lui donner un aspect rassurant et agréable, donc attractif, et enfin de présenter une interface sobre et complète à la fois, que l'utilisateur peut adapter à ses besoins.

Des fonctionnalités de type import/export ont été incluses dans la version 1.0, comme la génération de rapports ou la sauvegarde et la restauration d'un espace de travail.

La collaboration fructueuse entre les trois équipes du projet Hypercarte a permis de résoudre tous les problèmes rencontrés.

Même si les concepts mis en œuvre dans ce logiciel datent un peu[64], le logiciel Hypercarte reste un formidable outil pour les géographes et les décideurs, comme en témoignent les

[64] La première publication mentionnant les notions d'analyse spatiale multiscalaire date de 1997 [GRAS, 97].

réactions recueillies au cours de démonstrations faites auprès d'utilisateurs, et les perspectives sont vastes et multiples.

Perspectives

Nous distinguons trois champs de perspectives : le champ correctif, le champ d'amélioration et le champ prospectif.

Corrections

Pour gagner encore en lisibilité et en rapidité de traitement et d'affichage, il serait intéressant de faire varier la définition de la carte en fonction du facteur de zoom.

Dans les versions 0.9 et 1.0, la translation de la carte et le zoom sont mal corrélés. En effet, la translation est appliquée à la carte après le calcul du zoom, ce qui a pour conséquence de recentrer ou d'excentrer la carte lors de l'augmentation ou de la diminution du facteur de zoom. Il pourrait être envisagé de centrer le zoom sur un point de la carte choisi par l'utilisateur. Ce point, jugé non prioritaire par les géographes, est resté en suspens faute de temps.

Les cartes incluses dans le rapport de la version 1.0 sont des copies d'écran effectuées applicativement qui reprennent les caractéristiques de l'affichage en cours (taille de la carte, légende ou options masquées) selon le principe WYSIWYG[65]. Tout en laissant le choix du facteur de zoom, du centrage et de la taille de carte à l'utilisateur, il serait intéressant d'inclure pour chaque carte du rapport un panneau unique regroupant les informations de légende, d'options et d'échelle.

Améliorations

La version 1.0 du logiciel charge un jeu de données préparé préalablement. Le développement d'un module de chargement d'un jeu de données fourni par l'utilisateur (cf. annexe 5) pourrait être envisagé pour rendre l'application plus attractive.

Pour traiter des cartes très détaillées, ou pour fournir à l'utilisateur le choix parmi une grande quantité de statistiques, une partie des calculs des données (et de la carte ?) devra être déportée sur une machine dédiée. La mise en place de cette architecture client-serveur nécessite de résoudre des problèmes comme la vitesse de calcul de la machine distante ou le temps de transmission, etc.

Voies d'exploration

Le logiciel Hypercarte répond à une attente des géographes qui est l'analyse territoriale multiscalaire. D'autres voies d'explorations pourraient lui être adressées.

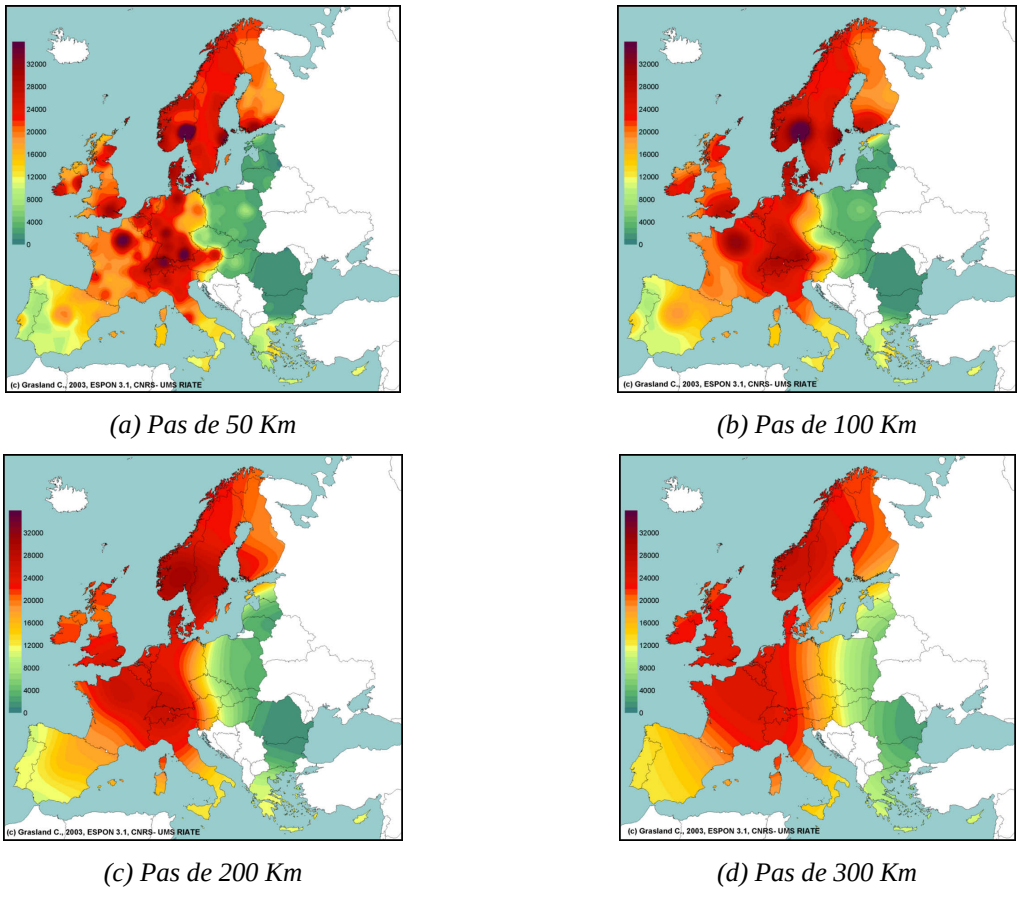
---

[65] WYSIWYG (*What You See Is What You Get*) : « ce que vous voyez est ce que vous obtenez ».

L'intégration du temps, avec la variation des territoires dans le temps, est assurément une problématique intéressante. A travers le développement d'autres SIG, les membres de l'axe 'Multimédia-Web' de l'équipe Sigma possèdent une expertise dans ce domaine.

La mise en place de profils utilisateurs et l'implémentation d'une forme d'adaptabilité de l'application sont également un axe de recherche possible.

L'actuelle version du logiciel Hypercarte implémente le module MTA. La réalisation du module MSA, implémentant des méthodes de lissage, reste à spécifier et à développer (cf. figure 7.1).



(a) Pas de 50 Km *(b) Pas de 100 Km*

(c) Pas de 200 Km *(d) Pas de 300 Km*

Figure 7.1 : Méthode de lissage gaussien avec différents pas, d'après [GRAS, 04]

Bilan personnel

D'un point de vue technique, j'ai apprécié de m'initier à Java, de travailler avec la modélisation objet, et d'approfondir mes connaissances en UML, Java, XML. J'ai découvert avec intérêt les domaines de recherche du projet Hypercarte que sont la géomatique et la géostatistique.

D'un point de vue plus personnel, j'ai eu l'opportunité de participer à une des conférences internationales auxquelles la version 1.0 d'Hypercarte a été présentée. Au cours de cette

année, j’ai également eu la chance de faire mes premiers pas dans l’enseignement universitaire. Ce sont des expériences très enrichissantes.

Bibliographie

[ADOB, 01] Adobe, 2001. *Adobe SVG Viewer*. Disponible en ligne : http://www.adobe.com/svg/.

[APPL, 04] Apple, 2004. *Apple Human Interface Guidelines*. Disponible en ligne : http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/OSXHIGuidelines.pdf.

[BAST, 93] Bastien, J.M.C., Scapin, D.L., 1993. *Preliminary findings on the effectiveness of ergonomic criteria for the evaluation of human-computer interfaces*. Conference on Human Factors in Computing Systems, INTERCHI'93, Amsterdam, Pays-Bas.

[BATI, 04] Batik SVG Toolkit, de l'Apache Software Foundation. http://xml.apache.org/batik/.

[BISS, 04] Bissler, T., 2004. *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.

[CNRS, 00] CNRS, 2000. *Guide de recommandations ergonomiques pour la conception et l'évaluation d'interfaces graphiques*. Disponible en ligne : http://www.dsi.cnrs.fr/bureau_qualite/ergonomie/documentation/guidergo.pdf.

[CSS, 04] W3C, 2004. *Cascading Style Sheets, level 2 revision 1*. Recommandation CSS version 2.1, mise à jour le 25 février 2004. Disponible en ligne : http://www.w3.org/TR/2004/CR-CSS21-20040225/.

[DANZ, 03] Danzart, A., Moissinac, JC., Potier, C, 2003. *Standards pour la cartographie animée sur Internet*. Revue internationale de Géomatique, volume 13, n°1/2003.

[DOM, 04] W3C, 2004. *Document Object Model (DOM) Level 3 Core Specification*. Spécification DOM niveau 3, version 1.0, mise à jour le 7 avril 2004. Disponible en ligne : http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/.

[DOUS, 99] Doussat, G., Germond, J., 1999. *Systèmes d'Information Géographique et Web : SIRVA, application à la gestion des risques naturels*. Rapport de stage année spéciale ENSIMAG, 1998-1999.

[ECLI, 04] Eclipse de l'*Eclipse Foundation*. http://www.eclipse.org/.

[ENSE, 03] Ensenlaz, C., 2003. *Finalisation d'une applet Java permettant la consultation des événements avalancheux recensés sur la commune de Vallorcine (Haute-Savoie)*. Mémoire de stage IUT2 Grenoble, 2002-2003.

[ESRI, 98] Environmental Systems Research Institute, 1998. *ESRI Shapefile Technical Description, An ESRI White Paper, July 1998*. Description technique du format de fichier ESRI Shapefile. Disponible en ligne : http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

[ESRI, 04] ArcGIS, de ESRI. http://www.esri.com/.

[FLAS, 03] Macromedia, 2003. *Macromedia Flash (SWF) File Format Specification*. Spécification du format de fichier Macromedia Flash (SWF). Disponible en ligne : http://www.macromedia.com/software/flash/open/licensing/fileformat/,

[FLEU, 98] Fleury, P., Bhattacharjee, S., Piron, L., Ebrahimi, T., Kunt, M., 1998. *MPEG-4 Video Verification Model: A Solution for Interactive Multimedia Applications*. Journal of Electronic Imaging, volume 7, pp. 502-515, juillet 1998.

[GEOC, 04] GéoClip de eMc3. http://www.geoclip.net/.

[GEOT, 04] GeoTools. http://geotools.codehaus.org/.

[GNOM, 02] GNOME Project, 2002-2004. *GNOME Human Interface Guidelines 2.0*. Disponible en ligne : http://developer.gnome.org/projects/gup/hig/.

[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.

[GRAS, 97] Grasland, C., 1997. *Spatial homogeneity and territorial discontinuities*. Séminaire de Géostatistique, INSEE. Mai 1997.

[GRAS, 00] Grasland, C., 2000. *Spatial Homogeneity and Territorial Discontinuities. Conference of European Statisticians*. UN/ECE Work Session on Methodological Issues Involving the Integration of Statistics and Geography, Neuchâtel, Suisse, 10-12 avril 2000. Disponible en ligne : http://www.unece.org/stats/documents/2000/04/gis/11.e.pdf.

[GRAS, 03a] Grasland, C., Lizzi, L., Martin, H., Mathian, H., Vincent, J.M., 2003. *Hypercarte : un outil d'analyse spatiale multiscalaire des inégalités régionales en Europe*. XXXIX^{ème} colloque de l'Association de Science Régionale de Langue Française : Concentration et ségrégation, dynamiques et inscriptions territoriales, Lyon, France, 1, 2 et 3 Septembre 2003. Disponible en ligne : http://asrdlf2003.entpe.fr/pdfpapiers/A2/211.pdf.

[GRAS, 03b] Grasland, C., Lizzi, L., 2003. *ESPON Project 3.1. Integrated Tools for European Spatial Development. Annex A: Multiscalar Territorial Analysis (3rd version)*. Disponible en ligne : http://www.espon.lu/online/documentation/projects/cross_thematic/1233/3.ir-3.1_annex_a.pdf.

[GRAS, 04] Grasland, C., 2004. *ESPON 31 FR Dictionary of Spatial Analysis Tools, part 2*. Dictionnaire des outils d'analyse spatiale réalisé dans le cadre d'ORATE 3.1.

[KHAM, 98] Khamphanh, V., 1998. *Développement d'une base de données pour un Système d'Information sur les Risques Naturels de la Haute Vallée de l'Arve*. Mémoire de stage IUT2 Grenoble, 1997-1998.

[J2DA, 04] Sun Microsystems, 2004. *Java 2D API*. Disponible en ligne : http://java.sun.com/products/java-media/2D/.

[JGOO, 04] JGoodies – Java User Interface Design de Karsten Lentzsch. http://www.jgoodies.com/.

[JTS, 04] Vivid Solutions, 2003. *JTS Topology Suit Version 1.3, Technical Specifications*. Disponible en ligne : http://www.vividsolutions.com/jts/

[JUMP, 04] JUMP – Unified Mapping Platform, de Vivid Solutions. http://www.vividsolutions.com/jump/

[KRAM, 96] Kramer, D., Joy, B., Spenhoff, D., 1996. *The Java ™ Platform – A White Paper*. Document technique, mai 1996. Disponible en ligne : http://java.sun.com/docs/white/platform/javaplatformTOC.doc.html.

[MAPI, 03] MapInfo, 2003. *MapInfo Professional v7.5 User Guide, Annex D: MapInfo Map Interchange Format*. Spécification des formats de fichier MIF et MID, 2003. Disponible en ligne : http://extranet.mapinfo.com/documentation/software/mapinfo_pro/075/english/MI_UG.pdf.

[MAPI, 04] MapInfo. http://www.mapinfo.com/.

[MAPS, 04] MapServer. http://mapserver.gis.umn.edu/.

[MART, 04] Martin, Ph., 2004. *Développement d'une interface cartographique d'analyse spatiale multiscalaire de phénomènes sociaux dans le cadre du projet Hypercarte pour l'étude et l'aménagement du territoire européen. Mémoire d'ingénieur CNAM*. Non publié.

[MOIS, 02] Moisuc, B., 2002. Réalisation d'une interface cartographique dans le cadre du projet Hypercarte Europe. Rapport de stage DESS DCISS Université Pierre Mendès-France, Grenoble 2, France.

[MICR, 04] Microsoft, 2004. *Official Guidelines for User Interface Developers and Designers*. Disponible en ligne : http://msdn.microsoft.com/library/en-us/dnwue/html/welcome.asp.

[MOZI, 04] Mozilla Project, 2004. *Mozilla SVG Project*. Projet de navigateur Web supportant SVG nativement. Site du projet en ligne : http://www.mozilla.org/projects/svg/.

[OGC, 99] Open GIS Consortium, 1999. *OpenGIS® Simple Features Specification For SQL, Revision 1.1*. Spécification de format de bases de données géospatiales,

révision 1.1, mai 1999. Disponible en ligne : http://www.opengeospatial.org/docs/99-049.pdf.

[OGC, 01] Open GIS Consortium, 2001. *Web Map Service Implementation Specification, Version 1.1.1*. Spécification d'implémentation des services Web cartographiques, novembre 2001. Disponible en ligne : http://www.opengeospatial.org/docs/01-068r2.pdf.

[OGC, 03] Open GIS Consortium, 2003. *OpenGIS® Geography Markup Language (GML) Implementation Specification, Version 3.0.* Spécification d'implémentation de GML, janvier 2003. Disponible en ligne : http://www.opengeospatial.org/docs/02-023r4.pdf.

[OMON, 04] EclipseUML de Omondo. http://www.omondo.com/.

[OPEN, 04] OpenMap, de BBN Technologies. http://openmap.bbn.com/

[PNG, 03] W3C, 2003. *Portable Network Graphics (PNG) Specification (Second Edition), Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E).* Spécification du PNG, deuxième édition, novembre 2003. Disponible en ligne : http://www.w3.org/TR/2003/REC-PNG-20031110/.

[ROSA, 02] Rosa-Puissant, M., 2002. *Développement d'une applet permettant la consultation des avalanches passées sur la commune de Vallorcine*. Mémoire de stage IUT2 Grenoble, 2001-2002.

[ROUS, 02] Roussillat, T., 2002. *Conception et réalisation d'une applet permettant la consultation des avalanches passées sur la commune de Vallorcine*. Mémoire de stage IUT2 Grenoble, 2001-2002.

[SMIL, 01] W3C, 2001. *Synchronized Multimedia Integration Language (SMIL 2.0)*. Recommandation SMIL version 2.0, mise à jour le 7 août 2001. Disponible en ligne : http://www.w3.org/TR/smil20/.

[SVG, 03] W3C, 2003. *Scalable Vector Graphics (SVG) 1.1 Specification*. Recommandation SVG version 1.1, mise à jour le 14 janvier 2003. Disponible en ligne : http://www.w3.org/TR/2003/REC-SVG11-20030114/ .

[TESN, 03] Tesnière, B., 2003. *Développement d'une interface Web pour la consultation d'information territoriale liée aux risques d'avalanches sur la commune de Vallorcine (Haute-Savoie)*. Mémoire de stage IUT2 Grenoble, 2002-2003.

[TROU, 02] Trouillon, A., 2002. *Conception et Réalisation d'un Système d'Information Historique pour la gestion des risques d'inondations*. Mémoire d'ingénieur CNAM soutenu le 16 janvier 2002.

[VILL, 98] Villanova, M., 1998. *Système d'Information sur les risques naturels de la haute vallée de l'Arve*. Rapport de DESS double compétence Informatique et Sciences Sociales, 1997-1998.

[XHTM, 01] W3C, 2001. *XHTML™ 1.1 - Module-based XHTML*. Recommandation XHTML version 1.1, mise à jour le 31 mai 2001. Disponible en ligne : http://www.w3.org/TR/2001/REC-xhtml11-20010531/.

[XML, 04] W3C, 2004. *Extensible Markup Language (XML) 1.0 (Third Edition)*. Recommandation XML version 1.0, mise à jour le 4 février 2004. Disponible en ligne : http://www.w3.org/TR/2004/REC-xml-20040204/.

Table des illustrations

Figures

Tableaux

Code

Annexes

Annexe 1 – Réalisation du site Web du projet Hypercarte

Le projet Hypercarte possède plusieurs sources de financement. Dans le cahier des charges du financement par l' « ACI[66] Masse de données », le projet Hypercarte doit avoir un site Web pour rendre compte de son activité. De plus, le projet Hypercarte a besoin d'une vitrine. Le site Web doit être sobre et clair.

Figure 8.1 : Page d'accueil du site Web du projet Hypercarte.

Les pages du site Web sont composées d'un bandeau (A), d'un menu déroulant (B)[67], de liens statiques (C) rappelant les titres de sous-menus et d'un bas de page (D) contenant la mention de copyright, la date de dernière mise à jour et des liens de validation.

En partant du contenu des pages d'un site Web dynamique basé sur PHP[68] et développé pour le projet Hypercarte par un stagiaire d'IUT, nous avons réalisé un site Web simple et fonctionnel. Il est statique, mais néanmoins modulaire grâce à l'utilisation de SSI[69].

[66] ACI : Action Concertée Incitative.

[67] Le menu déroulant que nous avons développé utilise le JavaScript. Si le JavaScript n'est pas activé dans le navigateur, seul le titre de chaque sous-menu est affiché (i.e. le menu ne déroule plus).

[68] PHP (*Hypertext Pre-Processor*) : langage de script exécuté côté serveur qui permet de rendre dynamique le contenu de pages Web.

[69] SSI (*Server-Side Include*) : inclusion côté serveur.

Si le JavaScript est activé dans le navigateur Web, deux pages du site ne sont jamais éloignées de plus de deux sauts hypertextes (i.e. : chaque page du site est accessible en deux sauts hypertextes maximum depuis n'importe quelle autre page du site). Sans JavaScript, le menu n'est plus déroulant. Ainsi, deux pages peuvent être éloignées au maximum de trois sauts hypertextes.

Figure 8.2 : Navigation dans les pages du site Web du projet Hypercarte.

Chaque page présente à l'utilisateur sa position dans l'arborescence logique du site. Ce rappel est composé de liens hypertextes.

Une indication sur la position de la page dans l'arborescence du site Web est affichée sur chaque page (cf. figure 8.2), excepté sur la page d'accueil (cf. figure 8.1).

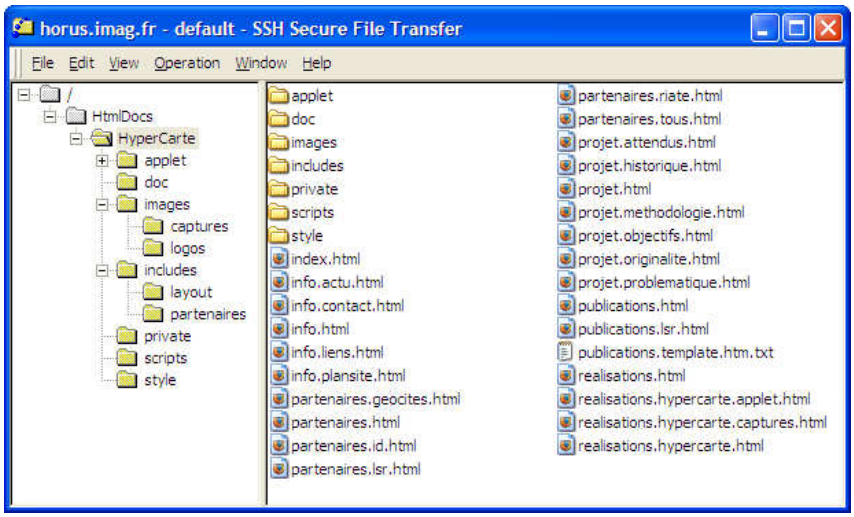


Figure 8.3 : Structure de l'arborescence de fichiers du site Web du projet Hypercarte.

Le nom des fichiers HTML reflètent la structure logique arborescente du site, en suivant la règle de nommage suivante : {<nom> = <menu>[.<structure>].html /\ <structure> =<item>[.<structure>]}.
Par exemple, le fichier realisation.hypercarte.applet.html correspond au sous item « Applet » de l'item « Hypercarte » du menu « Réalisation ».

Nous avons porté un soin particulier à rendre ce code facilement maintenable en adoptant une règle de nommage des fichiers HTML (cf. figure 8.3) et en séparant clairement chaque type de fichiers dans des répertoires dédiés (feuilles de style, scripts coté client, images, captures

d'écran, *includes* etc.). De plus, le code est pérenne puisqu'il est conforme aux recommandations XHTML 1.1 et CSS[70] 2.1 du W3C.

Le site a servi d'espace de dépôt duquel les versions successives du logiciel ont pu être téléchargées par les différents intervenants du projet (cf. figure 8.4).

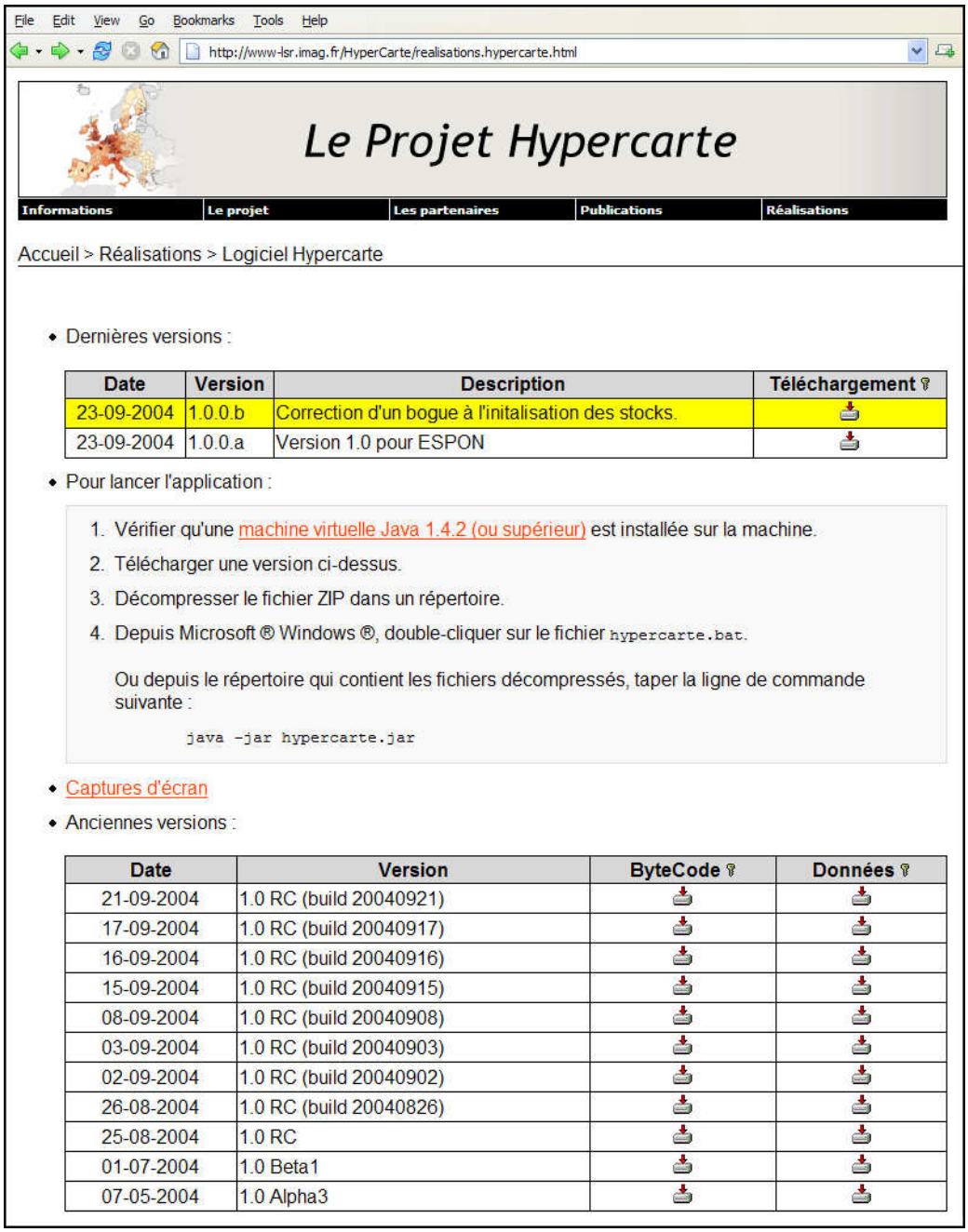
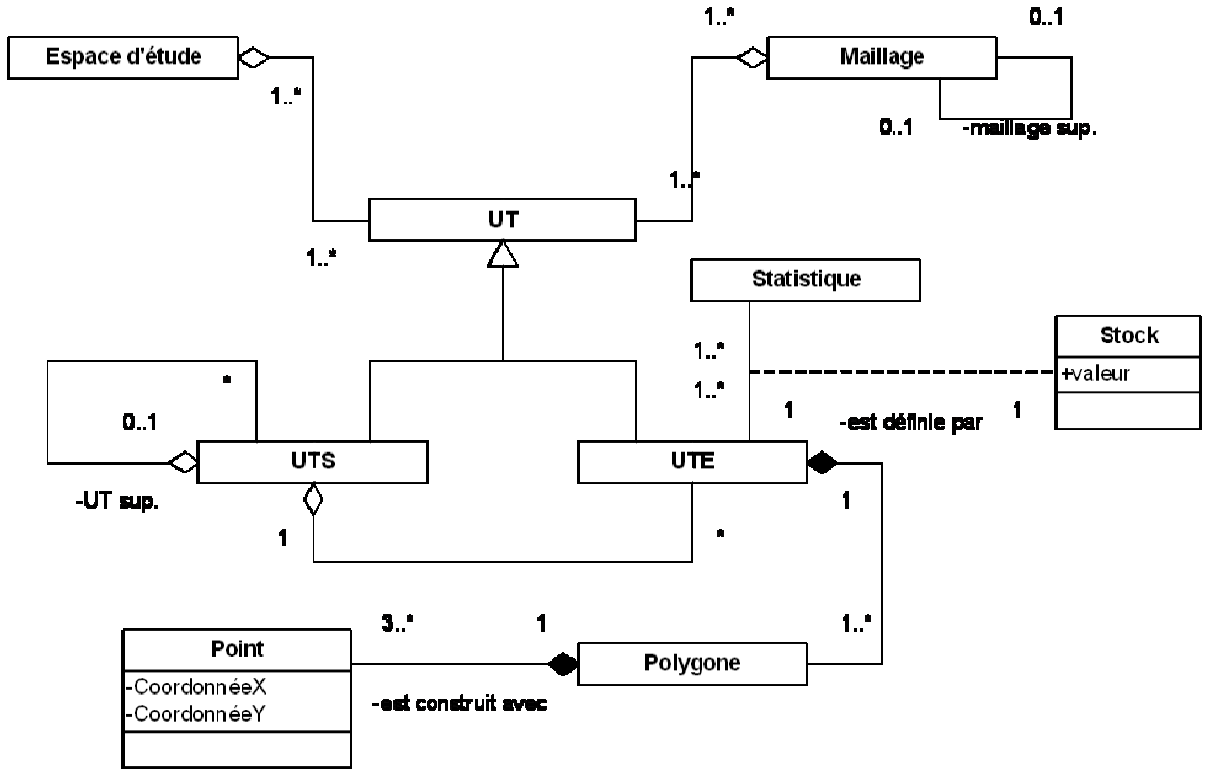


Figure 8.4 : Page de téléchargement du logiciel Hypercarte

[70] CSS (*Cascading Style Sheets*) : feuilles de style en cascade [CSS, 04].

Annexe 2 – Diagramme des concepts utilisés dans Hypercarte

Dans ce schéma, nous présentons le diagrammes de l'ensemble des concepts introduits dans la section 3.2 : unité territoriale, espace, maillage, statistique, stock et géométrie.



Annexe 3 – Code de la classe `Chart` de la version 0.9

Le code suivant est extrait de la classe `hypercarte.interfac.Chart` de la version 0.9 d'Hypercarte. Il illustre l'utilisation de branchements conditionnels pour gérer différents fonctionnements d'un même objet.

```
package hypercarte.interfac;
//...

public class Chart extends JPanel implements  PanelOptionListener,
ChoiceChartListener, Serializable,  DistributionUpdateListener,
PanelOptionZoomListener,PanelOptionInformationListener, ColorUpdateListener,
Cloneable {
  //...
  /* Le type de carte a afficher */
  private int type  = 0;

  // ...

  public void paint(Graphics gg)
  {
    //...
    switch (type)
    {
      case 0://carte gris
      {
        // dessin tout gris
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite, "NUTS2", "NUTS2");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.lightGray, Color.lightGray);
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite, "NUTS0", "NUTS0");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.black, Color.lightGray);
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), "NUTS2");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.black, Color.WHITE);
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceReference(), "NUTS2");
        dessineCarte.frontiereFondListUniteCouleur2(g, units,
panelOption.getIndicatorsDetail(), Color.black, Color.darkGray);
        // Fin de Dessin tout gris
        // dessin ut ele
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsUnitElementary());
        dessineCarte.borderListUnit(g, units, panelOption.getIndicatorsDetail(),
Color.gray);
        // dessin partition
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsPartitionReference());
```

```
        dessineCarte.borderListUnit(g, units, panelOption.getIndicatorsDetail(),
Color.darkGray);
       }
      break;
      case 1://carte avec des cercles
      {
        // dessin tout gris
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite, "NUTS2", "NUTS2");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.lightGray, Color.lightGray);
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite, "NUTS0", "NUTS0");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.black, Color.lightGray);
        // Fin de Dessin tout gris
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), "NUTS2");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.black, Color.WHITE);
        // dessin ut ele
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsUnitElementary());
        dessineCarte.borderListUnit(g, units, panelOption.getIndicatorsDetail(),
Color.gray);
        // dessin partition
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsPartitionReference());
        dessineCarte.borderListUnit(g, units, panelOption.getIndicatorsDetail(),
Color.darkGray);
        // les couleurs
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsUnitElementary());
        dessineCarte.circleListUnitColor(g, units,
panelOption.getIndicatorsDetail(), tailleBoulles, couleurCercle1,
panelOption.getChainsIndicatorA());
      }
      break;
      case 2://carte avec des cercles
      {
        // dessin tout gris
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite, "NUTS2", "NUTS2");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.lightGray, Color.lightGray);
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite, "NUTS0", "NUTS0");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.black, Color.lightGray);
        // Fin de Dessin tout gris
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), "NUTS2");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.black, Color.WHITE);
        // dessin ut ele
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
```

```
panelOption.getChainsSpaceEtude(), panelOption.getChainsUnitElementary());
        dessineCarte.borderListUnit(g, units, panelOption.getIndicatorsDetail(),
Color.gray);
        // dessin partition
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsPartitionReference());
        dessineCarte.borderListUnit(g, units, panelOption.getIndicatorsDetail(),
Color.darkGray);
        // les couleurs
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsUnitElementary());
        dessineCarte.circleListUnitColor(g, units,
panelOption.getIndicatorsDetail(), tailleBoulles, couleurCercle2,
panelOption.getChainsIndicatorB());
      }
    break;
      case 3:// carte en nuances de bleu
      {
        // dessin tout gris
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite, "NUTS2", "NUTS2");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.lightGray, Color.lightGray);
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite, "NUTS0", "NUTS0");
        dessineCarte.BorderSurfaceListUnitColor(g, units,
panelOption.getIndicatorsDetail(), Color.black, Color.lightGray);
        // Fin de Dessin tout gris
        // les couleurs
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsUnitElementary());
        dessineCarte.surfaceListUnitIntervalColor(g, units,
panelOption.getIndicatorsDetail(), couleursCarteBleu);
        // dessin ut ele
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsUnitElementary());
        dessineCarte.borderListUnit(g, units, panelOption.getIndicatorsDetail(),
Color.gray);
        // dessin partition
        units = calculCarte.getListUnitsFromSpaceGrid(listUnite,
panelOption.getChainsSpaceEtude(), panelOption.getChainsPartitionReference());
        dessineCarte.borderListUnit(g, units, panelOption.getIndicatorsDetail(),
Color.darkGray);
      }
      break;
      //...
    }
    //...
  }
  //...
}
```

Annexe 4 – Code de la classe `AbstractMap` de la version 1.0 et des classes la spécialisant

Extrait de la classe `hypercarte.ui.AbstractMap` :

```
package hypercarte.ui;
//...

abstract class AbstractMap extends IndexedPanel {
    //...
    protected abstract void paintMap(Graphics graphics);
    //...
}
```

Extrait de la classe `hypercarte.ui.Map` :

```
package hypercarte.ui;
//...

abstract class Map extends AbstractMap implements IGlobalEventListener,
        IIndexedEventListener {
    //...
    protected Color backgroundColor = new Color(206, 203, 186);

    protected Color elementaryZoningBorderColor = new Color(90, 90, 90);

    public void paintComponent(Graphics graphics) {

        super.paintComponent(graphics);

        paintMap(graphics);
    }
    protected void paintMap(Graphics graphics) {

        // Initalize rendering

        this.g = setGraphics(graphics);

        // Draw background i.e. units that do not belong to any zoning

        drawMapBackground(this.g, this.topZoningBorderColor,
                this.backgroundColor);

        // Draw study area at elementary zoning level

        drawUnits(this.g, this.repository.getContextUnitList(),
                this.elementaryZoningBorderColor, this.studyAreaBackgroundColor);
    }
}
```

Extrait de la classe `hypercarte.ui.ContextMap` :

```
package hypercarte.ui;
//...

class ContextMap extends Map {

    protected void paintMap(Graphics graphics) {

        super.paintMap(graphics);

        try {

            // Draw the units of the highest zoning level that belong to the
            // study area

            drawHighestZoningBorders(super.g, super.topZoningBorderColor);

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Extrait de la classe `hypercarte.ui.DiscMap` :

```
package hypercarte.ui;
//...

abstract class DiscMap extends Map implements IIndexedEventListener {
    //...

    protected DiscFactory discSize;

    public DiscMap(int mapIndex) {

        super(mapIndex);

        this.studyAreaBackgroundColor = new Color(255,255,230);

        try {
            this.discSize = new DiscFactory();
        } catch (Exception e) {
            e.getStackTrace();
        }
    }
    protected void paintMap(Graphics graphics) {
        super.paintMap(graphics);

        try {

            // Set colored disc sizes

            this.discSize.setMax((int) super.settings.getMap(super.mapIndex)
```

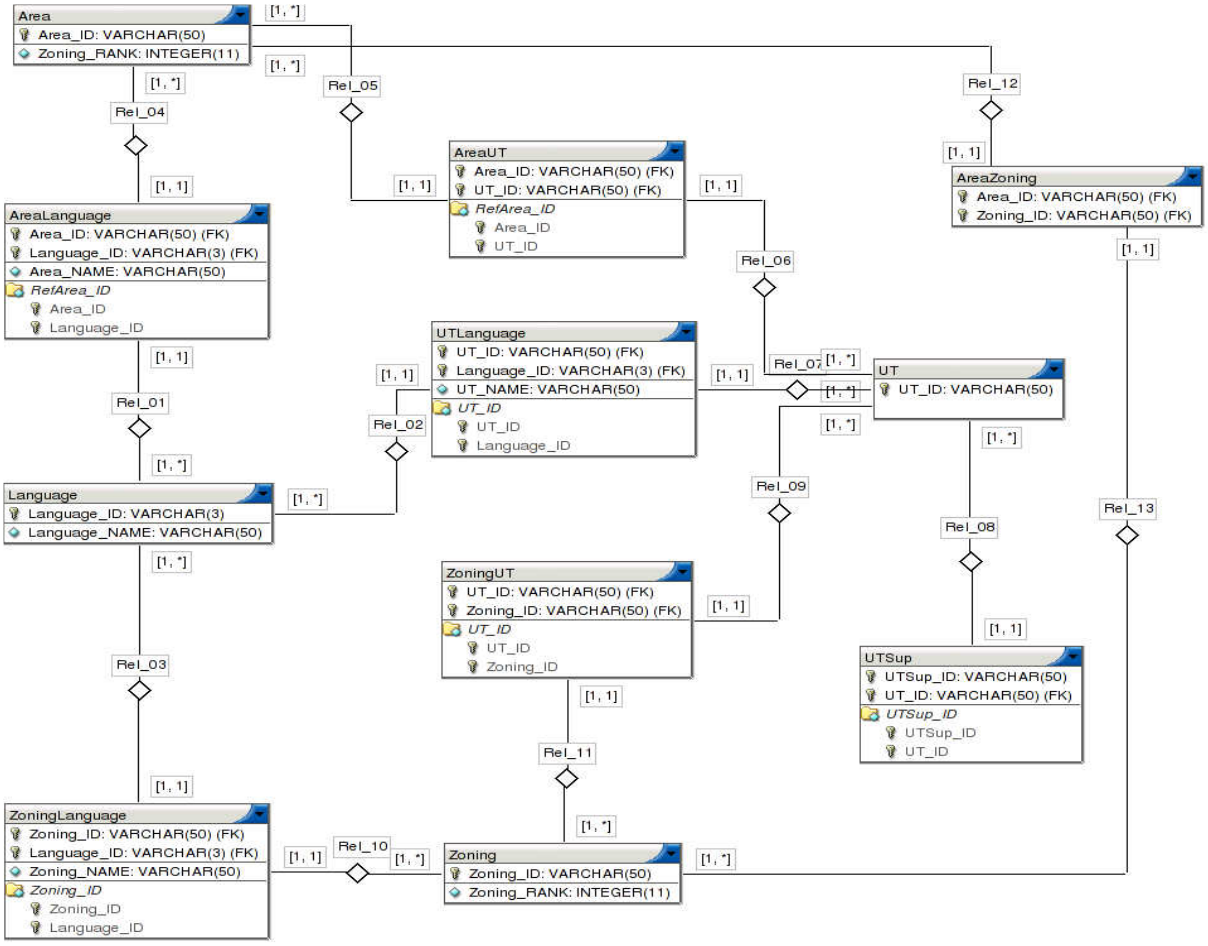
```
                    .getMax());
            this.discSize.setMin((int) super.settings.getMap(super.mapIndex)
                    .getMin());

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Extrait de la classe `hypercarte.ui.NumeratorMap` :

```
package hypercarte.ui;
//...

final class NumeratorMap extends DiscMap {

       protected void paintMap(Graphics graphics) {

              super.paintMap(graphics);

              try {

                     // Draw colored discs

                     drawUnitsDisc(super.g, super.discSize, Colors
                                   .getColors(super.settings.getMap(super.mapIndex)
                                                 .getColorHue(), 1)[0], super.settings
                                   .getNumeratorCode());

              } catch (Exception ex) {
                     ex.printStackTrace();
              }
       }
}
```

Annexe 5 – Structure de base de données

Le schéma ci-dessous, orienté base de données, est le résultat d'un groupe de travail composé des trois équipes du projet Hypercarte. Il décrit la structure de base de données qui pourrait être utilisée par le logiciel Hypercarte dans une prochaine version. L'utilisation d'une base de données est un moyen simple et efficace de contrôler la complétude des données (par exemple, en vérifiant que chaque UT possède une valeur statistique pour chaque stock).



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

Modélisation spatiale multiscalaire de phénomènes sociaux : Réalisation du logiciel Hypercarte

Olivier Cuenot **Grenoble, le XX mars 2005**

Résumé :

Comment rendre compte de phénomènes sociaux se produisant à différentes échelles (régional, national et continental) ? Pour répondre à cette question, le projet de recherche Hypercarte propose une modélisation spatiale multiscalaire qui met en évidence ces phénomènes à l'aide de cartes. Le logiciel Hypercarte implémente cette modélisation dans le but de générer ces cartes à la volée.

Partant d'un prototype du logiciel, nous avons repensé l'application. Nous l'avons réécrite, en prenant en compte les contraintes liées aux problèmes mis en évidence par le prototype comme le temps d'affichage ou l'ergonomie. {un peu lourd !}

Blah blah blah

Mots-clés : Système d'information géographique, Cartographie, Analyse territoriale multiscalaire, Interface graphique utilisateur, Ergonomie, Java, Modélisation objet, Réingénierie.

Keywords: Geographical Information System, Mapping, Multiscalar Territorial Analysis, Graphical User Interface, Ergonomics, Java, Object Modeling, Reengineering.
