CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

présenté par **Benjamin BRICHET-BILLET**

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en INFORMATIQUE

GenGHIS-Web : un portail collaboratif pour la création et le partage de Systèmes d'Information Spatio-Temporelle

Soutenu le 19 décembre 2011

JURY

Président : M. Eric Gressier-Soudan

Membres : M. Jean-Pierre Giraudin
M. André Plisson
M. Mathias Voisin-Fradin
M. Jérome Gensel
M. Philippe Genoud

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

présenté par **Benjamin BRICHET-BILLET**

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en INFORMATIQUE

GenGHIS-Web : un portail collaboratif pour la création et le partage de Systèmes d'Information Spatio-Temporelle

Soutenu le 19 décembre 2011

Les travaux relatifs à ce mémoire ont été effectués au Laboratoire d'Informatique de Grenoble sous la direction de *Philippe GENOUD*

« Tout ce qui est dans la limite du possible doit être et sera accompli. »

Jules Verne

Remerciements

Tout d'abord, je tiens à remercier les membres du jury :

- Monsieur Eric Gressier-Soudan, professeur au CNAM de Paris et président de ce jury ;
- Monsieur Jean-Pierre Giraudin, professeur à l'Université Pierre Mendès France de Grenoble ;
- Monsieur André Plisson, directeur du centre d'enseignement CNAM de Grenoble ;
- Monsieur Mathias Voisin-Fradin, directeur adjoint du CNAM de Grenoble.

J'adresse tous mes remerciements à Monsieur Jérôme Gensel, Professeur à l'Université Pierre Mendès France de Grenoble, pour m'avoir accueilli dans l'équipe qu'il dirige, STEAMER, et pour sa confiance qu'il m'a accordée tout au long de mon stage. Je remercie mon tuteur, Monsieur Philippe Genoud, maître de conférences à l'Université Joseph Fourier. Merci pour ses conseils et relectures aboutissant à l'achèvement de ce mémoire.

Je remercie également tous les membres de l'équipe STEAMER pour leur accueil et leur soutien. Je remercie tout particulièrement Anton Telechev, Benoît Le Rubrus, Laurent Poulenard et Mouna Snoussi pour leurs conseils.

Je remercie Monsieur Jean-Yves Nicolas pour les nombreuses relectures de ce mémoire.

Je remercie ma mère pour m'avoir toujours soutenu et encouragé dans mes études.

Enfin, j'adresse un grand remerciement à mon épouse pour sa présence et ses encouragements tout au long de mon cursus.

Table des matières

Table des figures

Liste des tableaux

Acronymes

CDM : Conceptual Data Model.

CSS : Cascading Style Sheets.

HQL : Hibernate Query Langage.

HTML : Hypertext Markup Language.

IDE : Integrated Development Environment.

IHM : Interface Homme Machine.

JSF : Java Server Faces.

MCD : Modèle Conceptuel de Données.

MCDI : Modèle Conceptuel de Données Instancié.

MDB : Modèle de Données Brutes.

MIF : MapInfo Interchange Format.

MP : Modèle de Présentation.

OGC : Open Geospatial Consortium.

OMG : Object Management Group.

OSI : Open Systems Interconnection.

PM : Presentation Model.

PU : Processus Unifié.

RD : Raw Data.

RMI : Remote Method Invocation.

SGBD : Système de Gestion de Bases de Données.

SIST : Systèmes d'Information Spatio-Temporelle.

SLD : Styled Layer Descriptor.

STIS : Spatio Temporal Information Systems.

SVG : Scalable Vector Graphics.

UML : Unified Modeling Language.

VDB : Valeurs des Données Brutes.

XML : Extensible Markup Language

Chapitre 1. Présentation

1.1. Introduction

Les Systèmes d'Information Géographique (SIG) sont des systèmes d'information capables d'organiser et de présenter des données alphanumériques spatialement référencées, ainsi que de produire des plans et des cartes. Leurs usages couvrent les activités géomatiques, c'est-à-dire l'ensemble des outils et méthodes permettant de représenter, d'analyser et d'intégrer des données géographiques.

Maguire *et al.* (1991) distinguent trois périodes principales dans l'évolution des SIG :

- *fin des années 1950 – milieu des années 1970* : début de l'informatique, premières cartographies automatiques ;
- *milieu des années 1970 - début des années 1980* : diffusion des outils de cartographie automatique/SIG dans les organismes d'État (armée, cadastre, services topographiques, ...) ;
- *depuis les années 1980* : croissance du marché des logiciels, développements des applications sur PC, mise en réseau (base de données distribuées).

L'avènement d'internet et du World Wide Web, en offrant au plus grand nombre un moyen d'acquérir et de diffuser de l'information géographique, ont entrainé l'explosion des SIG. L'apparition d'outils cartographiques directement accessibles sur le web (par exemple GoogleMaps), d'outils embarqués liés au GPS, mais aussi de logiciels libres ou d'outils dédiés aux pratiques coopératives ont eu pour conséquence de banaliser l'usage de l'information géographique.

De nos jours, l'usage croissant de données géoréférencées nous contraint à la traduction de ces données sous forme de représentations spatiales c'est-à-dire des cartes. Ces cartes ont des usages très variés tels que la localisation, la communication et l'analyse afin de comprendre les phénomènes spatiaux et leurs évolutions. De surcroît, les données deviennent

multidimensionnelles : spatiales, attributaires, temporelles, multimédias. Elles sont également hétérogènes, incomplètes et évolutives, ce qui augmente leur complexité. On constate aujourd'hui que cette évolution amène le besoin de nouvelles fonctionnalités : la visualisation de façon orthogonale de toutes les dimensions de l'information, l'adaptation de la présentation des données en fonction des thématiques et des objectifs, enfin la possibilité de procéder à des analyses préalables d'explorations des données.

1.2. Contexte

1.2.1. STeamer

L'équipe STeamer (*Spatio-TEmporal information, Adaptability, Multimedia and knowlEdge Representation*), au sein de laquelle j'ai effectué le stage qui a conduit à la rédaction du présent mémoire, est une équipe de recherche du Laboratoire d'Informatique de Grenoble (LIG). Ses travaux de recherche portent sur la représentation, le traitement et la diffusion d'informations spatio-temporelles. L'objectif est de proposer des formalismes, des méthodes et des outils permettant de progresser dans la conception, la mise en œuvre et l'utilisation de Systèmes d'Information Spatio-Temporelle. Il s'agit de concevoir et construire des solutions novatrices pour des applications qui requièrent la manipulation de données à références spatiales et temporelles, données qui peuvent être imprécises ou incomplètes. L'accent est mis sur la représentation sémantique des données et des métadonnées, à l'aide de différents formalismes (ontologiques, objets, à base de règles, etc.), ainsi que sur l'optimisation des méthodes d'accès et des traitements associés. Les contributions visent également la restitution de l'information spatio-temporelle à l'aide de composants cartographiques dynamiques adaptés. Les travaux de recherche de STeamer s'inscrivent systématiquement dans une démarche interdisciplinaire et opérationnelle : d'une part, ils s'appuient sur des domaines thématiques liés aux sciences de la terre, à l'environnement, et à l'analyse spatiale, d'autre part, ils visent le développement de plates-formes et de cadriciels opérationnels. L'application des travaux de l'équipe STeamer se fait dans le domaine de la géomatique et celui des risques naturels(1).

Pour résumer, les recherches de l'équipe STeamer sont organisées selon trois axes :

- la représentation et la modélisation d'informations spatio-temporelles ;
- la Géo-visualisation et la présentation adaptée d'informations multimédias et spatio-temporelles ;
- l'adaptabilité et la mobilité des Systèmes d'information.

1.2.2. GenGHIS

Parmi les logiciels développés par STeamer, se trouve GenGHIS (Generator for Geographic and Historical Information Systems) : un générateur de Systèmes d'Information

Spatio-Temporelle (SIST). Les travaux sur GenGHIS s'inscrivent dans les deux premiers axes de recherche de STeamer, c'est-à-dire dans le domaine de la géo-visualisation et la présentation adaptée d'informations spatio-temporelles ainsi que dans leur représentation et leur modélisation. En effet, le logiciel GenGHIS développé depuis 2004, essaye de répondre, en proposant des solutions novatrices, au contexte d'évolution des technologies informatiques et des SIG que nous avons évoqué en introduction. En s'appuyant sur une spécification déclarative d'un SIST et une modélisation orientée objet en deux parties distinctes de son contenu (les données) et de sa forme (présentation), GenGHIS propose une interface graphique permettant à des utilisateurs (scientifiques, géographes, historiens, etc.) de construire des SIST adaptés à leurs besoins et leurs objectifs, SIST pouvant être accédés directement depuis un navigateur web (cf. Figure 1).

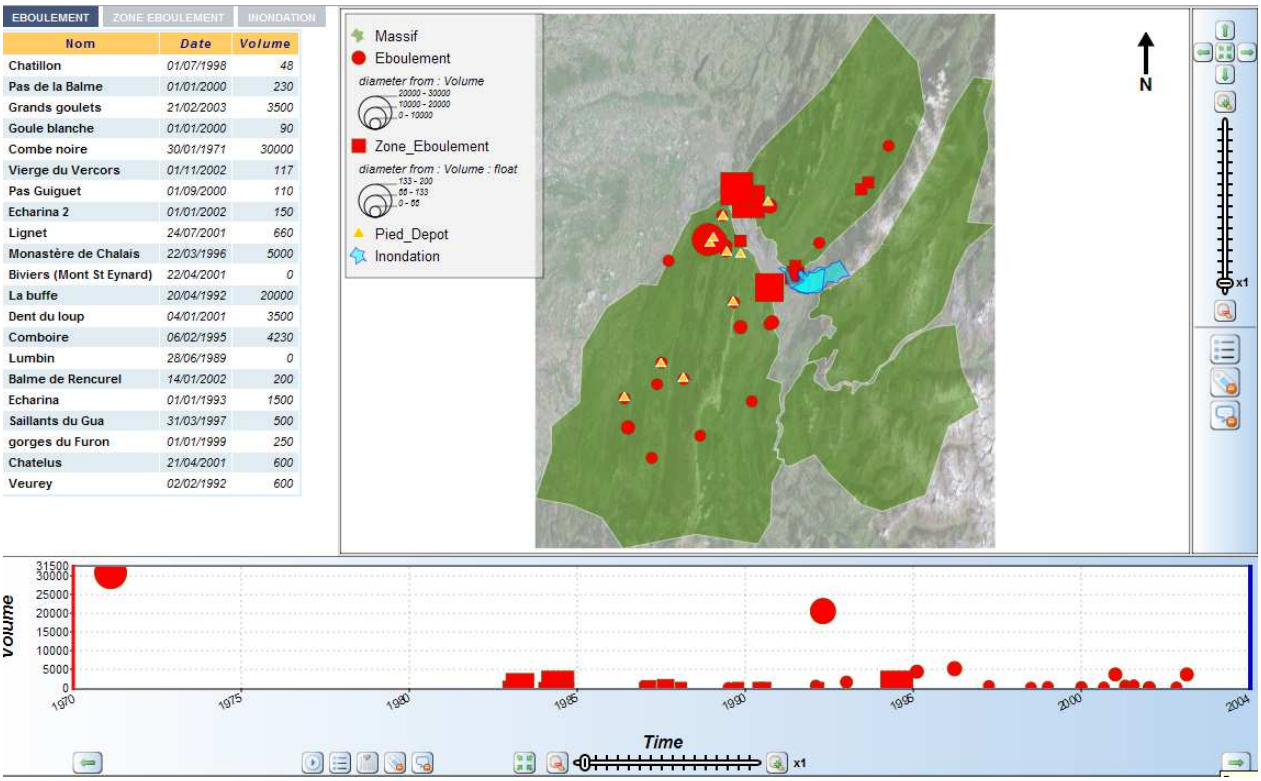


Figure 1 : Exemple d'application SIST générée par GenGHIS

Les travaux sur GenGHIS ont débuté en 2004, lorsque les membres de l'équipe STeamer[1] éprouvèrent le besoin de capitaliser leur savoir-faire en matière de conception de systèmes d'information pour les risques naturels. Initialement, les concepts mis en œuvre dans GenGHIS furent développé par Bogdan MOISUC(2) durant son doctorat d'informatique. Ensuite, ce logiciel fut amélioré par un certain nombre de stagiaires dont Laurent GAYET(3) qui, dans le cadre d'un stage en vue de présenter son mémoire d'ingénieur CNAM, a développé la version sur laquelle a porté notre travail.

[1] A l'époque, le LIG et l'équipe STeamer n'existaient pas encore. Les membres de la future équipe STeamer (fondée en 2007, en même temps que le LIG) étaientt alors rattachés à l'équipe SIGMA (Systèmes d'Information, inGénierie et MultimediA) du laboratoire LSR (Logiciels Systèmes Réseaux).

1.3. Problématique

Depuis le développement de GenGHIS, l'équipe STeamer effectue de nombreuses démonstrations de celle-ci, ce qui permit d'établir un constat sur ses limites :

L'application GenGHIS, dans sa dernière version est une application autonome monoposte, qui, dans le contexte actuel de développement des Systèmes d'Information Géographique sur le web est un frein à sa diffusion. En effet, l'utilisateur doit installer l'application GenGHIS sur son poste de travail où il construit et génère localement son Systèmes d'Information Spatio-Temporelle (SIST). Ensuite, s'il souhaite partager son résultat, il doit transmettre les différents fichiers générés composant le SIST et/ou les installer sur un serveur Web. En pratique, ceci est loin d'être aisé, en particulier pour des utilisateurs finaux non informaticiens issus de domaines divers et variés. Avec le développement des applications accessibles directement via internet et le web, la notion de portail collaboratif s'est ancrée. L'intégration de GenGHIS dans une plate-forme interactive couvrant à la fois les besoins de création et de diffusion permettrait d'avoir une porte d'entrée unique sur un service de SIST, partagé par une communauté d'utilisateurs.

1.4. Objectifs

L'objectif du travail qui m'a été confié et qui est présenté dans le présent mémoire est donc de répondre à la problématique précédemment énoncée : développer un portail web collaboratif permettant à des utilisateurs de construire, modifier et partager des SIST GenGHIS.

Pour réaliser cet objectif, nous avons non seulement dû entièrement concevoir puis implémenter l'application web gérant ce portail (application que nous appellerons dans la suite de ce document GenGHIS-Web), mais également faire un important travail de rétro-ingénierie de l'application produite par Laurent Gayet, à la fin de son stage d'ingénieur CNAM, afin de l'intégrer à GenGHIS-Web. En effet, GenGHIS-Web doit permettre de mettre en avant les fonctionnalités de GenGHIS. L'utilisateur devra pouvoir non seulement accéder à l'interface graphique des SIST générés, mais également construire, modifier, partager de nouvelles applications. L'application GenGHIS n'ayant plus pour vocation d'être monoposte, il nous a fallu comprendre son fonctionnement pour être ensuite en mesure de l'utiliser correctement, de la modifier et de la transformer en une application modulaire fonctionnant en mode client-serveur[2].

Ce sont ces différents travaux, menés en vue d'atteindre cet objectif, que nous présentons ici.

[2] L'architecture client/serveur désigne un mode de communication entre plusieurs ordinateurs d'un réseau qui distingue un ou plusieurs clients du serveur.

1.5. Plan du mémoire

Après ce chapitre introductif, la suite de ce mémoire est constituée de cinq chapitres.

Le second chapitre dresse un état de l'art sur les solutions existantes en termes de création de cartes interactives, à partir d'un navigateur internet.

Le chapitre trois est consacré à l'analyse de l'existant. A la fin de ce chapitre, nous dressons une synthèse du fonctionnement de l'application.

Le chapitre quatre présente notre proposition. Une première partie est consacrée aux processus de développement. Ensuite, nous définissons les besoins initiaux et nous poursuivons sur un échange concernant les spécifications des exigences. Enfin, nous abordons la navigation et l'ergonomie du site.

Le chapitre cinq expose le travail de développement que nous avons effectué durant le stage, à partir des choix de conception et d'architecture proposés au chapitre quatre ainsi que les outils et les méthodes utilisés.

Le dernier chapitre conclut ce mémoire par une synthèse des réalisations présentées dans le chapitre cinq, il aborde aussi les perspectives d'évolution de l'application. Enfin, il expose notre bilan du stage.

Chapitre 2. Etat de l'art

Notre recherche en matière de création de cartes en ligne nous a permis de constater que peu de solutions existaient. Toutefois, nous en avons retenu deux, permettant la création de cartes à travers un portail internet.
La première partie de ce chapitre présente Carmen, qui est un outil développé à l'initiative du Ministère de l'Écologie, du Développement durable, des Transports et du Logement (MEDDTL).
La seconde partie présente le travail d'un doctorant de l'université de Pau : Windmash.

2.1. Carmen

La diffusion des données environnementales cartographiques est un enjeu essentiel pour l'information tant des citoyens que des élus et pour leur association et adhésion aux différentes politiques environnementales. Carmen (**Car**tes du **M**inistère en charge de l'**En**vironnement) est une des applications du Ministère de l'Écologie, du Développement durable, des Transports et du Logement (MEDDTL), permettant d'accéder aisément en ligne aux données géographiques environnementales publiques, ainsi qu'à leur visualisation cartographique.

Ce projet s'inscrit également dans le cadre de la directive européenne Inspire. Cette dernière impose aux Etats de l'Union Européenne et à leurs autorités publiques, de bâtir une infrastructure maillée d'échanges pour l'information géographique environnementale publique. Cette infrastructure se base sur des dispositifs de catalogues, et des protocoles d'interopérabilité, permettant l'échange et la réutilisation des données entre différentes sources d'informations éclatées.

Ainsi, Carmen permet de publier les données géographiques environnementales, superposées à des fonds référentiels de l'IGN[3] (BD ORTHO, SCANS, BD Parcellaire…)(4).

L'application Carmen est dotée d'un site d'accompagnement permettant l'accès à un certain nombre de cartes, de se documenter et de trouver une assistance. Le support assuré par le Cemagref s'effectue par courrier électronique et sur le site : http://carmen.ecologie.gouv.fr. Une offre de formation est aussi disponible.

Carmen, c'est aussi un service d'hébergement, assuré par le BRGM[4]. Les administrateurs Carmen restent maîtres de leur propre site.

Enfin Carmen, c'est un référentiel géographique riche, alimenté grâce à un protocole signé entre l'IGN et le ministère.

2.1.1. Le module de consultation

Le module de consultation permet à tout internaute de visualiser les productions cartographiques des administrateurs Carmen, de télécharger des données sources SIG en vue d'une utilisation future, de se déconnecter de l'application en ligne, d'ajouter des objets graphiques grâce aux flux WMS[5]/WFS[6], d'imprimer et d'exporter des cartes.

Dans un premier temps, nous décrivons le fonctionnement global du module de consultation puis, plus précisément les différentes parties le composant.

2.1.1.1. Initialisation du module de consultation

Le lancement du module de consultation s'effectue par l'appel d'une URL[7] au moyen d'un navigateur internet. Cette URL passe en paramètre un numéro de service et un fichier mapfile[8] décrivant la carte et l'interface de consultation associés. L'URL ci-dessous permet d'accéder à la carte utilisée comme exemple.

```
http://carmen.application.developpement-
durable.gouv.fr/30/NATURE_PAYSAGE_BIODIVERSITE_RA.map
```

Lors du chargement de la page de consultation cartographique, sur le poste client, un service prend en paramètre le nom du mapfile de la carte et renvoie au poste client les différentes informations sous la forme d'un flux XML.

[3] Institut Géographique National
[4] Le BRGM, anciennement Bureau de recherches géologiques et minières, est l'organisme public français référent dans le domaine des sciences de la Terre pour la gestion des ressources et des risques du sol et du sous-sol.
[5] Web Map Service permet de produire des cartes de données géoréférencées à partir de différents serveurs de données.
[6] Web Feature Service est un protocole décrit dans des spécifications maintenues par l'Open Geospatial Consortium. Le service WFS permet, au moyen d'une URL formatée, d'interroger des serveurs cartographiques afin de manipuler des objets géographiques (lignes, points polygones...).
[7] Une URL (*Uniform Resource Locator*) est un format de nommage universel pour désigner une ressource sur Internet.
[8] Un fichier mapfile est un fichier de configuration utilisé par mapserver. MapServer est un environnement de développement libre permettant de construire des applications internet à référence spatiale.

Ces informations comprennent (5) :

- les informations générales sur la carte (par exemple : nom, projection) ;
- la description des différentes couches du mapfile (par exemple : nom, adresse de la couche, projection, metadata du layer, urls des images de la légende) ;
- les informations de structuration des couches en groupes techniques (information stockée dans les métadonnées du mapfile) ;
- un lien vers l'image de la vue générale ;
- la liste des fonctionnalités actives (paramétrage de l'interface) ;
- les paramètres d'initialisation du requêteur de localisation (par exemple : nombre de niveaux du requêteur, couche et champ associés à chaque niveau).

Le fichier de configuration généré à partir du mapfile lors de la requête du client est standardisé, Carmen ayant pris le parti de s'inspirer de la proposition OWS Context, pour la description des contextes de carte. Le format OWS Context est annoncé comme une extension du format WMC qui permet d'associer un contexte cartographique aux services WMS. L'OWS Context doit prendre en compte les autres services de l'Open Geospatial Consortium[9] (ex. : WFS, WCS).

La figure suivante (cf. Figure 2) est un exemple de module de consultation, dont le thème est : « Nature, Paysage et Biodiversité en Rhône-Alpes ».

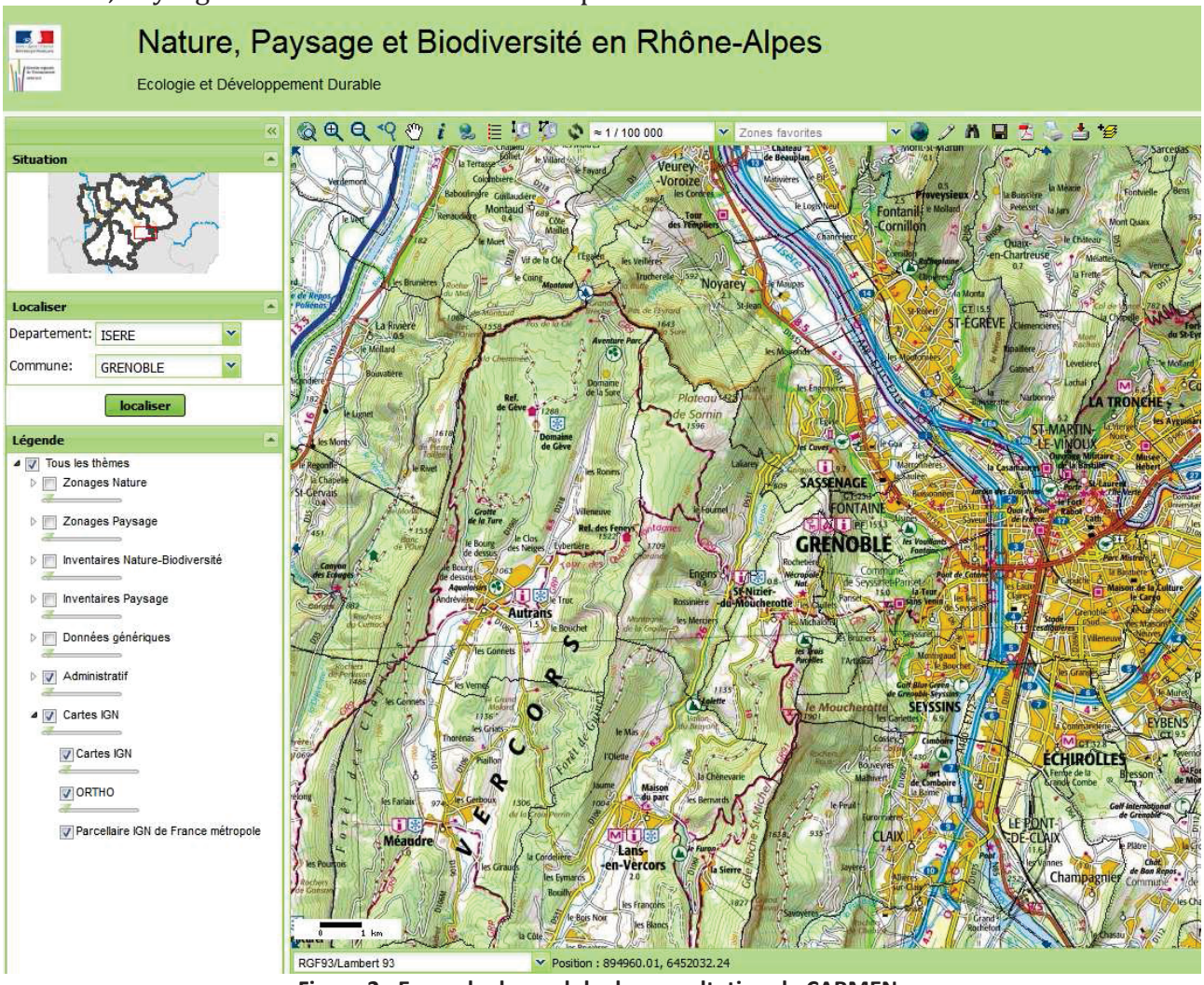


Figure 2 : Exemple de module de consultation de CARMEN

[9] L'OGC est un consortium international pour développer et promouvoir des standards ouverts.

2.1.1.2. La vue cartographique

Les vues cartographiques générées par Carmen utilisent la bibliothèque de fonction Javascript[10] : OpenLayers(5). OpenLayers permet d'afficher des fonds cartographiques tuilés, ainsi que des marqueurs provenant d'une grande variété de sources de données. Le principe de cette bibliothèque est d'associer un élément à chaque couche. Le rendu cartographique est construit par superposition des différents éléments .

2.1.1.3. La barre de navigation

La barre de navigation d'une vue cartographique est composée de trois modules. La figure suivante (cf. Figure 3) est une capture d'écran localisant ces modules.

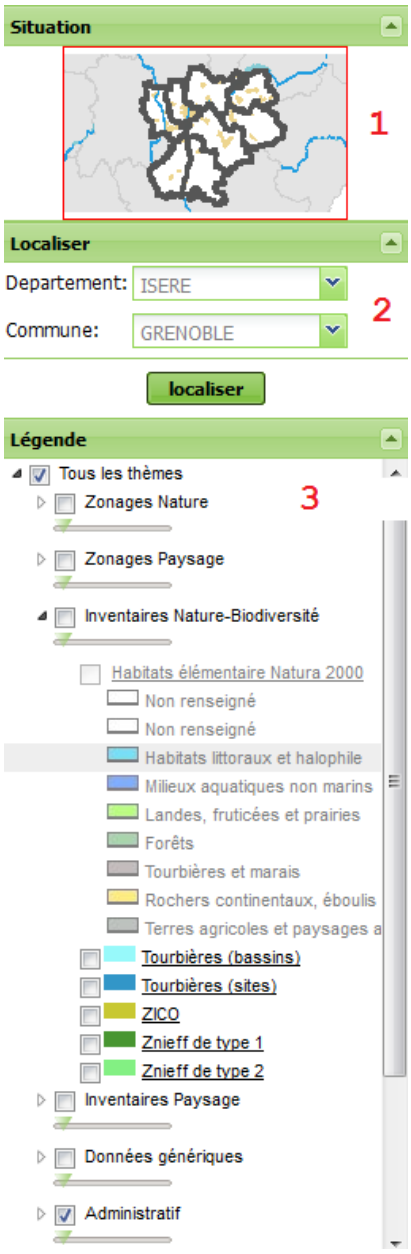


Figure 3 : Barre de navigation

[10] JavaScript est un langage de programmation de scripts.

Situation (1) : Ce module affiche une représentation miniature de l'intégralité de la carte et permet de donner une vue d'ensemble. Un cadre rouge situe la vue cartographique.

Localiser (2) : Dans le but de remplir différentes listes déroulantes (par exemple : région, département, communes), une requête de localisation fait appel à un service web sur le serveur. Nous communiquons au serveur une zone puis il renvoie une liste d'emprises (nom et emprise spatiale) correspondant soit à un ensemble des zones (départements), soit à un ensemble de sous-zones (communes d'un département).

Légende (3) : L'interface graphique de la gestion de légende (5) s'appuie sur une classe Javascript spécifique enrichissant les classes `Tree` de la librairie ExtJS[11].
Les actions sur les couches (activation, ouverture de groupe, etc) sont gérées au niveau du client, en utilisant des fonctionnalités d'OpenLayer, sans nécessiter d'appel au serveur. En effet, chaque nœud, représentant une couche ou un groupe de couches, est lié à la couche OpenLayers dans laquelle il est représenté.

2.1.1.4. Système de coordonnées spatiales

Dans la barre d'information (cf. Figure 4), l'utilisateur a la possibilité de modifier la projection dans laquelle sont affichées les cordonnées courantes du pointeur de la souris. La conversion de la position courante du système de coordonnées de la carte, (fournie par OpenLayers), vers le système de coordonnées sélectionné par l'utilisateur, est réalisée localement sur le poste client par une classe Javascript.

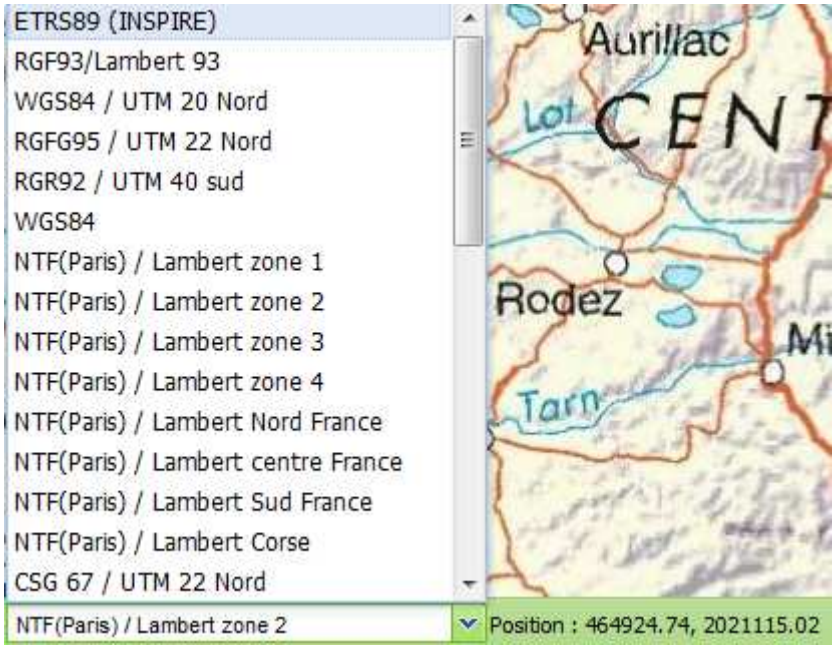


Figure 4 : Système de coordonnées spatiales

[11] La librairie javascript ExtJS propose de nombreux composants graphiques de haut niveau, en particulier elle dispose de classes `Tree`, permettant la réorganisation des nœuds de l'arbre.

Les différents noms de projection et paramètres de conversion associés, sont chargés lors du chargement du module de consultation cartographique.
Un système de coordonnées pivot (latitude/longitude) est utilisé pour les conversions. Par exemple, si le système de coordonnées de la carte est Lambert2 étendu et que les coordonnées doivent être affichées en Lambert 93, la conversion se fera en 2 temps :

```
Lambert2 étendu → pivot (lat/long) → Lambert 93
```

Ainsi, seuls les paramètres de conversion et de conversion inverse d'un système de coordonnées vers le système pivot ont besoin d'être connus au niveau du poste client.

2.1.1.5. Barre d'outils

Les outils de la barre d'outils (cf. Figure 5) soit s'appuient sur les outils cartographiques disponibles dans OpenLayers, soit font appel à des services spécifiques développés pour l'application.

Figure 5 : Barre d'outils Carmen (5)

La barre d'outils permet :

- le zoom, le déplacement, les mesures : les outils de navigation cartographiques de zoom (vue générale, zoom avant, zoom arrière, zoom précédent), de déplacement et de mesures (périmètre et superficie) sont repris sans changement d'OpenLayers ;
- les annotations : elles s'appuient sur OpenLayers, ce qui permet la saisie d'annotations géométriques ponctuelles, linéaires et surfaciques sur la carte, ainsi que leurs modifications et suppressions ;
- l'exportation, l'impression : il est possible d'imprimer la carte depuis le navigateur et de l'exporter au format image ou au format pdf[12]. La légende sera identique à celle affichée sur le poste client ;
- l'information sur les couches : à l'aide de la souris, il est possible de récupérer des informations sur les couches graphiques. La position (interrogation par un point), ou les limites de l'étendue (tracé d'un rectangle sur la carte) de l'interrogation, sont récupérées en utilisant les fonctions d'OpenLayers. Une requête WMS est envoyée au serveur. Les informations sont récupérées par le client, puis affichées dans une info bulle ;
- l'affichage de la légende : L'affichage de la légende de la carte est réalisé par un module Javascript spécifique, chargé sur le poste client. Ce module reproduit au format HTML[13] (*Hypertext Markup Language*), une version simplifiée de l'arbre de couches du cadre de légende (seules les couches actives sont représentées) ;

[12] Le ***Portable Document Format*** est un langage de description de pages d'impression créé par Adobe Systems.
[13] HTML est le format de données conçu pour représenter les pages internet.

- l'enregistrement de zones préférentielles : il est possible d'enregistrer l'emprise courante de la carte et de nommer cet enregistrement ;
- la sauvegarde et le chargement de contexte : l'enregistrement de contexte consiste à sauvegarder au format XML les caractéristiques de la carte (par exemple : l'étendue de la carte, les caractéristiques des couches, la structure de l'arbre de couches, les zones préférentielles, les annotations) ;
- le téléchargement de données : il est possible de télécharger des données sous format de base de données (table) ou des fichiers de couches (MIF/MID,…).
- l'ajout de couches WMS : un outil permet d'ajouter des couches WMS externes à la carte initiale à partir du catalogue WMS de Carmen.

2.1.2. Le module d'administration

L'un des points forts de Carmen est son module d'administration, accessible aux détenteurs d'un compte Carmen. Il permet à un utilisateur administrateur de gérer ses propres données SIG et de réaliser des cartes à partir de celles-ci et des référentiels IGN.

La figure suivante (cf. Figure 6) représente l'interface d'administration dans son ensemble. Les menus de la partie gauche permettent d'accéder aux différentes configurations que nous énumérons dans la suite du document.

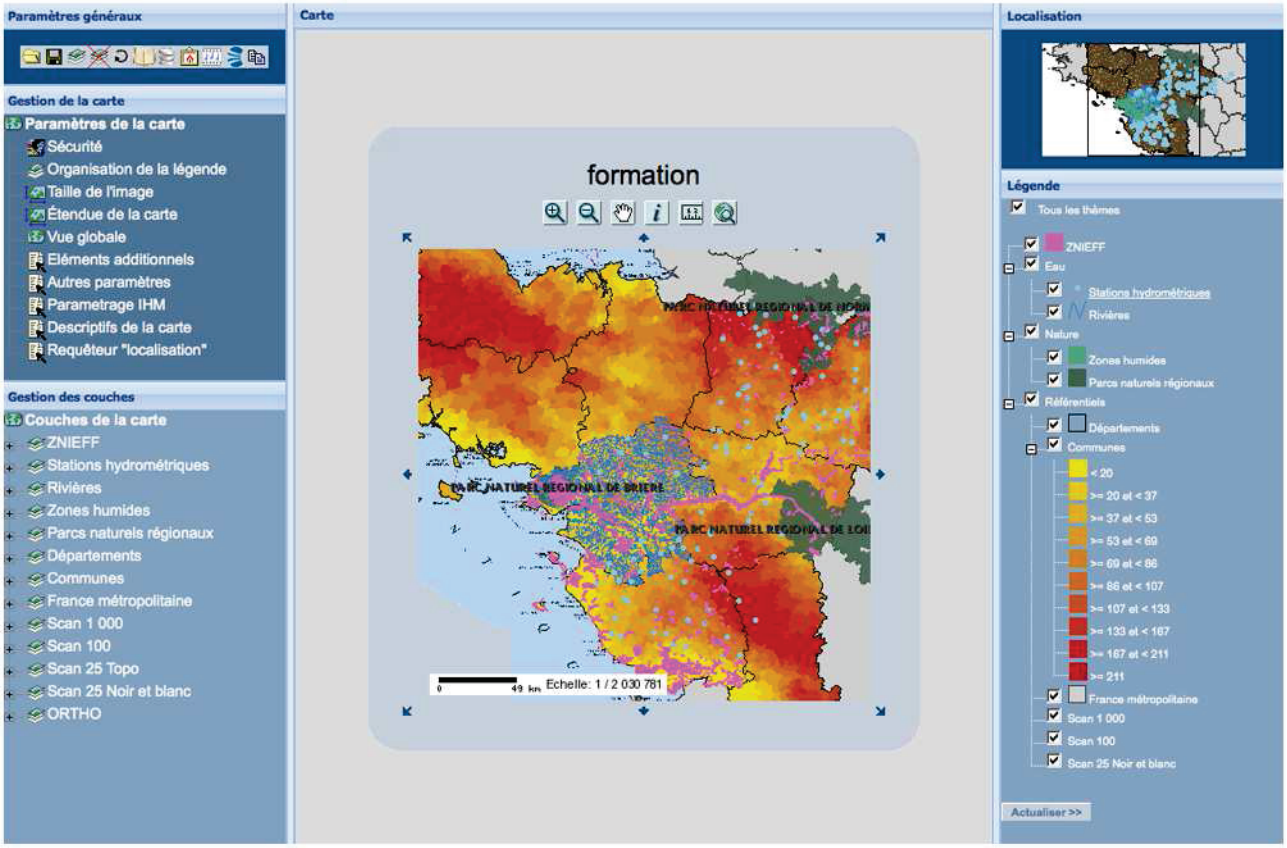


Figure 6 : Module d'administration de Carmen

Lors de la conception d'une carte, Carmen offre la possibilité d'insérer des couches SIG de sources variées et nombreuses : MapInfo, ArcView, PostGis, ECW, TIFF, WMS, …

Comme le montre la figure suivante (cf. Figure 7), l'intégration d'une couche suppose de renseigner un certain nombre d'informations telles que le champ qui la compose, un alias, un type ainsi qu'une adresse internet (URL).

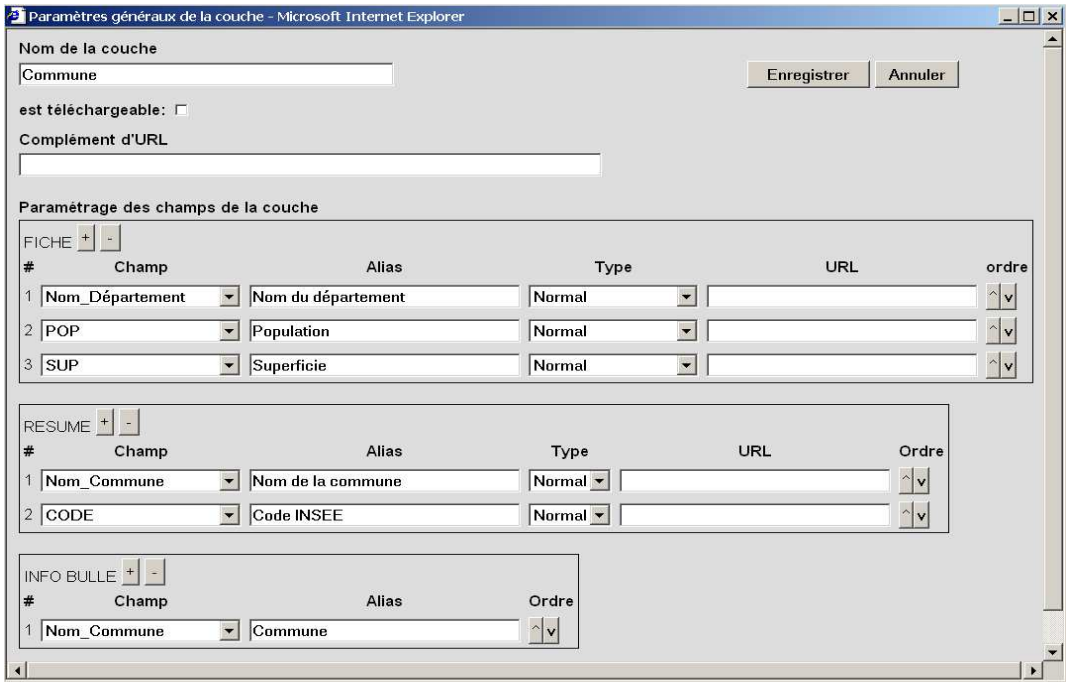


Figure 7 : Panneau d'intégration d'une couche (6)

Le concepteur a aussi en charge la gestion de la sémiologie cartographique. L'interface ci-dessous (cf. Figure 8) permet pour un type d'objet donné, de gérer le code couleur en fonction des valeurs d'un attribut (par exemple, ici, la surface).

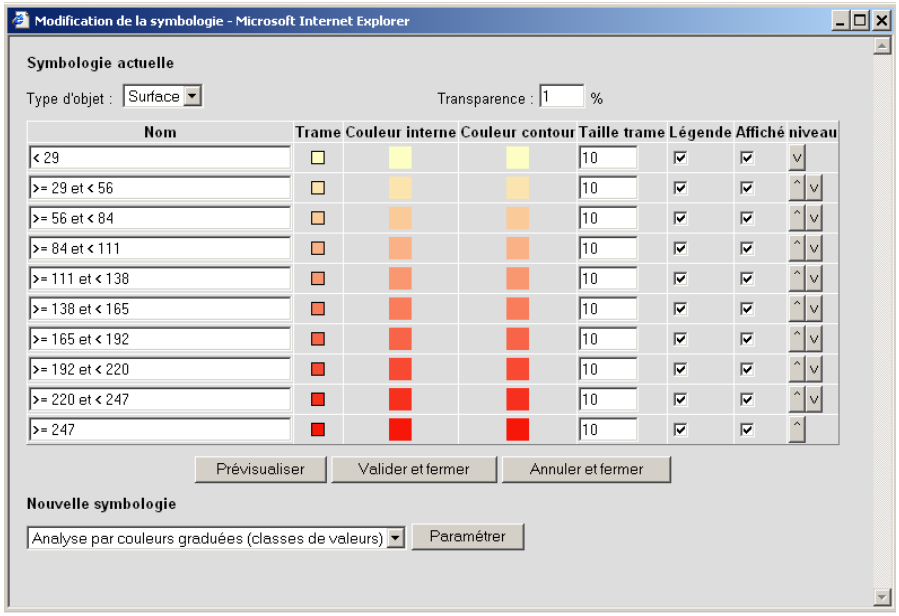


Figure 8 : Interface de modification de la symbologie(6)

Le concepteur a également en charge l'organisation de la légende (cf. Figure 9).

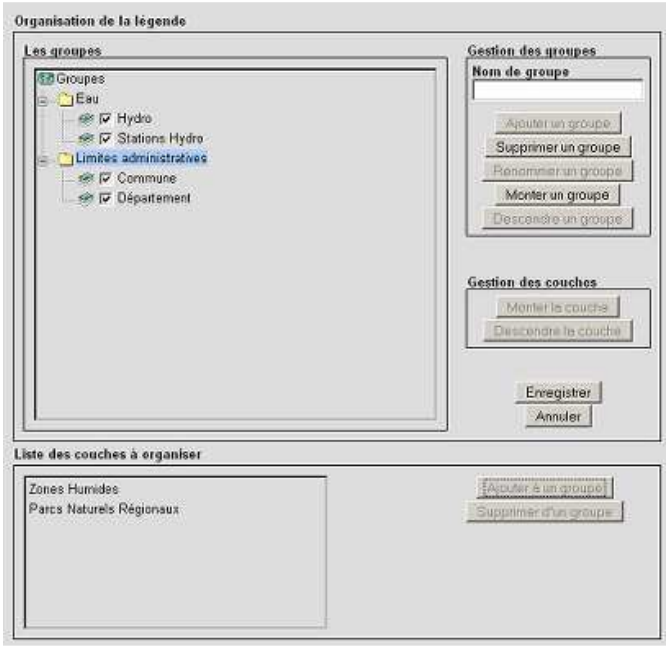


Figure 9 : Organisation de la légende

Pour un groupe donné (par exemple les limites administratives), l'utilisateur sélectionne les couches à insérer dans la légende (par exemple les communes et département).

Enfin, le concepteur définit l'interface Homme machine de l'outil de consultation ; pour cela il accède à l'interface de la figure suivante (cf. Figure 10) où il sélectionne / désélectionne ce qu'il souhaite mettre ou non à disposition sur l'interface de consultation.

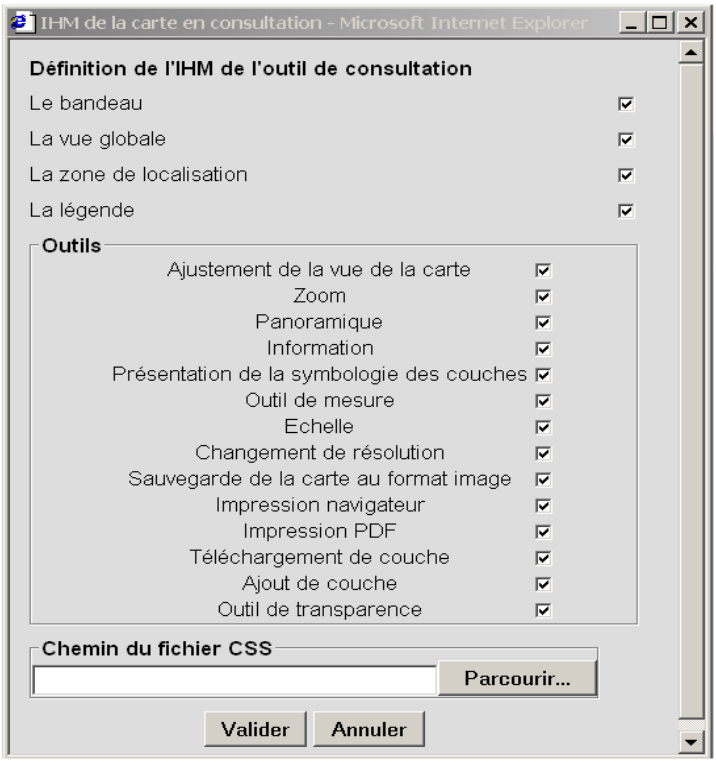


Figure 10 : Définition de l'IHM de l'outil de consultation (6)

2.2. WINDMash

WINDMash, développée par l'équipe T2I[14] de l'Université de Pau, est une application web, permettant de concevoir et de générer automatiquement une application interactive manipulant des informations géographiques. L'application WINDMash a été créée afin de permettre à des utilisateurs n'ayant aucune notion d'informatique de construire des applications web interactives géographiques à partir de données textuelles.

Le processus de conception proposé par WINDMash est composé de trois étapes (cf. Figure 11) :

- définition du contenu: le concepteur définit une chaîne de traitements, qui, à partir de données issues de texte brut, permet d'identifier automatiquement les entités géographiques qui l'intéressent ;
- interface : le concepteur associe des blocs d'affichages aux entités géographiques ;
- description de l'interaction : le concepteur créé des comportements pour l'application finale.

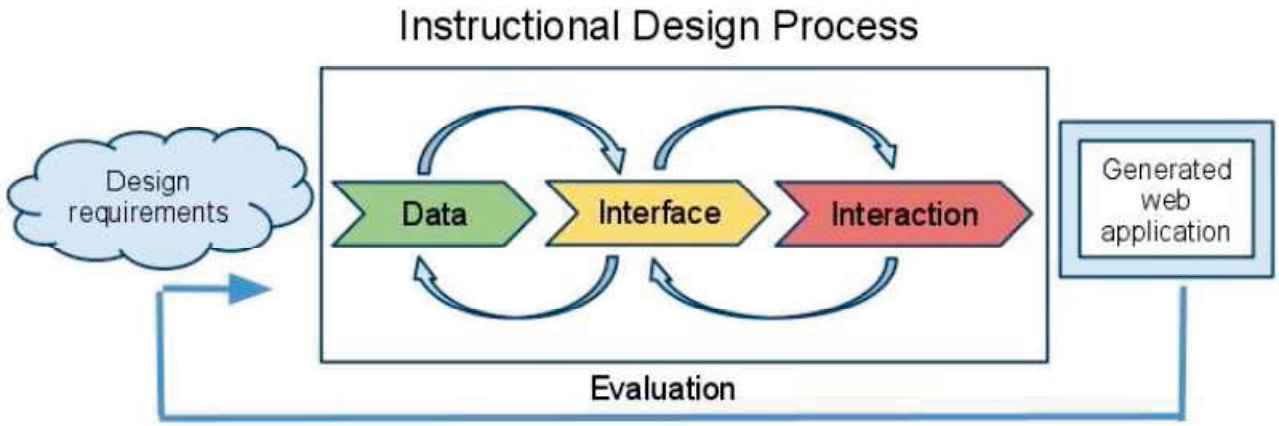


Figure 11 : Processus de création d'une application avec WINDMash (7)

Chaque étape est décrite plus précisément dans les sections suivantes.

2.2.1. Le module de contenu

Le module de contenu permet de renseigner et d'extraire les données fournies par le concepteur. A partir d'un ou de plusieurs textes bruts, le concepteur construit graphiquement une chaîne de traitements qui identifie automatiquement des entités géographiques. La finalité de ce module est de normaliser des textes au format WIND, c'est-à-dire des triplets RDF[15].

[14] T2I est une équipe dont les recherches concernent le traitement des informations spatiales, temporelles et thématiques pour l'adaptation de l'interaction au contexte et à l'utilisateur.

[15] Resource Description Framework (RDF) est un modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées, de façon à permettre le traitement automatique de telles descriptions.

La figure suivante (cf. Figure 12) est une capture d'écran du module de contenu et illustre son fonctionnement.

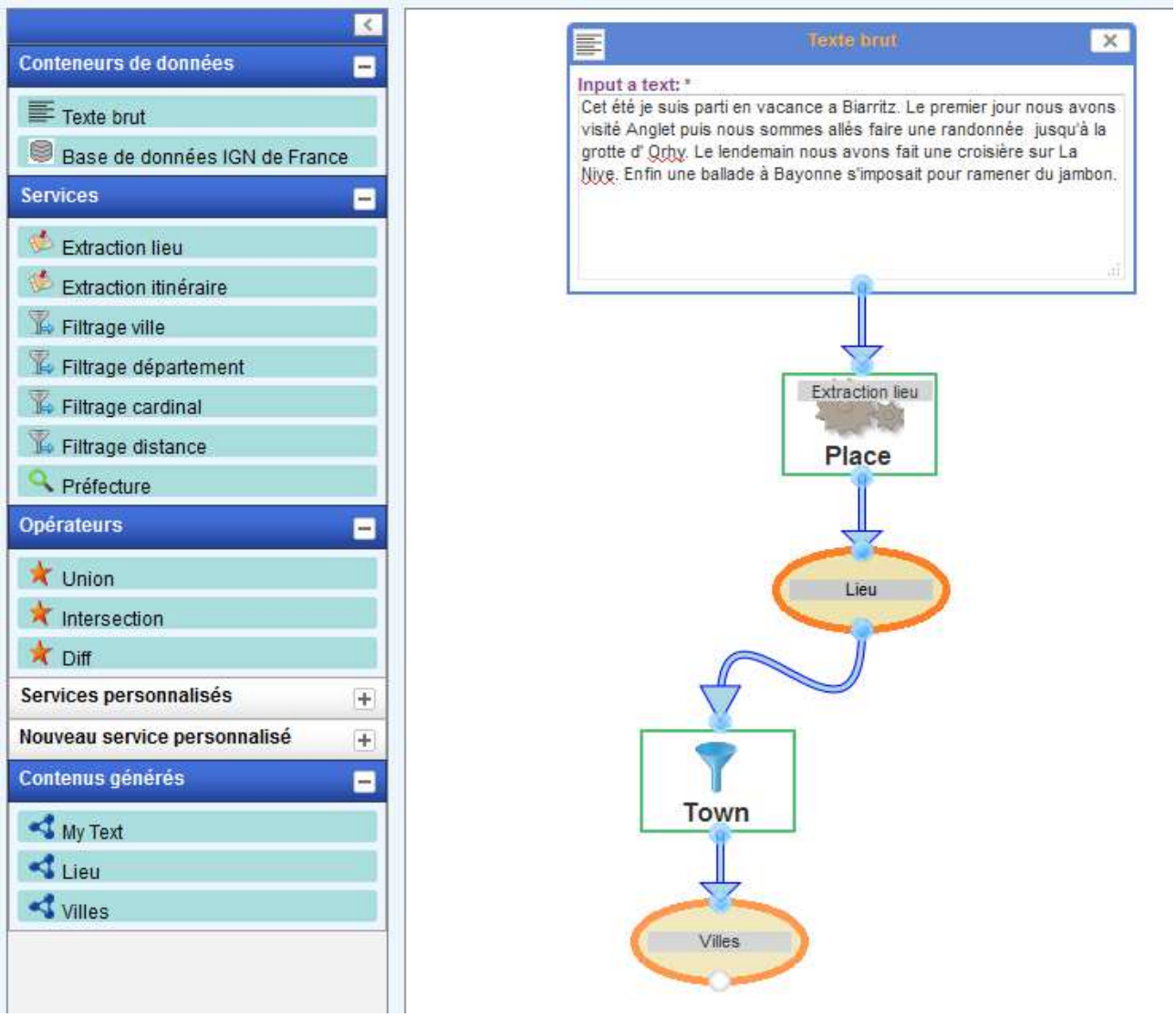


Figure 12 : Capture d'écran du module de contenu

La colonne de gauche contient tous les modules disponibles pour la construction de la chaîne de traitements pour l'extraction de contenu à partir de données. La partie droite présente la chaîne de traitements sous la forme d'un flot de données. Les boîtes représentent un module de traitement et les ovales les données produites par un traitement. La construction de la chaîne de traitements se fait par « glissé-déposé » (*drag-and-drop*) des modules depuis la colonne de gauche vers la partie droite de la fenêtre.

Dans la colonne de gauche, les modules disponibles, regroupés par famille, sont :

Les conteneurs de données : ils définissent la source des données qui peut être soit du texte brut, soit une base de données IGN. Dans notre exemple, la source de données est un texte brut : il s'agit d'un récit de voyage décrivant des vacances localisées géographiquement dans le département des Pyrénées-Atlantiques (64).

Les services :

- L'extraction de lieux et d'itinéraires sont les modules de transformations des données brutes en données normalisées (au format XML). Il est nécessaire de configurer ces modules, en sélectionnant parmi les références mises à disposition par le logiciel, celles que l'on veut retenir. Dans notre exemple, nous avons procédé à l'extraction des lieux avec les références :
  - Communes de la France ;
  - Oronymes 64[16] ;
  - Cours d'eau 64.
- Les services de filtrage (ville, département, cardinal, distance) permettent de filtrer les données normalisées issues de l'extraction. Dans notre exemple nous avons appliqué un filtre ville sur les données normalisées issues du module d'extraction de lieux ;
- Le service de préfecture distingue les préfectures parmi les villes contenues dans des données normalisées.

Les opérateurs : il est possible d'effectuer des opérations d'union, d'intersection et de différence sur les données normalisées.

Après avoir assemblé sa chaîne de traitements, le concepteur procède au lancement du traitement. L'application extrait les données et génère les contenus correspondants en fonction des services. Dans notre exemple, trois contenus ont été générés (en bas à droite de la colonne de gauche): *My Text*, Lieux et Villes.

Ces contenus sont ensuite utilisés par le module d'interface.

2.2.2. Le module d'interface

Le module d'interface permet au concepteur de construire et d'organiser l'interface de l'application qui sera générée par la suite. La figure ci-dessous montre l'interface de ce module.

[16] Un oronyme, est un néologisme grec signifiant « nom de montagne ».

Figure 13 : Capture d'écran du module d'interface

Le concepteur choisit un certain nombre d'afficheurs (colonne de gauche) qu'il dispose dans la partie droite de l'écran qui sert à assembler l'interface souhaitée. Ensuite, il associe les afficheurs aux différents contenus générés lors de l'étape précédente. Comme pour le module de contenu, toutes ces opérations s'effectuent par simple « glissé-déposé » de la colonne de gauche vers la partie droite de la fenêtre.

Les afficheurs proposés par WINDMash sont :

- Afficheur texte : il reprend l'intégralité du texte brut, en marquant les références géographiques. Dans notre exemple (cf. Figure 13), l'afficheur texte est le cadre bleu dénommé « *My text viewer 1* », les références géographiques sont soulignées ;
- Afficheur carte : les données normalisées sont automatiquement transformées en géométrie sur une couche de la carte. Un point représente un emplacement, un lieu ; une ligne représente une route, une rivière, un itinéraire ; un polygone représente une région, une ville, etc. Le concepteur choisit son fond de carte (plan, satellite,…), l'outil met à disposition un certain nombre de fournisseurs (Google, IGN,…). Dans notre exemple (cf. Figure 13), nous avons inséré les lieux dans une carte GoogleMap (rectangle orange) ;
- Afficheur calendrier : les dates associées aux données peuvent être affichées ;
- Afficheur liste : permet d'afficher les données sous forme de liste. Dans notre exemple la liste « *My list Viewer 3* » contient les lieux alors que la liste « *My list Viewer 4* » contient les villes ;
- Afficheur photo : affiche une photographie que l'utilisateur charge à partir de son poste.

La description de l'organisation de l'interface de l'application étant achevée, le concepteur peut passer au module d'interaction.

2.2.3. Le module d'interaction

Dans ce module, le concepteur définit les comportements pour l'application finale. Il conçoit l'interaction entre les composants visuels définis dans le module d'interface.

La figure suivante (cf. Figure 14) montre l'interface utilisateur du module d'interaction qui permet au concepteur, sur le même principe que pour les autres modules de WINDMash, de décrire graphiquement les interactions supportées par son application.

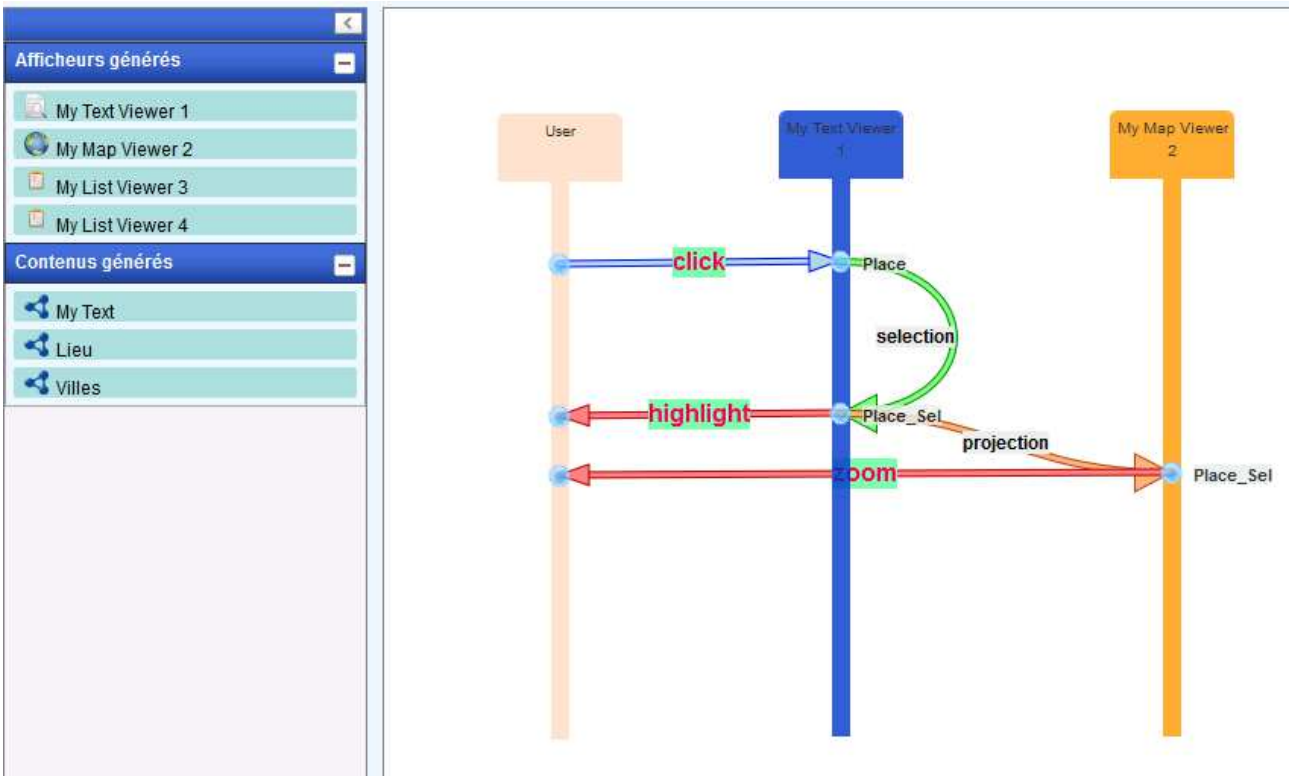


Figure 14 : Capture d'écran du module d'interaction

WINDMash représente la configuration de l'interface (partie droite de la figure en s'appuyant sur une représentation visuelle basée sur le même principe que les diagrammes de séquences UML. Chaque ligne verticale représente un élément acteur des interactions (l'utilisateur et les afficheurs). Les flèches représentent les interactions entre ces éléments.

Dans un premier temps, par glissé-déposé, le concepteur choisit les afficheurs sur lesquels il souhaite mettre de l'interaction.

Ensuite, il définit le type d'interaction qu'il souhaite mettre en place ; pour cela, il configure les interactions sur chaque afficheur (cf. Figure 15).

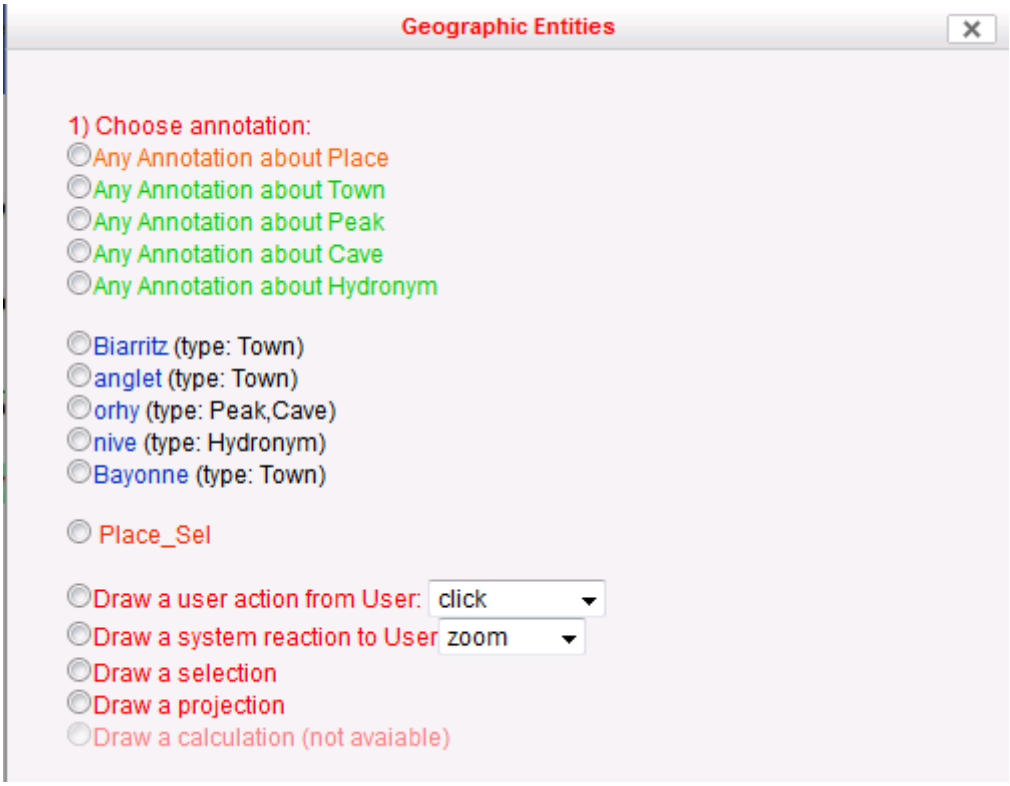


Figure 15 : Capture d'écran de la fenêtre de configuration des interactions

Le concepteur précise quels contenus (villes, préfectures, rivière…) doivent devenir support d'interaction (*Choose annotation*), ainsi que l'événement (clic ou survol de la souris) qui déclenche l'interaction *(Draw a user action from User)*.
Ensuite, le concepteur définit la réaction du système (zoom ou mise en évidence), suite au déclenchement de l'événement défini précédemment (*Draw a system reaction to User*).

Dans notre exemple (cf. Figure 14), lorsque l'utilisateur sélectionne un lieu dans l'afficheur de texte (*My text viewer 1*), le système enregistre la sélection (*Place_sel*). Nous avons ajouté deux réactions à l'événement : la première, est la mise en évidence du lieu sélectionné sur la fenêtre de texte (flèche rouge *highlight*). La seconde réaction est un zoom, au niveau de la carte, sur le lieu sélectionné (flèche rouge *zoom*).

2.2.4. L'application générée

Un simple bouton permet d'afficher un aperçu de l'interface graphique de l'application générée à l'aide de WINDMash. La figure suivante (cf. Figure 16) en est un exemple très représentatif, car elle fait ressortir à la fois le côté spatial et temporel de l'application générée. En plus d'informations spatiales affichées dans la carte, le texte brut contient des informations temporelles que le concepteur a mises en évidence dans un calendrier. Une interaction est configurée de façon à synchroniser les fenêtres textuelles, temporelles et spatiales.

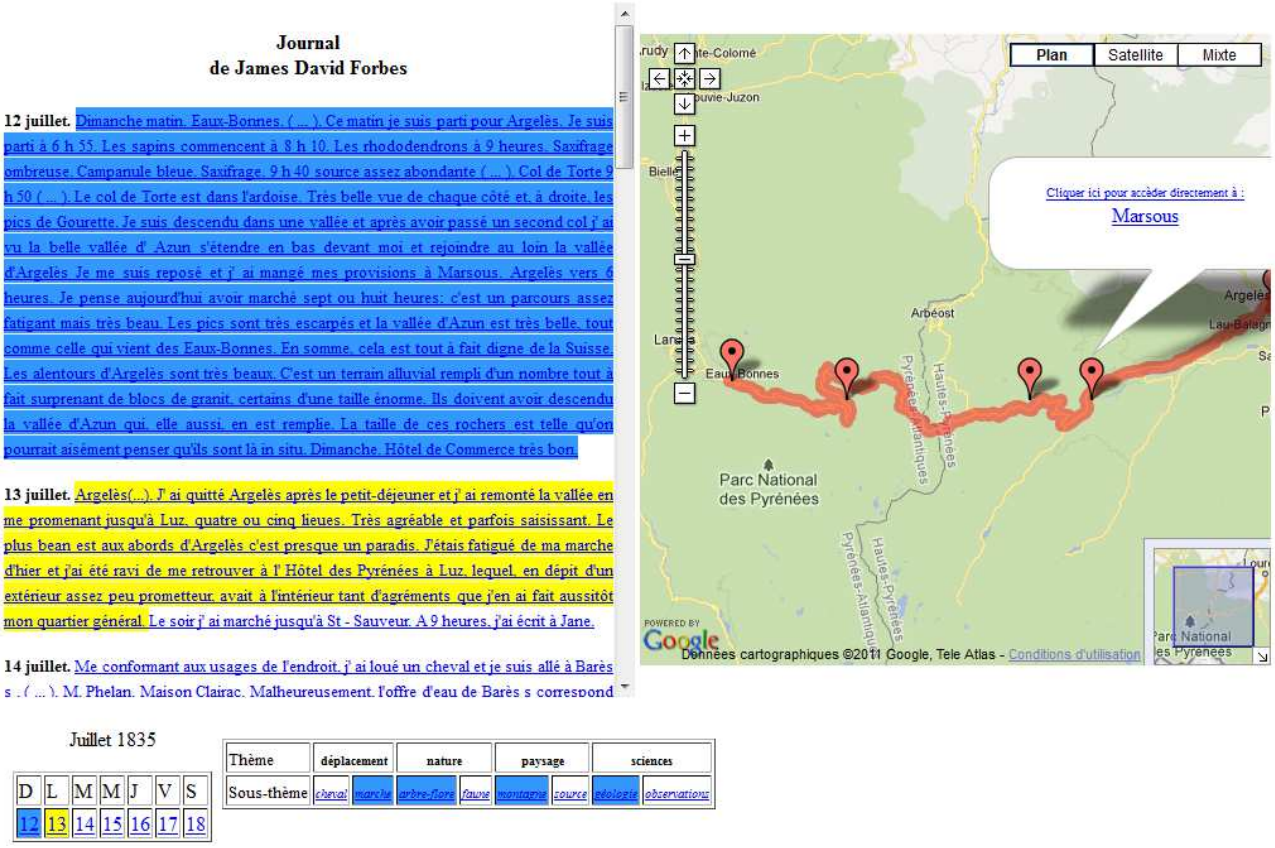


Figure 16 : Exemple d'une application générée par WINDMash (8)

2.3. Conclusion

L'objectif de ce chapitre était de faire l'état de recherches dans le domaine d'applications de cartographie en ligne.

Dans cette optique, nous avons retenu deux solutions sur lesquelles nous dressons cette conclusion.

En ce qui concerne WINDMash, la partie de l'application la plus intéressante est celle concernant la génération de l'application web. En effet, WINDMash est l'un des rares outils gérant l'aspect temporel. De plus, la construction de l'application s'effectuant graphiquement, elle rend son utilisation accessible à des utilisateurs non informaticiens.

Carmen est, quant à elle, un exemple d'application à base de composants open-sources, composée d'une architecture pérenne et d'une grande documentation. Cette application en ligne a des performances comparables aux outils propriétaires (MapInfo, ArcGIS, …) tout en respectant les standards OGC. Victime de son succès, plus de dix milles cartes consultées par jour, le site d'accompagnement est malheureusement souvent inaccessible. En terme d'objectif, Carmen tend vers le type d'application que l'on souhaite concevoir, malheureusement elle n'intègre pas l'aspect temporel.

Le tableau suivant (cf. Tableau 1) permet de comparer les deux solutions en fonction d'un certain nombre de critères.

Critères	Carmen	WINDMash
Types d'application	Projet d'état.	Prototype de laboratoire.
Accès	Limité à un certain nombre d'organismes.	Accès par n'importe qui après enregistrement.
Technologies utilisées	OWS, OpenLayer, WMS.	?
Standards	OGC	RDF
Sources des données	MapInfo, ARCView, Table (BDD)	BDD IGN, texte brut
Métadonnées	Oui	Oui
Données temporelles	Non	Oui
Données spatiales	Oui	Oui

Tableau 1 : Synthèse des solutions

L'état de l'art a permis de mieux appréhender le sujet de ce mémoire et a servi de base pour la réalisation du projet qui nous incombe.

Chapitre 3. GenGHIS - Analyse de l'existant

Dans ce chapitre, après une vue d'ensemble de l'architecture du logiciel, nous abordons en détail le fonctionnement et la structure de GenGHIS ainsi que les différents composants de l'application. Nous détaillons l'application GenGHIS et les SIST générés. Puis, nous dressons un bilan des concepts sur lesquels nous pourrons ensuite appuyer nos propositions.

3.1. Présentation générale de GenGHIS

GenGHIS est un logiciel permettant à un utilisateur non informaticien de construire sa propre application de visualisation interactive pour des données intégrant à la fois des dimensions spatiales et temporelles. Cette application est ensuite accessible directement depuis un navigateur internet quelconque.

L'application GenGHIS propose une interface graphique se présentant sous la forme d'un assistant guidant l'utilisateur dans le processus de création d'une application de visualisation (cf. Figure 17). L'interface principale est divisée en trois parties correspondant aux différentes étapes de ce processus :

Figure 17 : Interface générale de l'application GenGHIS

1. *Load a Data Model*, est le point d'accès aux écrans qui permettront l'acquisition des données.
2. *Load or Create a Presentation Model*, permet à l'utilisateur de décrire le contenu de l'interface graphique de l'application à générer et de la présentation des données.
3. *Generate the SIST application*, concerne finalement la génération du code de l'application de visualisation.

Enfin, l'interface est dotée d'une console permettant d'afficher des messages à l'utilisateur, sur le bon accomplissement ou l'échec des différentes étapes du processus.

La figure ci-dessous (cf. Figure 18) représente un exemple d'application générée avec GenGHIS. Cette interface graphique affichée dans la fenêtre d'un navigateur internet est constituée d'un cadre attributaire présentant les données, d'un cadre spatial sous forme de carte, et d'un cadre temporel sous la forme d'un graphique temporel pouvant être animé. Ces

différents cadres sont synchronisés entre eux, une interaction sur l'un des cadres (par exemple une sélection) étant répercutée immédiatement sur les autres.

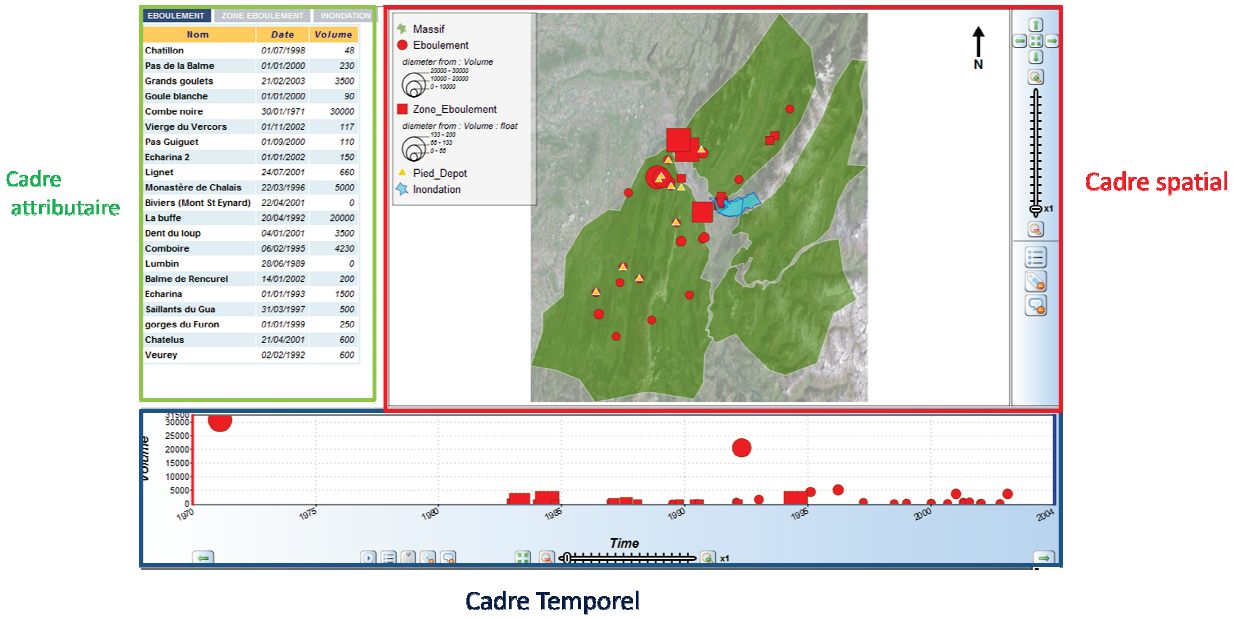


Figure 18 : Exemple de SIST généré

Par la suite, lorsque nous emploierons le nom GenGHIS, cela sera fait en référence à cette dernière application. Afin de ne pas confondre l'application GenGHIS proprement dite et une application générée par GenGHIS, nous emploierons par la suite le terme de SIST GenGHIS pour ces dernières.

3.2. L'application GenGHIS

3.2.1. Architecture générale de l'application GenGHIS

D'un point de vue architectural, l'application GenGHIS se décompose en trois modules correspondant aux trois étapes du processus de création d'un SIST GenGHIS telles qu'elles sont proposées par l'interface utilisateur :

- le module d'acquisition des données : ce module permet à l'utilisateur d'importer dans GenGHIS les données pour lesquelles il souhaite produire une application de visualisation interactive ;
- le module d'instanciation de présentations : ce module permet à l'utilisateur de spécifier la manière dont il souhaite représenter les données précédemment importées ;

- le module générateur de SIST : ce module effectue la génération du code pour l'application de visualisation à partir des informations fournies aux deux modules précédents.

Au cœur de GenGHIS, une base de connaissances AROM-ST permet l'intégration des données (au sens large, c'est-à-dire à la fois les données attributaires et les données propres à la présentation). AROM (Allier Relations et Objets pour Modéliser) est un langage de représentation de connaissances qui permet de définir à la fois des modèles de données et des modèles de présentation, modèles que l'utilisateur va instancier à l'aide des modules d'acquisition de données et de présentation. AROM-ST est une extension d'AROM qui permet la prise en compte des informations spatiales et temporelles.

C'est à partir de ces modèles instanciés que le générateur de GenGHIS peut ensuite produire l'ensemble de fichiers constituant l'application de visualisation (le SIST, Systèmes d'Information Spatio-Temporelle). Cet ensemble de fichiers, regroupé dans une page HTML, forme un tout, contenant à la fois les données ainsi que les différents éléments, permettant la navigation et les animations. L'interface de visualisation interactive ainsi construite, est directement affichable dans un navigateur Web.

3.2.2. Technologies utilisées

Comme nous l'avons vu dans la partie précédente, l'application GenGHIS et les SIST GenGHIS, constituent deux parties indépendantes qui utilisent des technologies différentes. La figure suivante (cf. Figure 19) dresse un schéma de la décomposition architecturale de GenGHIS, afin d'illustrer les différentes technologies mises en œuvre.

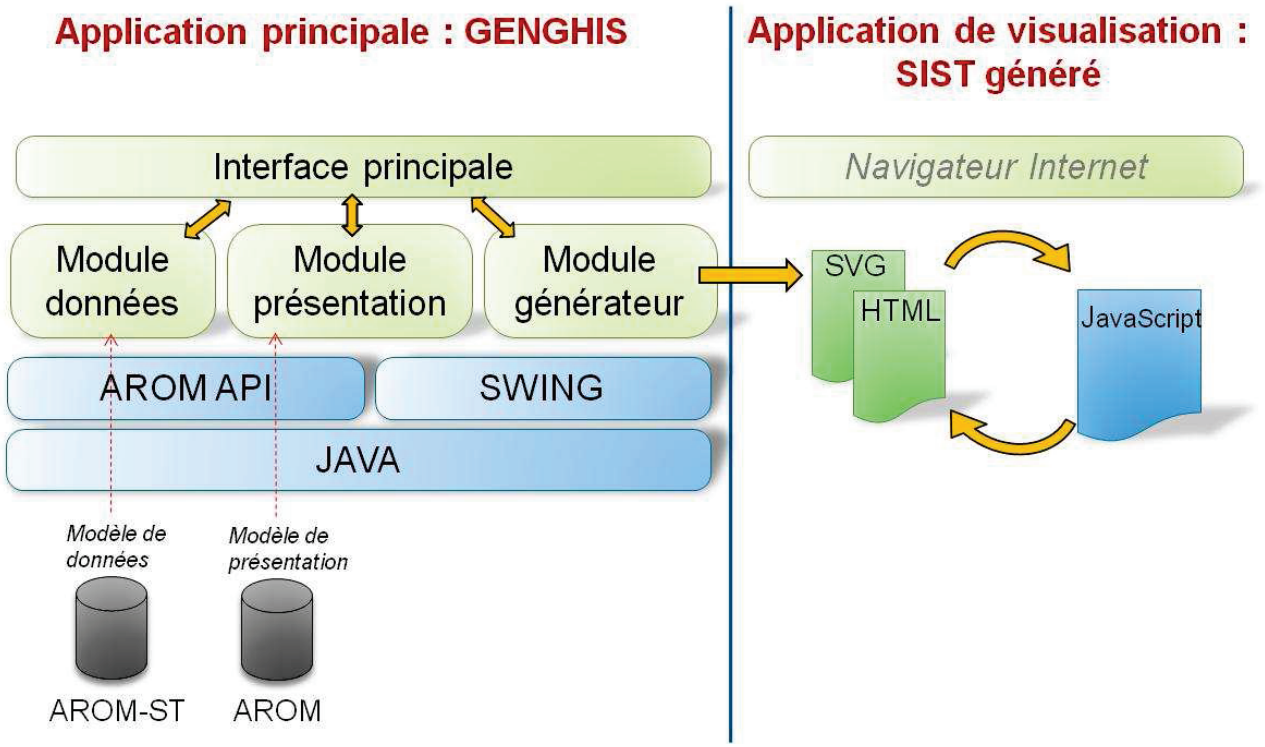


Figure 19 : Technologies mises en œuvre dans GenGHIS (3)

Du côté de l'application GenGHIS, nous retrouvons dans le bas du schéma AROM et son extension AROM-ST, le système de stockage et de modélisation des données sur lequel s'appuie GenGHIS. Les fondations de l'application se basent sur la technologie JAVA. Au dessus se trouvent la bibliothèque graphique SWING ainsi que l'interface de programmation AROM-API permettant de manipuler les entités de modélisation AROM et AROM-ST. La couche supérieure représente le cœur de l'application et les éléments de l'application en relation avec l'utilisateur (IHM).

L'application de visualisation générée est un mixte de technologies HTML, SVG[17] (*Scalable Vector Graphics*) et JavaScript.

Dans la suite du document nous décrivons plus précisément ces briques logicielles.

3.2.2.1. AROM

AROM (Allier Relations et Objets pour Modéliser) est à la fois un langage de représentation de connaissances centré objets (RCO) et une plate-forme logicielle pour l'exploitation de modèles décrits dans le langage AROM. AROM permet une modélisation des données proche du paradigme objet. Pour cela, il se base sur les concepts de « Classe » et d' « Association » pour représenter les objets et les liens qui les unissent.

Les classes AROM sont constituées d'un ensemble d'attributs (ou variables) caractérisées par un ensemble de contraintes (type, cardinalité,..). Comme dans un langage objets, ces classes sont structurées de manière arborescente selon une relation de spécialisation. Contrairement à un langage de programmation, le langage AROM ne définit pas d'instructions permettant d'effectuer des traitements sur les objets modélisés : les classes ne possèdent pas de méthodes[18].

En AROM, les relations entre objets sont explicitées à l'aide d'associations. Ces associations sont des entités de modélisation à part entière, de même niveau que les classes. Cette particularité démarque AROM des autres modèles qui sont essentiellement basés sur la notion d'attributs/liens. Les associations AROM sont semblables aux classes d'association employées en UML[19] : une association représente un ensemble de liens similaires entre plusieurs classes, distinctes ou non. Les associations peuvent être d'arriter quelconque (il n'y a pas de restrictions sur le nombre de classes reliées) et peuvent posséder leur propres variables (attributs). Tout comme les classes, les associations peuvent être spécialisées par l'ajout de nouvelles variables. Dans la terminologie AROM, les classes reliées par une association définissent chacune un rôle. Une instance d'association est appelée *tuple* (2) : elle regroupe les *objets* (instances de classes AROM) liés par cette relation ainsi que les valeurs des éventuels attributs associés.

AROM intègre la plupart des mécanismes d'inférence présents dans les systèmes de représentation de connaissances, tels que la valeur par défaut, l'héritage, l'attachement procédural, le filtrage et la classification (2).

[17] Graphique vectoriel adaptable est un format de données basé sur XML conçu pour décrire des ensembles de graphiques vectoriels

[18] Il existe dans AROM un langage de modélisation algébrique (LMA) qui permet la définition d'expression permettant de déterminer la valeur d'un attribut.

[19] UML : *Unified Modeling Language* est un langage de modélisation graphique à base de pictogrammes.

AROM propose un langage textuel pour la description des classes, associations, objets (instances de classes) et *tuple* (instances d'association). AROM offre également une notation graphique pour la représentation des modèles au travers de l'utilisation de l'environnement de développement AROM-IME (*Integrated Modelling Environement*). Comme le montre la figure suivante (cf. Figure 20), les notations graphiques utilisées dans AROM sont très proches des notations UML pour les diagrammes de classes, hormis quelques détails de représentation qui divergent.

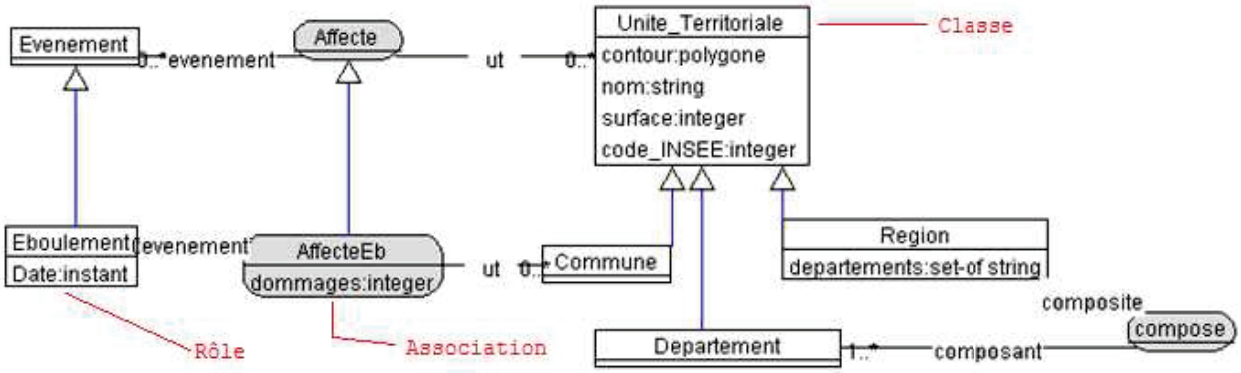


Figure 20 : Exemple de modèle réalisé à partir de l'IME d'AROM(3)

De plus, le stockage du modèle et de son contenu s'effectue sous la forme d'un fichier texte éditable. Ce type de stockage a l'avantage d'être facilement lisible car il n'est pas nécessaire de passer par un logiciel spécialisé pour effectuer une quelconque modification. En revanche, il en découle une baisse de performances lorsque le nombre d'occurrences est important.

Enfin, une particularité d'AROM est sa capacité d'adapter son méta-modèle à un domaine spécifique en ajoutant aux types et opérateurs de base d'AROM de nouveaux types et opérateurs. Cette possibilité d'extension permet, comme nous allons le voir dans la section suivante, de rajouter une représentation du temps et de l'espace à AROM.

3.2.2.2. AROM-ST

AROM, dans sa version générique, ne propose pas de constructions permettant de représenter directement le temps et l'espace ; aussi a-t-il été étendu en AROM-ST (ST pour Spatio-Temporel), afin d'introduire les concepts et les types spatio-temporels(9).

Les types spatiaux d'AROM-ST (cf. Figure 21) sont conformes aux spécifications de l'OGC (10).
La figure ci-après énumère les différents types géométriques(2) :

- le point (classe *Point*), défini par une abscisse et une ordonnée ;
- la polyligne (classe *Polyline*), composée d'une succession de points ;
- le polygone (classe *Polygon*), défini par une polyligne fermée qui constitue son contour, et éventuellement un ensemble de polylignes fermées qui constituent ses trous ;
- les types *Line* et *LinearRing* sont définis en appliquant des contraintes sur le type *Polyline*. Une ligne est une polyligne contenant exactement deux points. Une polyligne fermée (classe *LinearRing*) possède un point de départ identique au point d'arrivée et doit contenir au moins 3 points.

A partir de ces formes simples, sont définis des types géométriques complexes : *MultiPoint* (nuage de points), *MultiLine* (un ensemble de polylignes) et *MultiArea* (ensemble de polygones).

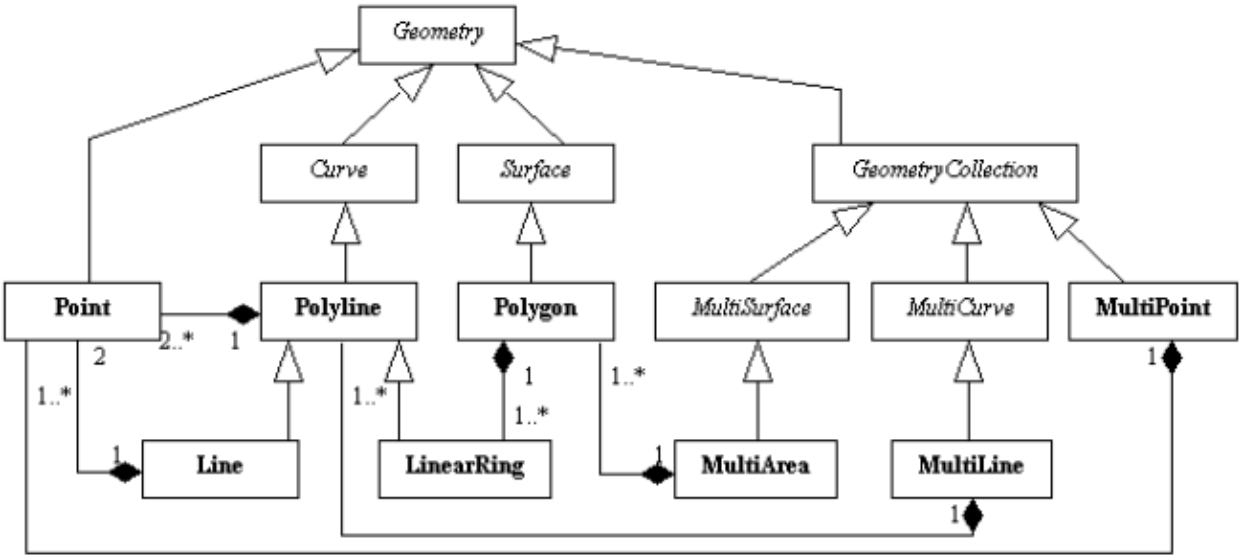


Figure 21 : Modèle géométrique d'AROM-ST (9)

Les types temporels simples en AROM-ST sont le type *Instant* et le type *Interval*. Les types composés, constituant des collections d'instants ou d'intervalles, sont les types *MultiInstant* et *MultiInterval* (cf. Figure 22).

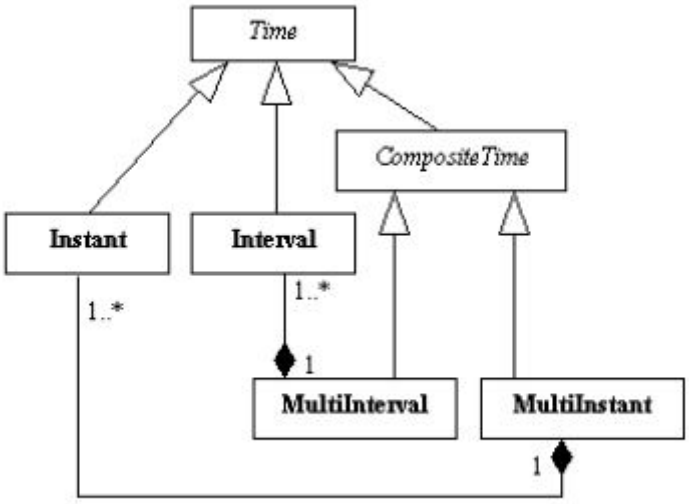


Figure 22 : Types temporels de données (9)

3.2.2.3. Java/Swing

L'application GenGHIS est un logiciel de type client lourd[20] développée dans le langage de programmation orientée objets : Java. Le choix du langage Java a été dicté par le fait que la plate-forme AROM a elle-même été développée dans ce langage et est accessible au travers d'une API (*Application Programming Interface*) Java. L'un des avantages de Java, est sa portabilité : grâce à la machine virtuelle Java (JVM) qui interprète le Bytecode généré par le compilateur Java, une même application peut s'exécuter sur des plates-formes (Système d'exploitation + matériel) différentes. Java offre en outre de nombreuses bibliothèques standards pour faciliter le développement d'applications. C'est en particulier sur la bibliothèque Swing et ses composants graphiques standards que s'appuie GenGHIS pour la réalisation de son interface utilisateur.

3.2.2.4. CSS/HTML/SVG/JavaScript

CSS (*Cascading Style Sheets*), HTML, SVG et Javascript, utilisées dans l'application de visualisation font partie des technologies de l'internet les plus utilisées.
Les feuilles de style en cascade (CSS) vont de paire avec HTML, elles permettent de définir une séparation nette entre la présentation et la structure d'un document. HTML (ou XHTML) décrit cette structure, CSS permet de lui appliquer une mise en forme en définissant un ensemble de styles graphiques qui sont appliqués au document HTML. La modification de la présentation ne dépend alors que de la modification des feuilles de style CSS.
SVG est un langage basé sur XML. Il utilise différentes balises propres à son format pour décrire des formes géométriques 2D tels que des rectangles, des cercles, etc. Il permet à la fois de décrire des formes de bases mais aussi des formes plus sophistiquées appelées chemins, qui autorisent l'affichage de formes complexes (polygones, multi-lignes, courbes de Bézier, etc.). Sa simplicité en fait un outil largement reconnu dans la cartographie interactive.
Pour finir, JavaScript est un langage de programmation de scripts principalement utilisé dans les pages web interactives. Il permet de manipuler à la fois la structure de documents HTML affichés (modèle DOM) et le style CSS associé.

3.2.3. Composants de GenGHIS

3.2.3.1. Structure générale de GenGHIS

L'application GenGHIS respecte dans sa conception le modèle standard MVC (Modèle Vue Contrôleur)[21] sur lequel sont également basés les composants graphiques de la bibliothèque Java Swing utilisée pour la réalisation de l'interface utilisateur. Elle est constituée d'un contrôleur principal (classe *ControlerGenGHIS*) qui oriente les flux de traitement entre les contrôleurs spécifiques de l'interface gestionnaire du modèle de données (classe *ControlerEditKBData*), de l'interface gestionnaire du modèle de présentations (classe *ControlerEditKBPresentation*) et le module générateur de code(classe *Generator*), ainsi que la

[20] Logiciel qui propose des fonctionnalités complexes avec un traitement autonome.
[21] Le modèle MVC est un patron de conception qui organise l'interface homme-machine.

classe *KBHandler* qui permet d'interagir avec les bases de connaissances servant à modéliser les données échangées par ces différents composants. La figure suivante (cf. Figure 23) reprend les éléments cités précédemment.

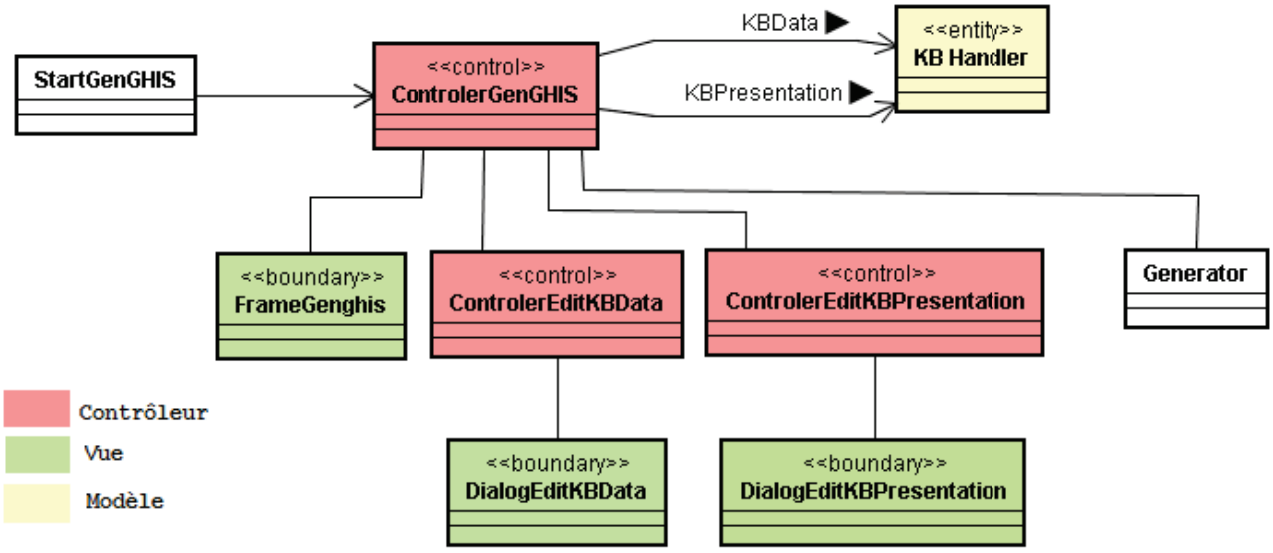


Figure 23 : Diagramme de classes de la structure générale de GenGHIS (3)

3.2.3.2. Le module de données

Sources des données

Dans GenGHIS, il est possible d'importer des données à partir de deux types de fichiers :

- les fichiers MIF/MID (*MapInfo Interchange Format*) : c'est le format d'échange ou d'export du logiciel MapInfo. Le fichier « .mid » contient les informations attributaires, et le fichier « .mif » contient la structure de la table et la géométrie ;
- les fichiers Excel : à partir d'une feuille de calculs du tableur Excel, GenGHIS considère chaque colonne comme les valeurs d'un champ défini par le nom et le type de la colonne.

Etant donné le nombre de solutions logicielles pour les systèmes d'information géographique et donc la provenance multiple des données à importer, une solution commune et extensible aux différents formats de fichiers a été implémentée. Une interface commune *DataFeature* définie une abstraction des données et permet d'appliquer le concept du polymorphisme aux différentes sources de données : la prise en compte d'un nouveau type de fichier de données nécessitant simplement la réalisation de cette interface au travers d'une classe spécifique. La figure suivante (cf. Figure 24) présente un diagramme de classes montrant les classes supportant les différents formats de fichiers de données pouvant être importés dans GenGHIS. Elle fait aussi apparaître le package AROM API qui lui-même utilise les composants géométriques de la librairie JTS. Cette librairie permet de gérer les différentes formes géométriques supportées par AROM-ST(3).

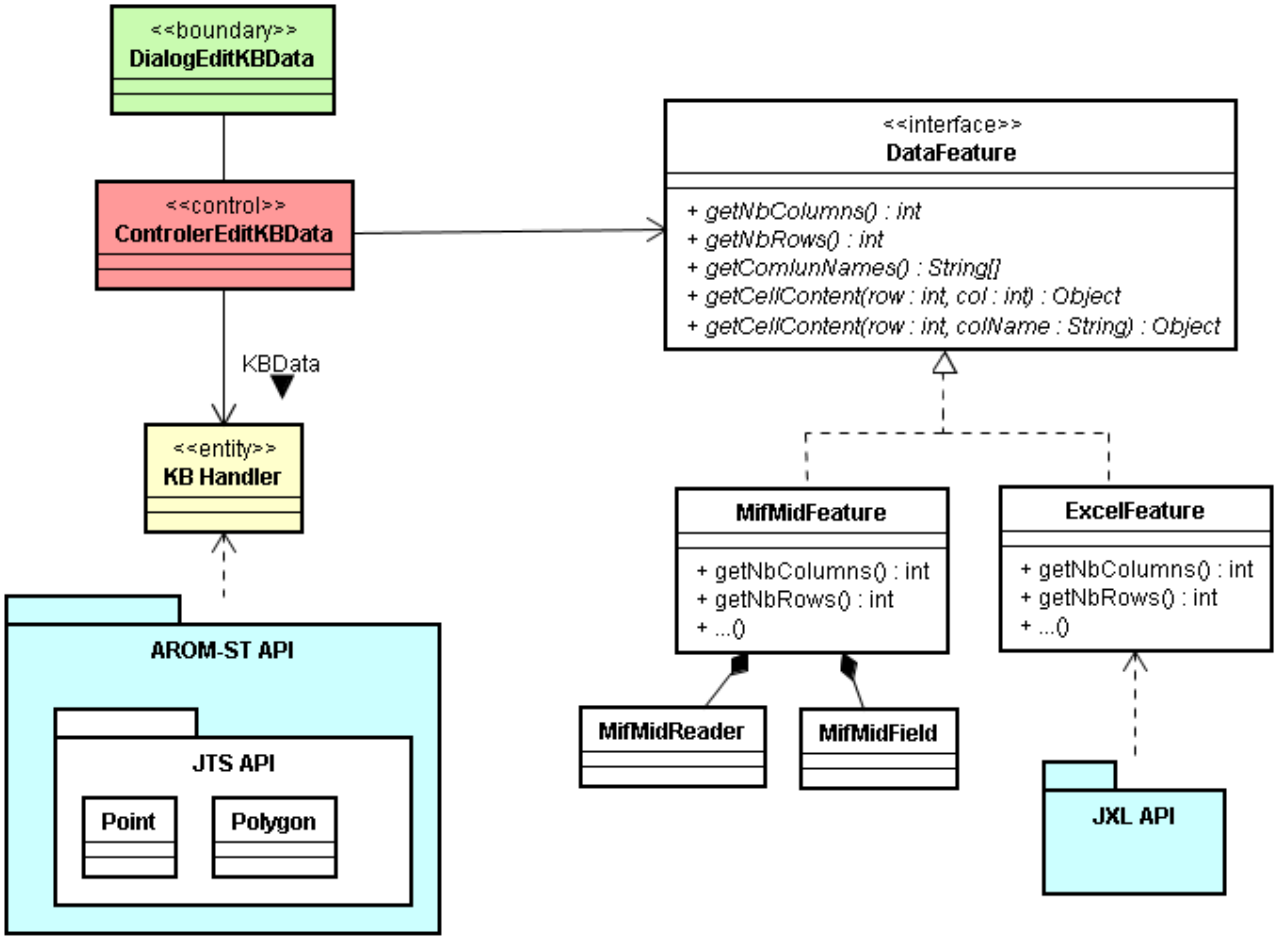


Figure 24 : Diagramme de classes des formats de fichiers (3)

A terme le système peut être étendu à d'autres formats de fichiers tels que le « .shp » du logiciel ArcView.

L'interface d'acquisition des données

La première étape pour l'acquisition des données est le chargement d'un modèle de données valide. L'utilisateur choisit un modèle de données au format AROM-ST, représenté par un fichier de type « .txta[22] » enregistré sur le disque. Le contenu de ce fichier renseigne sur la structure des classes et associations du modèle de données spatio-temporelles. Le rôle principal de ce modèle est de prédisposer, en fonction d'un domaine particulier, les structures de classes propres à ce dernier. Prenons par exemple le domaine des risques naturels où le modèle AROM-ST pourrait comporter les classes : commune, éboulement, inondation,…

La seconde étape consiste à charger un fichier de données externe (MIS/MID ou Excel), tel que décrit dans la partie précédente. Ce fichier contient les informations des instances à importer.

Ensuite, l'utilisateur doit faire correspondre les attributs des entités AROM du modèle de données avec les différents champs des données importées. Cette mise en correspondance

[22] le format "txta" est le format textuel des bases de connaissances AROM. Il existe également un format binaire, plus performant, utilisé lorsque la base de connaissances contient de nombreuses instances (objets et tuples).

s'effectue interactivement à l'aide de l'interface graphique présentée sur la Figure 25. Celle-ci montre un exemple d'association entre les attributs d'une classe `Commune` et les valeurs de champs situés dans un fichier mif/mid. Nous avons fait correspondre l'attribut « Contour » de la classe Commune au champ « REGION » du fichier de données et l'attribut « Nom » au champ « NOM_COM ».

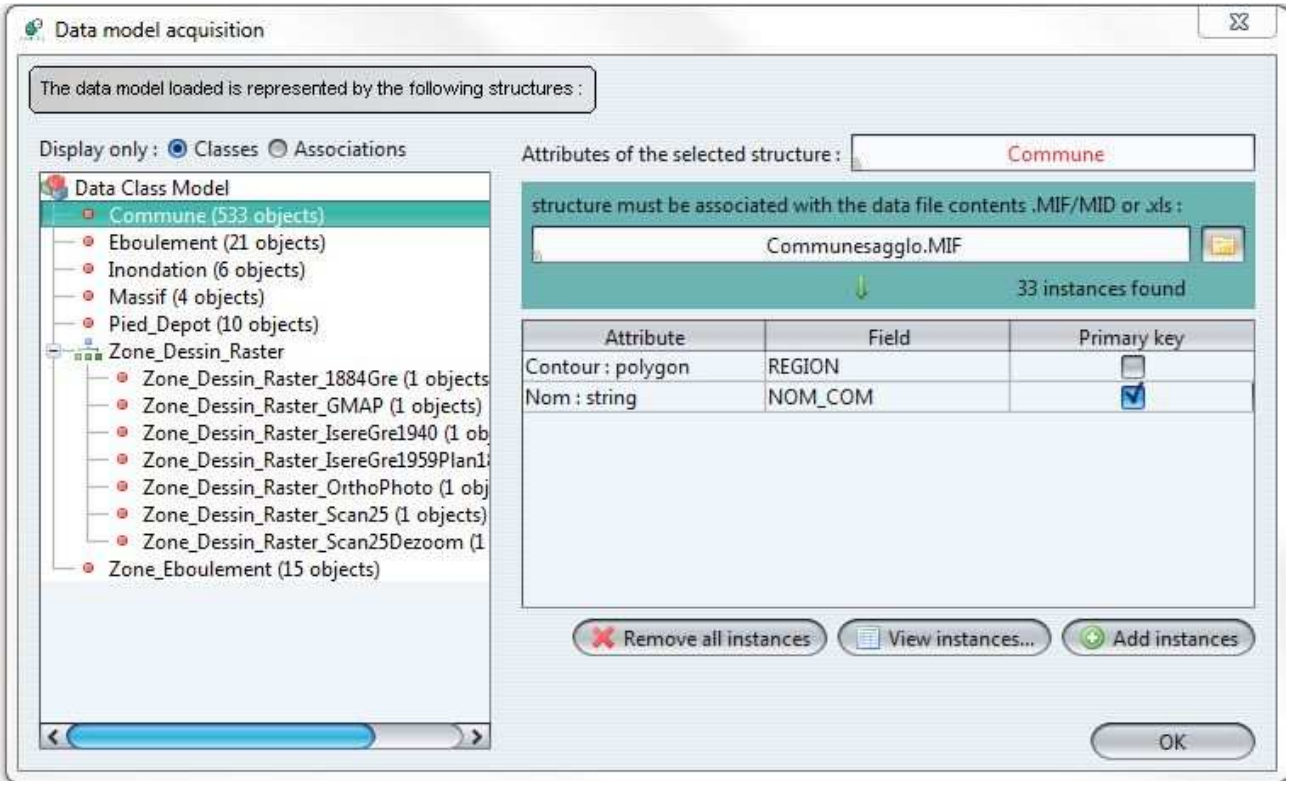


Figure 25 : Capture d'écran de la fenêtre d'acquisition

Une fois l'ensemble des correspondances entre le modèle AROM-ST et les différents champs du fichier de données définis, l'utilisateur peut procéder à l'importation des données proprement dite à l'aide du bouton « *add instances* ». Comme son nom l'indique, l'activation de ce bouton déclenche la création d'instances du modèle AROM à partir des valeurs contenues dans le fichier de données.

Il faut noter la nécessité d'indiquer une clé primaire, afin d'éviter l'ajout de doublons dans la base de connaissances, lors de mise à jour multiples.

L'interface de visualisation et de modification des données

Une fois les données importées, il est possible de visualiser et/ou de modifier les instances AROM qui les portent. Cette opération est accessible depuis la fenêtre d'acquisition des données. Après avoir sélectionné une entité dans la liste et avoir cliqué sur le bouton « *view instances..* », une interface sous la forme d'un tableur apparaît (cf. Figure 26). Cette fenêtre indique l'entité sélectionnée (*Selected structure* : Commune) ainsi que les attributs la composant (1er ligne du tableau : Contour et Nom). Les autres lignes correspondent aux instances (enregistrements) de l'entité décrite. Cette interface permet d'effectuer des opérations unitaires sur les instances d'une entité comme la création, la suppression ou la modification d'un enregistrement(3) ; elle agit sur les attributs des objets du modèle. Sur le

même principe qu'un tableur, les cellules sont éditables. Après avoir effectué les modifications souhaitées, le contenu du tableau est sauvegardé directement dans la base AROM.

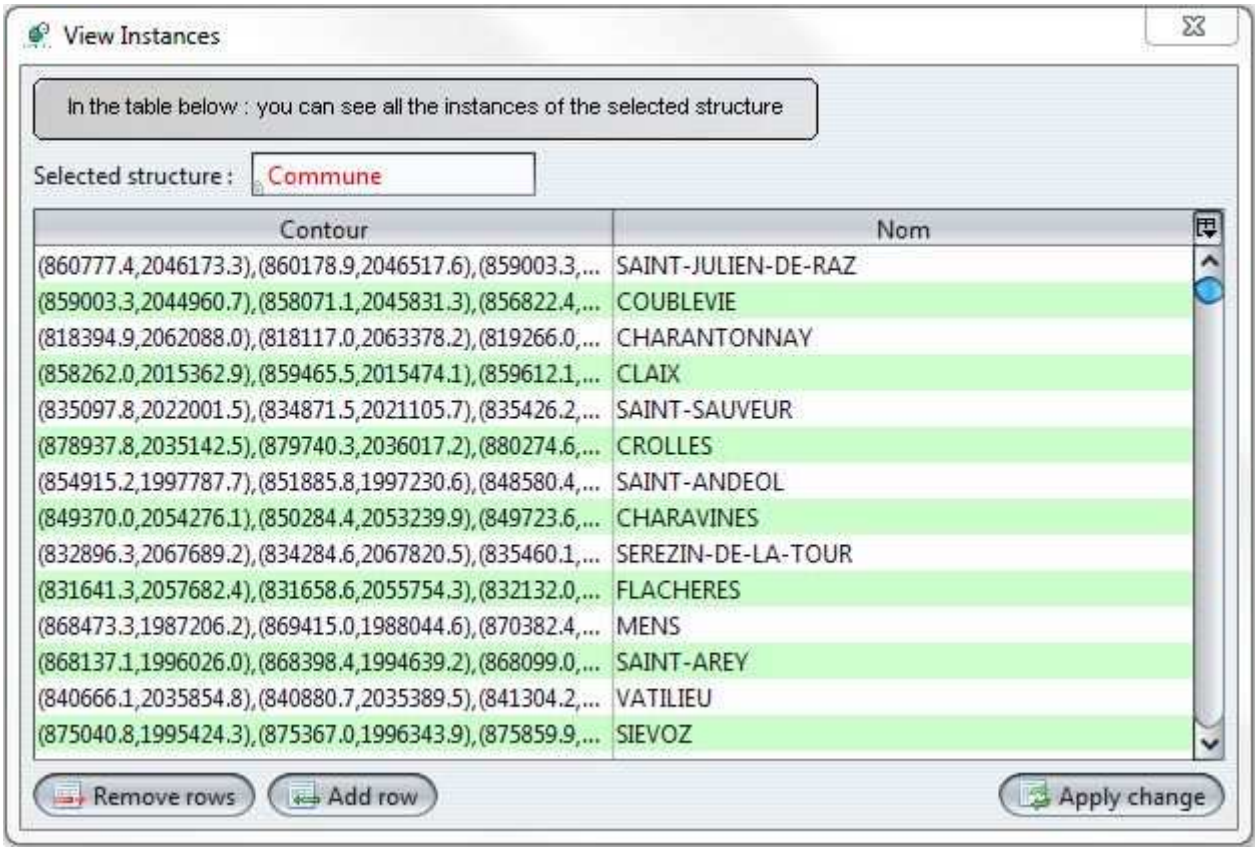


Figure 26 : Capture d'écran de l'interface de visualisation et modification des enregistrements

3.2.3.3. Le module de présentation

Le module de présentation se base sur le standard *Styled Layer Descriptor* (Descripteur de Couche Stylisé) spécifié par l'*Open Geospatial Consortium* afin de décrire le style multi-échelle des couches de carte. Il rajoute aussi les styles variables, le dynamisme (3 états graphiques) et des extensions pour le temporel et l'attributaire. Une présentation est fractionnée en trois éléments structurels imbriqués : la fenêtre, les couches et les styles. La figure suivante (cf. Figure 27) schématise les relations entre les éléments structurels d'une présentation.

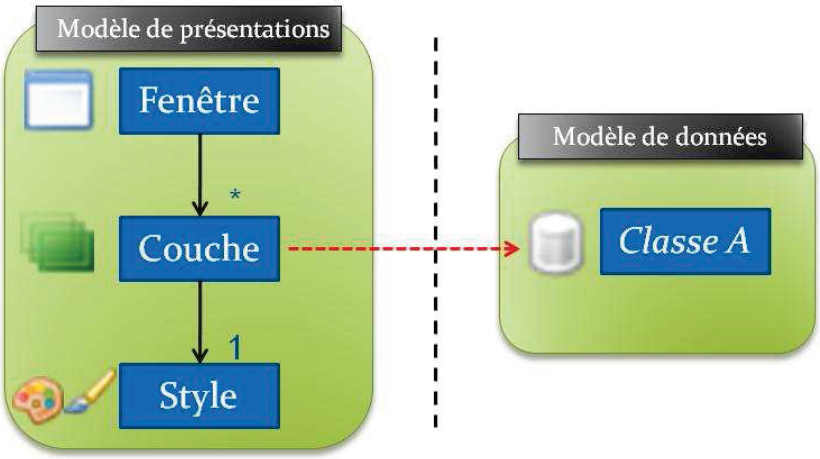


Figure 27 : Eléments structurels d'une présentation (3)

L'élément fenêtre

Une fenêtre fournit un cadre et une zone d'affichage à des couches. GenGHIS propose 3 sortes de fenêtres :

- les fenêtres spatiales ;
- les fenêtres temporelles ;
- les fenêtres attributaires.

La figure suivante (cf. Figure 28) donne une représentation des fenêtres dans GenGHIS.

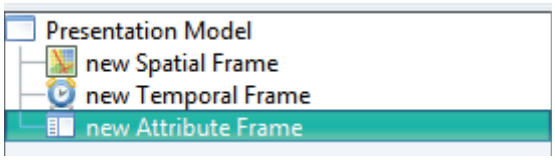


Figure 28 : Les types de fenêtres dans GenGHIS

L'élément couche

Une couche est liée à une classe du modèle de données et permet d'afficher selon un style visuel, la totalité des instances de cette classe (3).

Cet élément générique est dérivé en éléments spécifiques selon les composantes dimensionnelles de la présentation :

- sous sa forme spatiale, une couche dispose comme propriétés, d'une composante spatiale (*Spatial Slot*) qui doit être de type géométrique spatial, et une composante thématique (*String Slot*) de type chaîne de caractères ;
- sous sa forme temporelle, une couche possède comme propriétés, une composante temporelle (*Temporal Slot*) qui doit être de type « date », une composante « axe des ordonnées » permettant d'indiquer quelle donnée sera représentée sur le graphique temporel, et une composante thématique (*String Slot*) de type chaîne de caractères ;
- sous sa forme attributaire, une couche est représentée par un onglet qui montre une classe donnée. Chaque onglet possède une ou plusieurs colonnes représentant les attributs de la classe. Par exemple, pour la classe inondation, nous pouvons faire apparaitre les colonnes : nom, date, intensité,…

L'élément style

Un style est rattaché à une couche et par conséquent il l'est indirectement à une classe du modèle de données. Il permet de stocker toutes les caractéristiques visuelles dont doivent hériter les objets à dessiner (3). Un style est constitué d'un ensemble de symboles.

Sur une carte, les symboles ont une place prépondérante car ils représentent l'abstraction. Dans GenGHIS, ils encapsulent les caractéristiques graphiques d'un style géographique. Les symboles de l'application peuvent être appliqués à une forme polygonale, une forme linéaire, une forme de type point, mais aussi aux formes ponctuelles et aux formes complexes. Il est possible, par exemple, d'appliquer un style de façon à représenter une géométrie définie par un point à l'aide d'un cercle ou d'un rectangle.

GenGHIS offre la possibilité de faire varier la taille ou la couleur d'un objet graphique en fonction de la valeur changeante de l'attribut d'une classe de rattachement du modèle de données.

Dans le but de faciliter la tâche de l'utilisateur lors de la définition d'un style, il lui est possible de dupliquer un style existant et ensuite de le modifier.

Il est à noter que les règles proposées dans GenGHIS ont été simplifiées et que par conséquent elles ne respectent pas complètement les principes de sémiologie graphique appliqués à la cartographie.

La figure suivante (cf. Figure 29) montre la fenêtre de l'assistant pour la construction d'un modèle de présentations. La partie gauche présente, sous la forme d'un arbre, la structure des éléments fenêtres, couches et styles. La partie droite varie en fonction de l'élément sélectionné dans la partie gauche. Dans cet exemple, l'affichage correspond aux propriétés d'un symbole défini pour un style associé à une couche, sur laquelle sont visualisés les objets de type Massif du modèle de données.

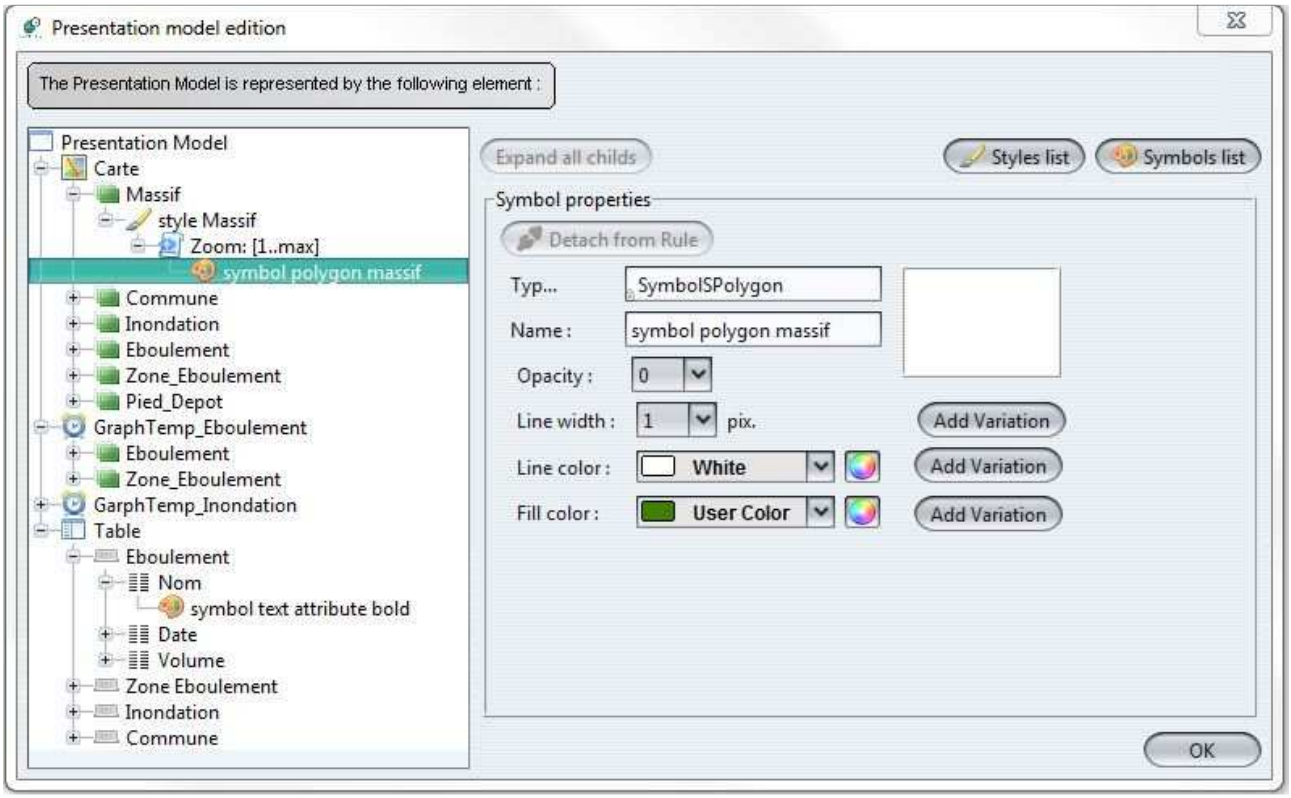


Figure 29 : L'interface du module de présentation

3.2.3.4. Le module générateur d'un SIST GenGHIS

Nous allons aborder le dernier module de l'application GenGHIS : le générateur de SIST. Ce module a pour objectif la construction de l'application selon des exigences que l'utilisateur a renseignées dans les modules de données et de présentation. Il est dépourvu d'interface graphique puisqu'il est totalement automatique et transparent pour l'utilisateur.

Comme le montre la figure suivante (cf. Figure 30) le générateur se base sur le modèle de données et sur le modèle de présentations, respectivement de types AROM-ST et AROM. On peut aussi remarquer la présence de ressources sous forme d'images et de fichiers JavaScript.

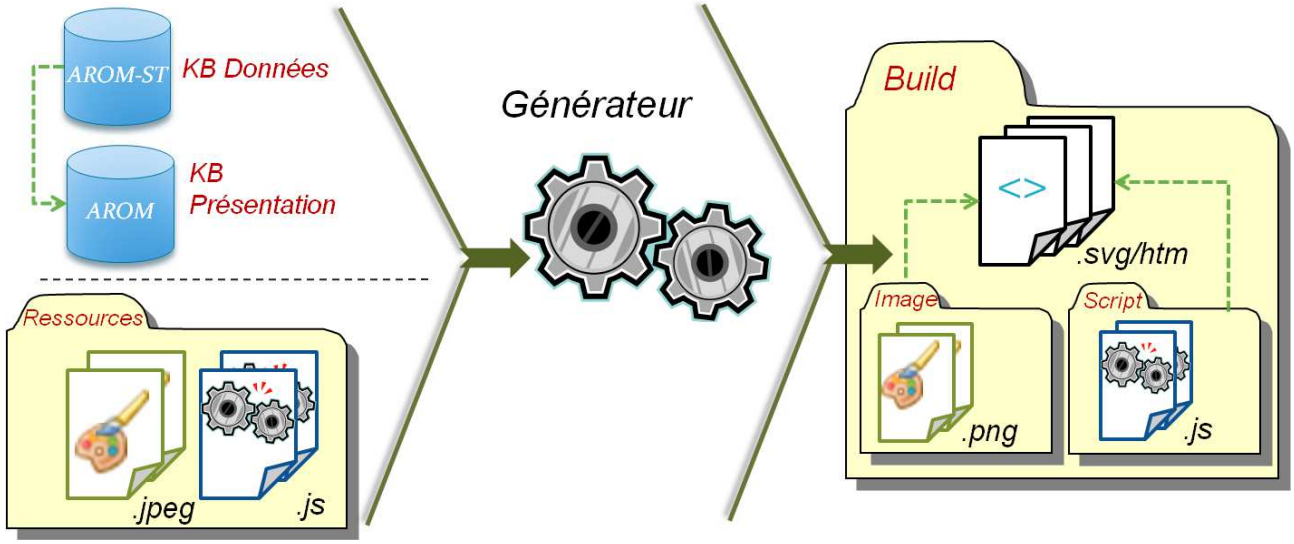


Figure 30 : Données d'entrée et de sortie du générateur (3)

Les fichiers JavaScript gèrent l'animation et l'interaction dans l'application générée, ils permettent à l'utilisateur de modifier directement les informations contenues dans les fichiers HTML et SVG. Ces fichiers JavaScript sont prédéfinis et ne sont pas modifiés par le générateur ; les interactions possibles entre l'utilisateur et les divers éléments visuels SVG/HTML ont été déterminées à l'avance. Le rôle du générateur est donc simplement de référencer dans les fichiers SVG et HTML générés, les fichiers JavaScript en fonction des besoins.

Le processus de génération de l'application SIST, s'effectue en partant des éléments de base, ne pouvant plus être décomposés, tels que les styles et les formes visuelles SVG. Ceux-ci sont ensuite imbriqués pour former des composants de plus en plus complexes jusqu'à obtenir la page HTML hébergeant l'application finale. La figure suivante (cf. Figure 31) montre l'imbrication des différents éléments visuels générés.

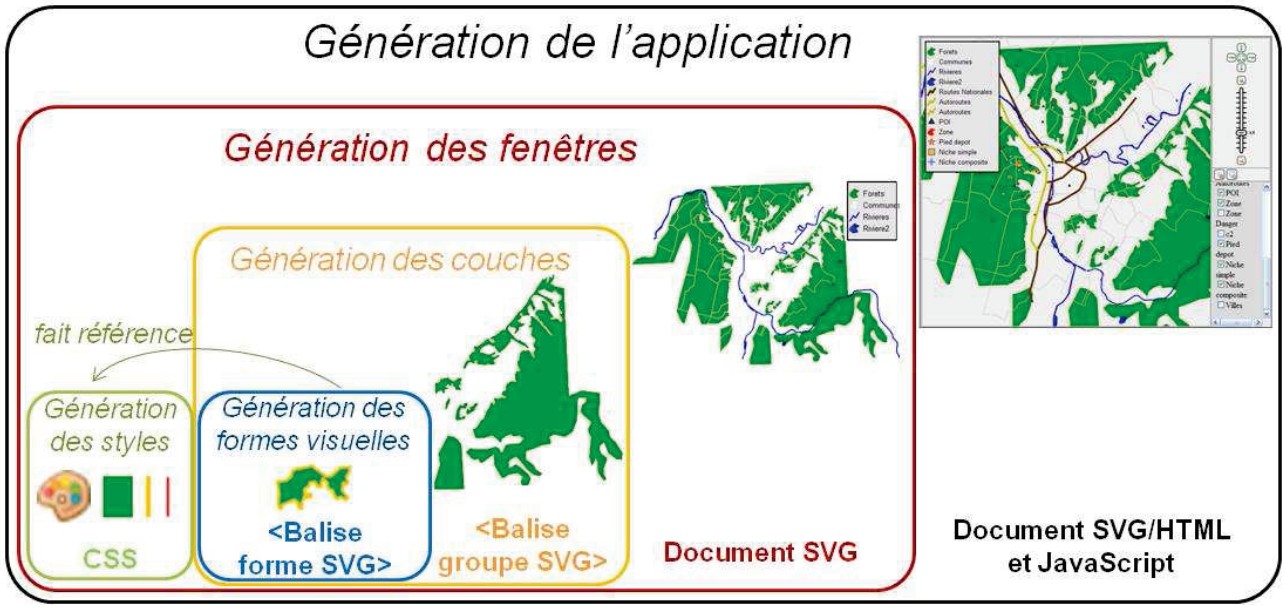


Figure 31 : Schéma de l'imbrication des éléments visuels générés (3)

- *Les styles*

Les styles décrits dans le modèle de présentations sont transformés en des chaînes de styles CSS. Chaque style CSS est une chaîne de caractères composée d'un nom suivi d'un ensemble d'attributs entre accolades. Les attributs définissent le style graphique et sont accompagnés par leurs valeurs. L'exemple ci-après présente un style de nom *symbol-polygon-massif-Normal*, et possédant les attributs *opacity* et *pointer-events* avec les valeurs respectives *1.0* (complètement opaque) et *visible*.

```
.symbol-polygon-massif-Normal { opacity :1.0 ;pointer-events :visible ; }
```

D'une manière générale chaque style est appliqué à une ou plusieurs formes visuelles.

- *Les formes*

Les formes sont traduites par des balises SVG propres à l'élément graphique à instancier. Dans une couche de la présentation, pour chaque entité Symbole, il sera créé autant d'éléments graphiques qu'il y a d'instances de la classe du modèle de données associées à la couche. Ces éléments graphiques créés, contiennent les coordonnées des formes géométriques, ou du texte. L'attribution des caractéristiques graphiques à un élément s'effectue au travers d'une référence au style CSS correspondant dans la représentation SVG de l'élément. L'exemple ci-dessous est celui d'un polygone où le style CSS est celui de l'exemple précédant :

```
<polygon id="Object:Massif620372826" class="symbol-polygon-massif-Normal"  points="106.0 247.0, 107.0 247.0, 107.0 247.0, 107.0 247.0, 107.0 248.0, 107.0 248.0, 107.0 248.0, 108.0 248.0, 107.0 248.0, 107.0 248.0, 107.0 248.0, 106.0 247.0, 106.0 247.0, 106.0 247.0, 106.0 247.0" />
```

- *Les couches et les fenêtres*

Après avoir généré les styles CSS et les balises de forme SVG, le générateur encapsule ces composants dans des couches et des fenêtres. Les documents HTML et SVG sont structurés de façon à restituer le plus clairement possible l'information. Les fichiers SVG servent de présentation à la partie spatiale et temporelle de l'application. Les fichiers HTML, quant à eux, sont aussi utilisés pour la partie attributaire. Ils servent également de cadre de présentation pour l'adaptation au navigateur internet.

3.3. Le SIST généré

Dans cette section, nous décrivons plus précisément les différents composants d'une application SIST générée par GenGHIS, notamment les différents types de volets que propose GenGHIS pour l'interface utilisateur de l'application : le volet spatial, le volet temporel et le volet attributaire. Enfin, nous décrivons le système d'interaction et de synchronisation entre ces différents volets.

3.3.1. Le volet spatial

Le volet spatial ou cartographique est un dessin SVG où sont affichées les différentes couches d'objets spatiaux (cf. Figure 32). Les menus et les infos bulle sont réalisés à l'aide de la technologie HTML/CSS.

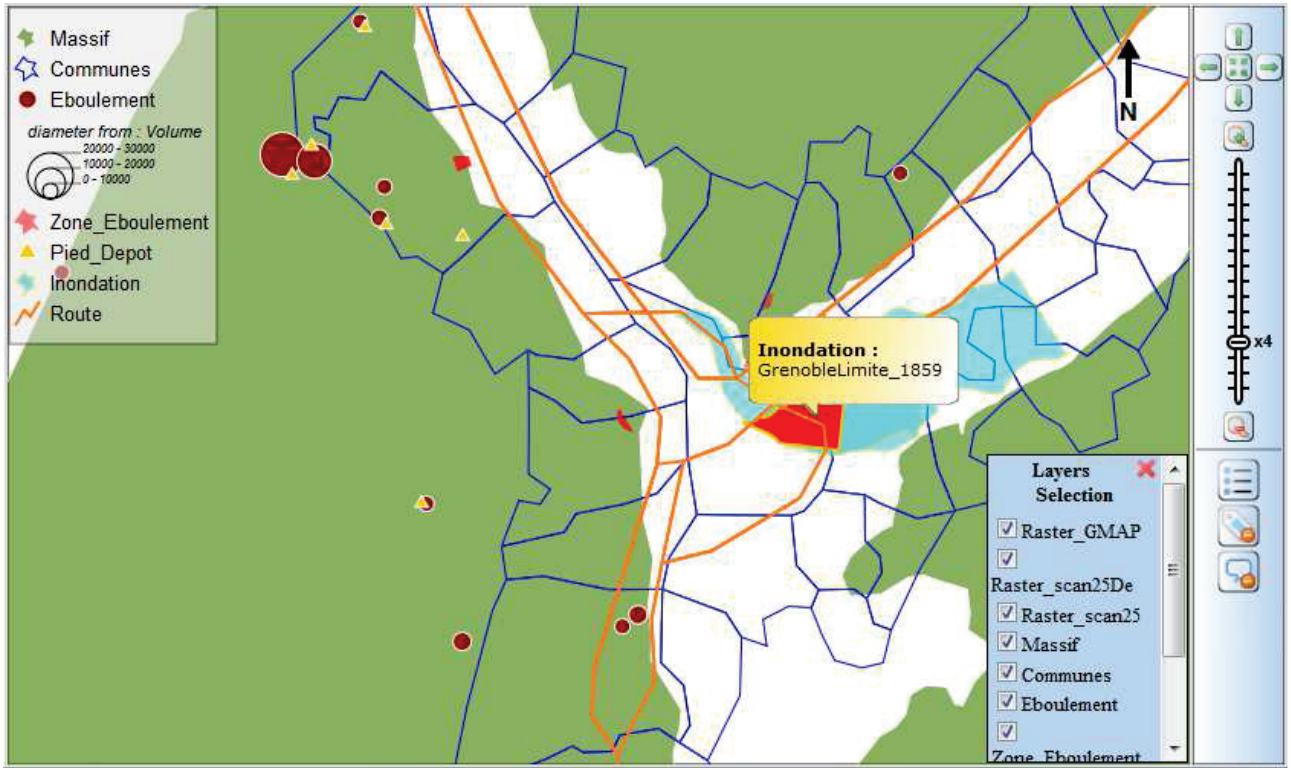


Figure 32 : Capture d'écran du volet spatial

Le volet cartographique affiche une légende (en haut à gauche sur la Figure 32) qui représente les différentes couches affichées, ainsi que leur nom et éventuellement leurs variations. Cette légende est déplaçable dans la fenêtre et son contenu est dynamique. En effet, le cadre *Layers Selection* (en bas à droite sur la Figure 32) permet d'afficher ou de cacher une couche. Lorsqu'une couche est masquée, la carte et la légende sont redéfinies et redessinées automatiquement. Les objets graphiques que la couche cachée contient n'apparaissent plus sur la carte et le contenu de la légende est modifié de manière à faire disparaître le symbole qui leur est associé.

Enfin, la partie cartographique possède une zone de contrôle (sur la droite de la Figure 32), qui regroupe un ensemble de commandes permettant la modification du facteur de zoom, ainsi que la navigation. Notons aussi la présence de trois boutons qui permettent de faire apparaître ou disparaître la légende, le sélectionneur de couches et les informations affichées sous forme de bulles.

3.3.2. Le volet temporel

Le volet temporel se présente sous la forme d'un graphique (cf. Figure 33) qui, à l'identique du volet spatial, est constitué d'un document SVG pour le dessin, et de documents HTML servant de squelette et intégrant les menus contextuels.

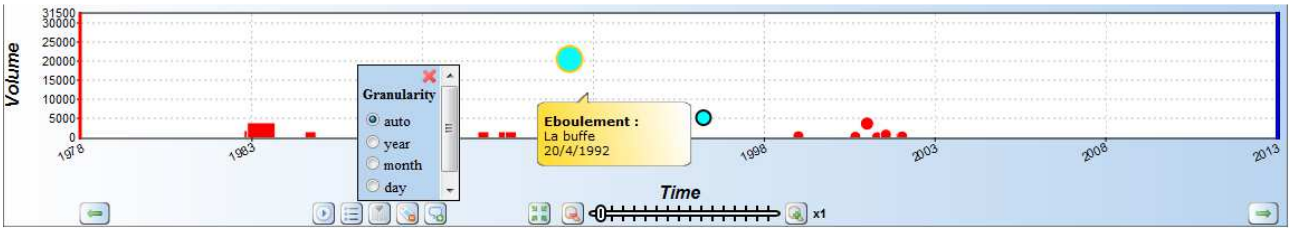
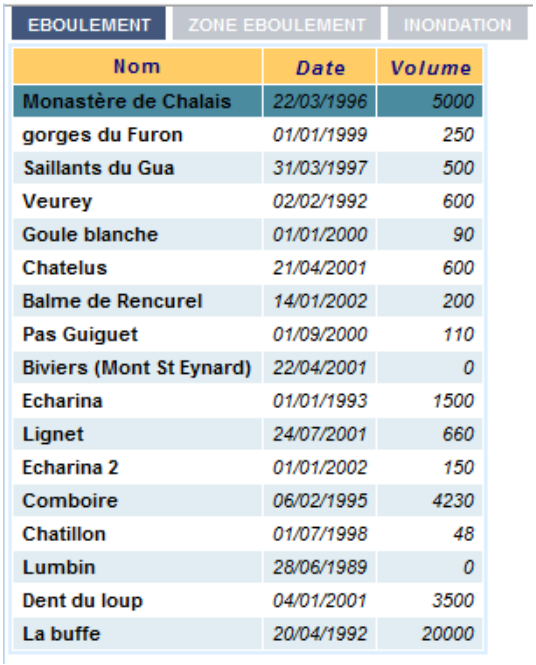


Figure 33 : Capture d'écran du volet temporel

Ce volet possède un repère orthogonal ayant pour axe des abscisses l'échelle du temps et pour axe des ordonnées un attribut du modèle de données sélectionné par le concepteur lorsqu'il a défini son modèle de présentations. L'axe du temps est, par défaut, géré dynamiquement par l'application qui en fonction des données, choisit la périodicité la mieux adaptée. Néanmoins, la fenêtre *Granularity* permet de fixer la granularité du temps parmi trois valeurs possibles: annuelle, mensuelle ou journalière. Une particularité intéressante pour restreindre le domaine d'étude est la possibilité de fixer des valeurs minimales et maximales à l'axe temporel (respectivement en rouge et en bleu sur la Figure 33).
Le volet temporel possède les mêmes fonctionnalités que le volet spatial en ce qui concerne le déplacement, le zoom, la légende, les informations bulle, et l'affichage des couches. A cela, s'ajoute une autre fonctionnalité : l'animation temporelle. Une ligne parcourt l'axe du temps du début à la fin et met en surbrillance les objets graphiques qu'elle croise sur son passage. Cette fonctionnalité permet de faire ressortir ces objets sur les autres fenêtres de l'application grâce à la synchronisation des styles.

3.3.3. Le volet attributaire

Le volet attributaire prend la forme d'un tableau à onglets. Chaque onglet correspond à l'une des classes du modèle de données que le concepteur a choisi d'afficher dans son SIST et contient une table de données où chaque colonne correspond aux différents attributs de cette classe. Ainsi, une ligne de cette table représente un objet instancié de la classe associée et affiche la valeur de ses différents attributs. Le tableau à onglets est constitué d'un document HTML maître. Chaque onglet est un autre document HTML qui décrit la table de données (cf. Figure 34).


EBOULEMENT | ZONE EBOULEMENT | INONDATION

Nom	Date	Volume
Monastère de Chalais	22/03/1996	5000
gorges du Furon	01/01/1999	250
Saillants du Gua	31/03/1997	500
Veurey	02/02/1992	600
Goule blanche	01/01/2000	90
Chatelus	21/04/2001	600
Balme de Rencurel	14/01/2002	200
Pas Guiguet	01/09/2000	110
Biviers (Mont St Eynard)	22/04/2001	0
Echarina	01/01/1993	1500
Lignet	24/07/2001	660
Echarina 2	01/01/2002	150
Comboire	06/02/1995	4230
Chatillon	01/07/1998	48
Lumbin	28/06/1989	0
Dent du loup	04/01/2001	3500
La buffe	20/04/1992	20000

Figure 34 : Capture d'écran du volet attributaire

Il est possible de trier les données en cliquant sur le nom des colonnes choisies pour indexer le tri. Enfin, la sélection unitaire (par exemple « Monastère de Chalais » dans la Figure 34) d'une ou plusieurs lignes dans la table de données déclenche la mise en surbrillance des objets graphiques sélectionnés dans l'ensemble des fenêtres du SIST.

3.3.4. Synchronisation entre les volets

Comme nous l'avons vu dans les sections précédentes consacrées aux volets d'une application SIST GenGHIS, une synchronisation s'opère entre l'ensemble des éléments visuels liés à une même entité. Dans le modèle de données, l'instance d'une classe peut correspondre à différents objets graphiques et ce, sur les trois composantes de représentation visuelle. Prenons par exemple une instance de type inondation (cf. Figure 35) : celle-ci va apparaître sur le volet attributaire sous sa forme de données (image 1), sur le volet spatial sous sa forme

de graphique SVG (version spatiale) (image 2), et sur le volet temporel sous sa forme de graphique SVG (version temporelle) (image 3).

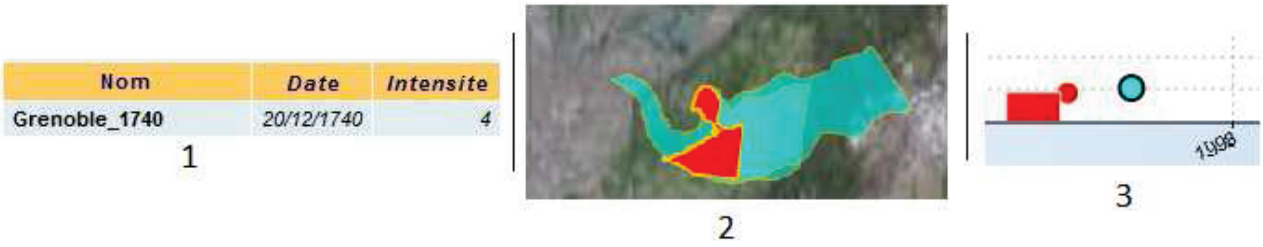


Figure 35 : Différentes représentations graphiques d'une instance d'une classe Inondation

Il est nécessaire de synchroniser l'état de l'ensemble des éléments visuels liés à une même entité(3). Cette synchronisation s'effectue sous deux formes distinctes :

- *la synchronisation des styles* : Elle permet qu'un objet ait le même type de style à travers les éléments visuels le caractérisant, à savoir : normal, accentué, survolé et accentué-survolé ;
- *la synchronisation de la visibilité* : Nous avons vu que l'utilisateur peut modifier l'espace d'étude à partir des volets spatiaux et temporels. Cette modification de l'espace d'étude doit être répercutée sur l'ensemble de l'application. Par exemple, si un zoom sur la carte a pour conséquence de faire disparaître de celle-ci une inondation, la représentation graphique de cette instance doit disparaître non seulement de la carte, mais aussi sur le graphique temporel et dans le tableau attributaire. Le SIST gère cette synchronisation en cachant automatiquement les objets sur l'ensemble des volets.

3.4. Synthèse

De cette analyse de l'existant, nous essayons de tirer un bilan qui donne une vision synthétique des différents constituants qui interviennent dans le processus de génération d'un SIST GenGHIS et de la manière dont ce processus est piloté par les différents modules de l'application GenGHIS. Cette vision synthétique est importante, car elle nous permet de définir un certain nombre de concepts qui n'avaient pas nécessairement été explicités par les concepteurs de GenGHIS, concepts sur lesquels nous pourrons ensuite appuyer nos propositions dans le chapitre suivant.

Comme nous le montre la Figure 36, l'application GenGHIS se compose de trois modules : le module de données, le module de présentations et le module générateur. Ces modules réalisent un certain nombre d'opérations mettant en œuvre différents constituants qui sont soit accessibles par l'utilisateur de l'application GenGHIS, soit intrinsèques au fonctionnement de cette dernière.

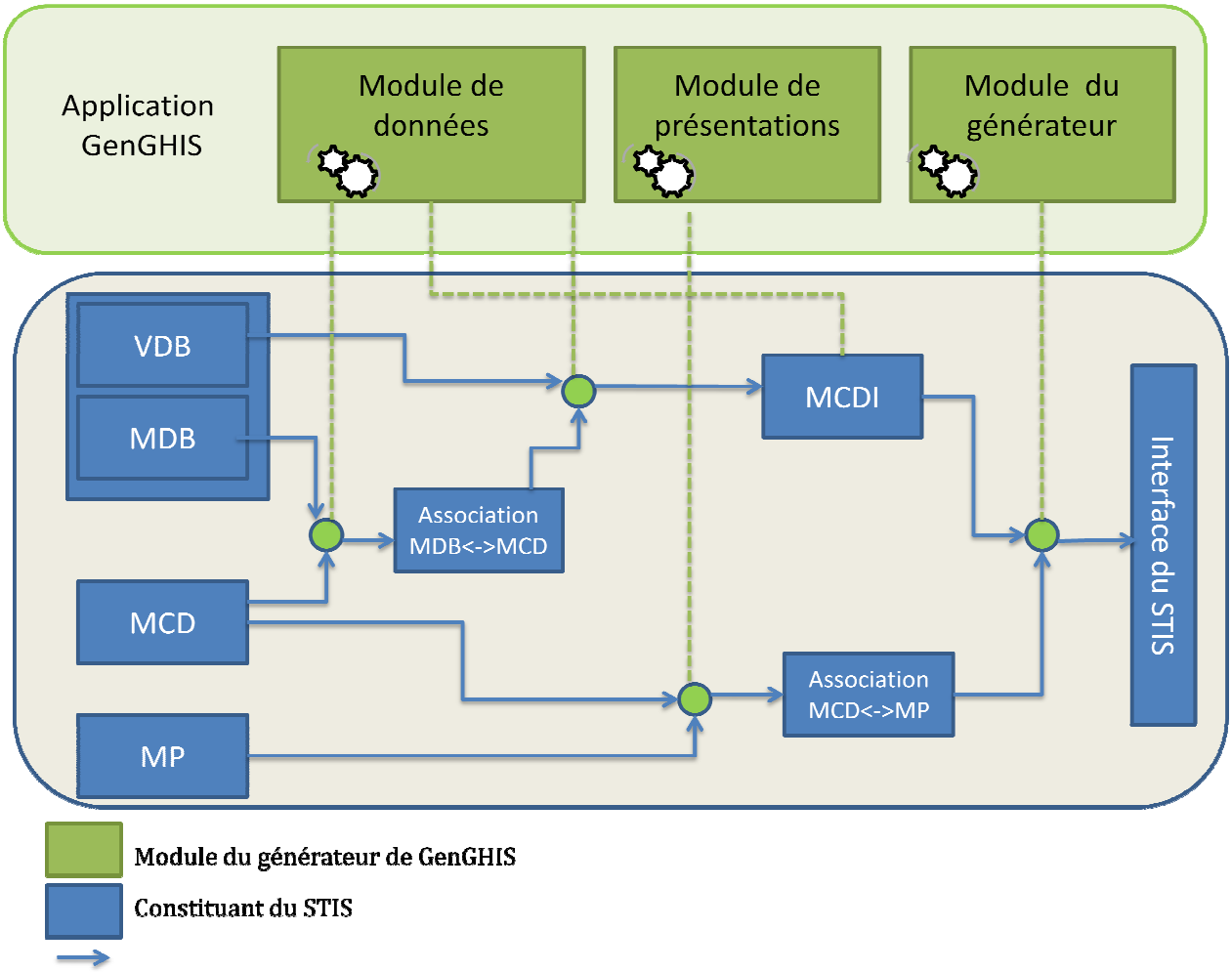


Figure 36 : Constituants d'un SIST

Les éléments de la Figure 36 sont détaillés dans les sections qui suivent.

3.4.1. Constituants d'un SIST GenGHIS

Le terme constituant d'un SIST GenGHIS désigne un ensemble de données qui servent d'entrée ou qui sont produites en sortie de l'une des opérations effectuées par l'un des modules de l'application GenGHIS.

Modèle Conceptuel de Données (*Conceptual Data Model- CDM*). Le Modèle Conceptuel de Données (MCD) définit un ensemble d'entités et de relations, décrivant un domaine d'application donné (par exemple les inondations). A une application SIST correspond un seul MCD ; par contre plusieurs applications SIST peuvent partager un même MCD. Dans la version actuelle de GENGHIS, ce modèle est décrit en AROM-ST.
La figure suivante (cf. Figure 37) présente l'extrait d'un Modèle Conceptuel de Données qui fait apparaître une classe inondation, elle-même composée d'un certain nombre de variables qui la définissent : « Hauteur_eau » de type décimal, un « nom » de type chaîne de caractères,…

```
class: Inondation
variables:
 variable: Hauteur_eau
  type: float
 variable: Nom
  type: string
 variable: Date
  type: date
 variable: Dommage_physiques
  type: string
 variable: Intensite
  type: integer
```

Figure 37 : Extrait d'un Modèle Conceptuel de Données

Données Brutes (*Raw Data*) : Les Données Brutes sont les données telles qu'elles sont fournies en entrée du système GenGHIS. Elles sont dans un format plus ou moins structuré (fichier texte, fichier excel, document MIF/MID).
Les données brutes peuvent elles-mêmes être séparées en deux constituants :

- **le Modèle de Données Brutes (*Raw Data Model- RDM*)** : le Modèle de Données Brutes (MDB) définit la structuration des données brutes (nom et type des colonnes d'un fichier Excel, nom et type des champs d'un fichier MIF/MID...) ;
- **VDB** (Raw Data Values - RDV) : les valeurs des données brutes.

Association MDB<->MCD : il s'agit d'un ensemble de relations définissant une correspondance entre le modèle de données brute (MDB) et le modèle conceptuel de données (MCD).

Modèle Conceptuel de Données Instancié (*Instanciated Conceptual Data Model- ICDM*) : Le Modèle Conceptuel de Données Instancié (MCDI) contient les entités de modélisation du MCD ainsi que leurs instances issues des données brutes. La figure suivante (cf. Figure 38) est un extrait d'un modèle conceptuel de données instancié.

```
instance: Inondation1495384865
is-a: Inondation
 Hauteur_eau = 0.0
 Nom = "GrenobleLimite_1859"
 Date = "01/11/1859,12:00:00"
 Dommage_physiques = "pertes : 400 000 francs or"
 Intensite = 2
```

Figure 38 : Extrait d'un Modèle Conceptuel de Données Instancié

«Inondation1495384865 » est l'instance de la classe Inondation (*is-a*) dont les valeurs sont : « Hauteur-eau = 0.0 », « Nom = "GrenobleLimite_1859" », etc.

Modèle de Présentations (*Presentation Model- PM*) : Un Modèle de Présentations (MP) contient un ensemble d'entités décrivant les propriétés graphiques d'une visualisation, sous forme de styles, règles et symboles. Il peut être générique ou être plus précis pour un domaine. Nous pouvons par exemple créer un style rivière et lui attribuer une couleur, une forme,… La figure suivante (cf. Figure 39) est un extrait d'un modèle de présentations faisant apparaitre une classe « SymbolSText », elle-même composée d'un certain nombre de variables qui la définissent : « style » de type entier, « font » de type chaîne de caractères,…

```
class: SymbolSText
super-class: SymbolSPoint
variables:
 variable: style
  type: integer
 variable: font
  type: string
 variable: align
  type: string
 variable: colorFont
  type: integer
 variable: sizeFont
  type: integer
```

Figure 39 : Extrait d'un Modèle de Présentations

Association MCD<->MP : Il s'agit d'un ensemble de relations décrivant la correspondance entre un MCD et un MP.

Interface du SIST (*STIS Graphical User Interface*) : A partir de l'association MCD<->MP et du modèle de données instancié, GenGHIS génère l'interface graphique pour la visualisation des données du Systèmes d'Information Spatio-Temporelle. Dans la version actuelle de GenGHIS l'interface du SIST est un ensemble de fichiers JavaScript, SVG, HTML structurés en une page HTML pouvant être chargée depuis n'importe quel navigateur Web.

3.4.2. Les opérations de l'application GenGHIS

Les constituants d'un SIST GenGHIS ayant été défini, nous pouvons maintenant présenter les différentes opérations effectuées par l'application GenGHIS en précisant pour chacune d'elle les constituants qu'elle nécessite en entrée et les constituants qu'elle produit en sortie.

Le module de données

Création de l'association MDB<->MCD :
Cette opération prend en entrée un modèle conceptuel de données (MCD) et un modèle de données brutes (MDB) permettant à l'utilisateur de construire l'association MDB<->MCD.
Il est demandé à l'utilisateur de faire correspondre les entités du modèle conceptuel avec les attributs définis dans le modèle de données brutes. Par exemple, pour un MCD concernant les inondations : la classe Commune contient les attributs Contour et Nom. Le fichier .xls de l'utilisateur contient les colonnes nom_ville et contour_ville. L'utilisateur devra faire correspondre l'attribut Contour avec la colonne contour_ville et l'attribut Nom avec la colonne nom_ville. Dans la version de l'application GenGHIS présentée dans ce chapitre, cette association est un composant interne, elle n'est ni conservée à la fin de l'utilisation de l'application, ni enregistrable par l'utilisateur. De même, le modèle de données brutes (MDB) n'est pas dissocié des valeurs des données brutes (VDB). L'utilisateur ne perçoit pas directement ces notions qui sont implicitement attachées au fichier de données qu'il importe dans l'application GenGHIS.

Instanciation du modèle de données :
Cette opération ne peut avoir lieu qu'après l'opération de création de l'association MDB<->MCD. Elle prend en entrée cette association MDB <-> MCD et un jeu de données (VDB) conforme au MDB, puis elle instancie (remplit) le modèle conceptuel des données à l'aide des valeurs des données brutes (VDB). Le résultat est un modèle conceptuel de données instancié (MCDI).

Modification du MCDI :
Cette opération permet à l'utilisateur de modifier le modèle conceptuel de données instancié (MCDI) en ajoutant, supprimant, modifiant des objets. L'entrée et la sortie de cette opération sont le MCDI.

Le module de présentation

Création de l'association MCD<->MP :
Cette opération prend en entrée un modèle conceptuel de données et un modèle de présentations. Il est demandé à l'utilisateur de faire correspondre les attributs de l'élément à instancier, avec les caractéristiques visuelles d'un objet (style). Lorsque l'utilisateur a terminé de travailler sur son modèle de présentations, GenGHIS enregistre à la suite de ce modèle cet ensemble de relations.

Le module du générateur

Génération du SIST :
Cette opération prend en entrée le modèle conceptuel des données instancié (MCDI) et l'association MCD<->MP pour générer l'interface graphique du SIST. Grâce à l'association MCD<->MP elle peut créer le code CSS, SVG et HTML qui permet de représenter les différents objets du MCDI.

Cette section conclut le chapitre trois portant sur l'étude de l'existant. Cette analyse a servi de base à la conception de l'application GenGHIS-Web, objet du prochain chapitre.

Chapitre 4. Conception

Dans le cadre d'un projet informatique, la conception regroupe les étapes dans lesquelles nous effectuons l'étude des données et l'étude des traitements à effectuer.
Dans un premier temps nous abordons les processus de développement. Ensuite, élément essentiel à tous les cahiers des charges, nous exprimons les besoins initiaux à partir desquels nous spécifions les exigences. Enfin, nous modélisons la navigation du futur portail web et nous abordons la notion d'ergonomie.

4.1. Processus de développement

Lors de l'élaboration d'un projet, il est d'usage de définir un processus de développement, c'est-à-dire une séquence d'étapes partiellement ordonnées, qui concourent à l'obtention d'un système logiciel(11). L'objet d'un processus de développement est de produire, dans des temps et des coûts prévisibles, des logiciels de qualité qui répondent aux besoins de leurs utilisateurs. Si l'on veut que le développement d'un logiciel aboutisse dans de bonnes conditions, il n'est pas possible de s'affranchir de ce type de processus. Ci-après, deux processus de développement sont résumés : les méthodes agiles (AM) et le processus unifié(UP).

- *Les Méthodes Agiles*

Les méthodes agiles prônent quatre valeurs fondamentales(12) :
-l'équipe : « Personnes et interactions plutôt que processus et outils » ;
-l'application : « Logiciel fonctionnel plutôt que documentation complète » ;
-la collaboration : « Collaboration avec le client plutôt que négociation de contrat » ;
-l'acceptation du changement : « Réagir au changement plutôt que suivre un plan ».
Il existe plusieurs méthodes agiles, par exemple : « *Extreme programming »* ou « scrum ».

- *Le Processus Unifié*

Le Processus Unifié (UP, pour *Unified Process*) est un processus de développement logiciel « itératif et incrémental, centré sur l'architecture, conduit par les cas d'utilisation et piloté par les risques ». Cette architecture est modélisée en utilisant le langage UML. Le Processus Unifié est organisé selon les quatre phases suivantes :

  - la phase d'initialisation conduit à définir la « vision » du projet ;
  - la phase d'élaboration permet d'identifier et de décrire la majeure partie des besoins des utilisateurs, de construire l'architecture de base et de lever les risques majeurs ;
  - la phase de construction consiste à concevoir et implémenter l'ensemble des éléments opérationnels ;
  - enfin, la phase de transition permet de transmettre le système informatique des développeurs aux utilisateurs finaux.

De par le contexte particulier de ce projet mené au sein d'une équipe de recherche, nous avons pris la décision de ne pas suivre un processus de développement spécifique. Le projet a été conduit en suivant un processus reprenant à la fois des préconisations des Méthodes Agiles (MA) et du Processus Unifié (PU) mais de manière simplifiée. Pour les tâches de conception nous nous sommes appuyés sur l'utilisation d'un sous-ensemble nécessaire et suffisant du langage UML.

4.2. Expression initiale des besoins

La première phase du développement est de collecter, d'analyser et de définir les besoins de haut niveau et les caractéristiques du futur portail GenGHIS-Web. Cette section est donc axée sur les fonctionnalités requises par les utilisateurs, et sur la raison d'être de ces exigences. Elle décrit le rôle du système (ce qu'il doit faire) et la manière dont l'utilisateur peut exécuter les fonctions du système.

Pour observer une similitude avec le Processus Unifié, cette partie du travail correspond au travail réalisé lors de la phase d'initialisation.

4.2.1. Besoins fonctionnels

L'objectif de GenGHIS-Web tel qu'il a été défini est la mise en œuvre d'un portail permettant à un utilisateur d'accéder, à partir d'un navigateur web quelconque, à des SIST GenGHIS et éventuellement de créer, modifier et partager ses propres SIST.

La création des SIST : GenGHIS-Web doit permettre à un utilisateur de créer de nouveaux SIST GenGHIS. Nous conservons les étapes de l'application originelle, à savoir le choix d'un MDC, d'un MP, et leur instanciation. Pour l'identification et le partage des SIST, il est nécessaire de leur donner un nom unique, ainsi qu'une brève description.

La modification des SIST : l'utilisateur pourra revenir sur la conception d'un SIST. Il retrouvera toutes les étapes et informations qu'il a laissées durant la création du SIST, notamment les différentes associations MCD<->MDB et MCD<->MP.

La visualisation des SIST : c'est la finalité de leur conception. L'utilisateur pourra visualiser et interagir avec des SIST : ses propres SIST qu'il a créé lui-même ou des SIST construits par d'autres utilisateurs et que ceux-ci ont décidé de partager avec lui.

Le partage des SIST : le créateur d'un SIST peut rendre celui-ci accessible à d'autres utilisateurs. Le partage peut se faire avec tous les utilisateurs potentiels de GenGHIS-Web ou de manière plus restreinte avec un groupe d'utilisateurs privilégiés. Par ailleurs, il est possible de partager uniquement l'interface de visualisation du SIST sans possibilité de modification ou bien de partager l'ensemble des constituants du SIST.

De ces besoins de partage, on peut en déduire deux caractéristiques associées au SIST :

- Une visibilité qui définit quels utilisateurs ont accès au SIST en consultation. Les différentes visibilités possibles sont
  - ***publique*** : l'interface de visualisation du SIST est accessible à tous les utilisateurs de GenGHIS-Web ;
  - ***semi-publique*** : une description du SIST est accessible à tous les utilisateurs de GenGHIS-Web, mais l'accès à son interface de visualisation est soumise à l'approbation du propriétaire du SIST ;
  - ***privée*** : l'interface de visualisation du SIST n'est accessible, après authentification, qu'à des utilisateurs sélectionnés par le propriétaire du SIST.

- Une liste des utilisateurs ayant un accès privilégié. Cet accès peut être de trois natures :
  - en ***lecture-seule*** : l'utilisateur a accès uniquement à l'interface de visualisation du SIST ;
  - en ***lecture-copie*** : l'utilisateur a accès à l'interface de visualisation du SIST mais aussi aux composants qui le caractérisent et qu'il pourra réutiliser pour créer ses propres SIST ;
  - ***propriétaire*** : l'utilisateur est le créateur du SIST, il peut le modifier et le supprimer. Il gère aussi les caractéristiques d'accès et la visibilité.

4.2.2. Besoins non fonctionnels

Les besoins non fonctionnels couvrent différents aspects décrits ci-dessous : ergonomie, utilisabilité, maintenance et confidentialité.

Une **ergonomie** sobre et efficace : les parties 10 à 17 de la norme ISO 9241 qui traitent de l'ergonomie du logiciel, permettent d'avoir une vue globale sur les principes à mettre en place lors de la phase de spécification et de développement.

Dans un contexte d'utilisation spécifié, **l'utilisabilité** est définie par la norme ISO 9241-11 comme « le degré selon lequel un produit peut être utilisé, par des utilisateurs identifiés, pour atteindre des buts définis avec efficacité, efficience et satisfaction ». Rappelons que GenGHIS-

Web s'adresse à un public non nécessairement informaticien, l'utilisabilité de l'application est donc un critère important. Pour les mêmes raisons, le guidage de l'utilisateur sera essentiel. La mise en page du site facilitera au maximum la démarche, à l'aide d'une présentation claire et intuitive.

La facilité de **maintenance** : le projet sera amené à évoluer dans le temps. Sa pérennité sera assurée via une maintenance évolutive qui permettra de faire évoluer l'application en l'enrichissant de fonctions ou de modules supplémentaires.

La **confidentialité** : bien que la tendance soit à la diffusion des données géographiques, certaines n'en demeurent pas moins privées. L'application devra s'assurer que l'information ne soit accessible qu'à ceux dont l'accès est autorisé.

4.2.3. Contraintes de conception

Nous nous sommes fixés les contraintes de conception suivantes :

- *le stockage des données* : Les SIST doivent être sauvegardés de façon fiable et durable. Le stockage doit être fait sur un système de gestion de bases de données (SGBD) ;

- *le langage de programmation* : La version de GenGHIS sur laquelle nous nous basons a été conçue avec le langage de programmation Java. Le portail internet devra assurer l'interopérabilité entre les composants logiciels récupérés de l'application GenGHIS et sa nouvelle interface ;

- *la modularité de l'application* : La version actuelle de l'application GenGHIS est étroitement liée à AROM. Les modules de données et de présentation s'appuient directement sur AROM-API. AROM n'étant plus maintenu, il sera à terme nécessaire de le remplacer par un autre système de modélisation pour représenter les modèles de données et de présentations. C'est pourquoi, nous souhaitons rendre le code de l'interface graphique de ces modules le plus indépendant possible de l'API AROM.

4.3. Spécification des exigences

L'expression préliminaire des besoins donne lieu à une modélisation par des cas d'utilisation. Dans cette section, nous identifions en premier lieu les acteurs de l'application, ensuite, nous identifions et construisons les cas d'utilisation.
Cette partie du travail est effectuée lors de la phase d'élaboration du Processus Unifié.

4.3.1. Identification des acteurs

En premier lieu, il convient de définir une typologie des utilisateurs de GenGHIS-Web.

Un utilisateur ou acteur représente un rôle joué par une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système étudié.

Différents types d'utilisateurs peuvent être distingués :

- **Utilisateur « visiteur »** : Un utilisateur de cette nature est un internaute quelconque, qui peut se connecter au portail GenGHIS-WEB et accéder à sa partie publique. L'application GenGHIS-Web met, à sa disposition l'interface graphique d'un certain nombre de SIST (en consultation) avec laquelle il pourra interagir sans possibilité de modification de celle-ci, ni des données qu'elle affiche. Le visiteur pourra consulter la description d'un certain nombre de SIST à accès restreint et effectuer une demande aux propriétaires pour y avoir accès.

- **Utilisateur « avec privilèges »** : Un utilisateur de cette nature a, en plus des droits de l'utilisateur "visiteur", accès à un certain nombre de SIST dont la consultation est restreinte. Ce type d'utilisateur doit s'authentifier auprès du système.

- **Utilisateur « concepteur »** : En plus de la consultation de SIST à accès restreint, ce type d'utilisateur a la possibilité d'interagir avec certains des modules du Générateur GenGHIS pour créer/modifier des SIST. Il est "propriétaire" de ces SIST : il peut autoriser leur accès à d'autres utilisateurs et il peut également les supprimer. Ce type d'utilisateur doit s'authentifier auprès du système.

- **Administrateur** : Un utilisateur "administrateur" peut gérer les droits d'accès de tous les autres utilisateurs. De même, l'administrateur a accès à tous les SIST et possède les mêmes droits que leur propriétaire. Ce type d'utilisateur doit s'authentifier auprès du système.

Il ne faut pas confondre rôles et entité concrète. Une même entité concrète peut jouer successivement différents rôles par rapport au système étudié. Dans notre cas, les droits des utilisateurs héritent d'une hiérarchie ascendante, comme le montre la figure suivante (cf. Figure 40) :

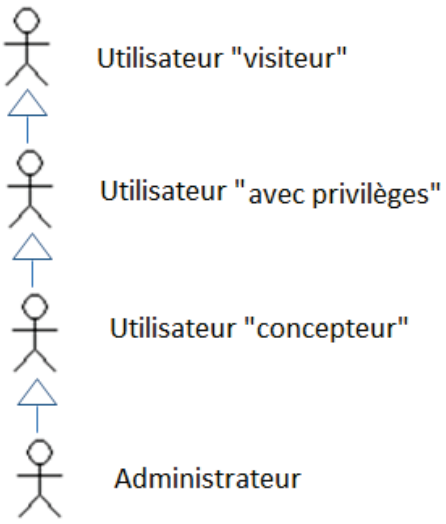


Figure 40 : Typologie des utilisateurs de GenGHIS-Web

4.3.2. Spécification des exigences d'après les cas d'utilisation

Pour chaque acteur identifié précédemment, il convient de rechercher les différentes intentions « métier » selon lesquelles il utilise le système, c'est le rôle des cas d'utilisation. Ils représentent un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable intéressant pour un acteur en particulier(13). Un cas d'utilisation modélise un service rendu par le système. Il exprime les interactions acteurs/système.
Il est nécessaire de rappeler que les cas d'utilisation des utilisateurs ascendants s'appliquent aux utilisateurs de niveau inférieur. En plus des diagrammes de cas d'utilisation, nous formulons une description textuelle de ceux-ci. Cette description textuelle n'étant pas normalisée par UML, nous en proposons une sous forme de tableaux. Pour chaque cas d'utilisation d'un acteur donné un tableau présente : l'objectif du cas d'utilisation et le scénario nominal, mais aussi, s'il y a lieu, les préconditions et les postconditions.

4.3.2.1. Cas d'utilisation de l'utilisateur « visiteur »

L'expression de besoins préliminaires a mis en évidence trois cas d'utilisation, pour l'utilisateur « visiteur » représenté sur la Figure 41.

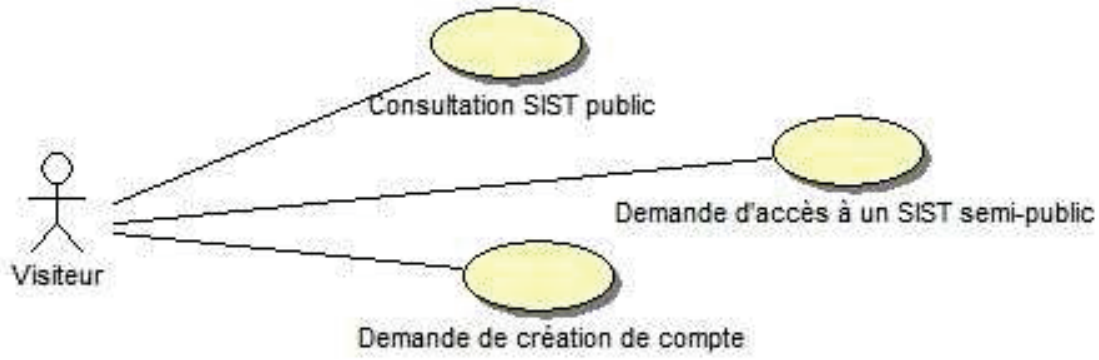


Figure 41 : Cas d'utilisation du visiteur

Les tableaux suivants fournissent une description plus complète des cas d'utilisation pour l'utilisateur *Visiteur*.

- Consultation d'un SIST public (cf. Tableau 2) :

Consultation des SIST publics	
Objectif	L'utilisateur consulte librement l'interface de visualisation d'un SIST accessible à tout le monde.
Précondition(s)	Au moins un SIST doit avoir pour visibilité : publique.
Postcondition(s)	Aucune

Scénario nominal	1. L'utilisateur accède à la liste des SIST publics. 2. L'utilisateur choisit un SIST parmi ceux-ci. 3. L'utilisateur accède à l'interface de visualisation du SIST choisi.

Tableau 2 : Description contextuelle du cas d'utilisation consultation des SIST publics

- Demande d'accès à un SIST semi-public (cf. Tableau 3) :

Demande d'accès à un SIST semi-public	
Objectif	L'utilisateur demande au propriétaire d'un SIST, disponible seulement à une liste restreinte d'utilisateurs, un accès sur celui-ci.
Précondition(s)	Au moins un SIST doit avoir pour visibilité : semi-publique.
Postcondition(s)	La demande d'accès à un SIST doit être expédiée.
Scénario nominal	1. L'utilisateur parcourt la liste de tous les SIST à accès restreint. 2. L'utilisateur choisit un SIST. 3. L'utilisateur choisit le type d'accès qu'il souhaite obtenir : lecture-seule ou lecture-copie. 4. L'utilisateur fournit un certain nombre de renseignements le concernant (certains seront obligatoires dont, en particulier, son adresse email). 5. L'utilisateur soumet sa demande via l'interface (celle-ci est transmise pour validation au propriétaire du SIST concerné).

Tableau 3 : Description contextuelle du cas d'utilisation demande d'accès à un SIST semi-public

- Demande de création de compte (cf. Tableau 4) :

Demande de création de compte	
Objectif	L'utilisateur souhaite accéder aux autres fonctionnalités de l'application, pour cela il doit s'enregistrer auprès du portail.
Précondition(s)	aucune
Postcondition(s)	Une demande de création de compte doit être envoyée.
Scénario nominal	1. L'utilisateur fournit un certain nombre de renseignements le concernant (certains seront obligatoires dont son adresse email). 2. L'utilisateur choisit s'il souhaite avoir le profil utilisateur « avec privilèges » ou utilisateur « concepteur ». 3. L'utilisateur soumet sa demande via l'interface (celle-ci est transmise pour validation à l'administrateur GenGHIS-Web).

Tableau 4 : Description contextuelle du cas d'utilisation demande de création de compte

4.3.2.2. Cas d'utilisation de l'utilisateur « avec privilèges»

Suite à une réponse favorable de demande d'accès à un SIST semi-public ou à la création d'un compte, l'internaute devient un visiteur « avec privilèges ». Nous avons mis en évidence quatre cas d'utilisation représentés sur la Figure 42.

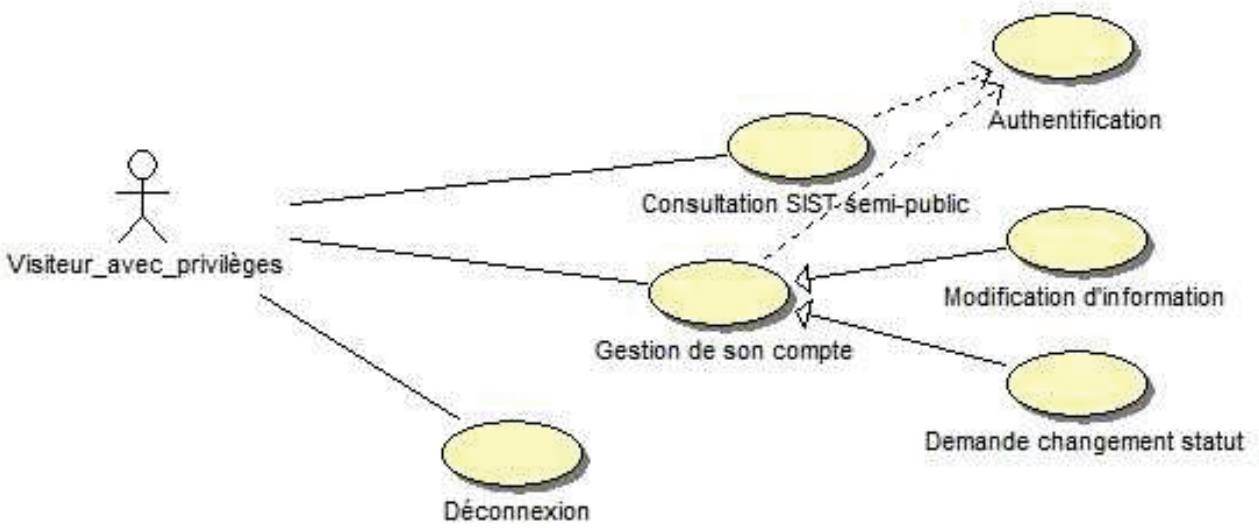


Figure 42 : Cas d'utilisation du visiteur avec privilèges

Les tableaux suivants fournissent une description plus complète des cas d'utilisation pour l'utilisateur « avec privilèges ».

- L'authentification (cf. Tableau 5) :

Authentification	
Objectif	L'utilisateur s'authentifie auprès du système afin de pouvoir accéder aux SIST "semi-publics" du portail GenGHIS-Web pour lesquels il est autorisé.
Précondition(s)	Le système a validé l'inscription.
Postcondition(s)	L'authentification a réussi et l'utilisateur a accès aux fonctionnalités réservées aux utilisateurs privilégiés
Scénario nominal	1. L'utilisateur accède à la page d'authentification 2. L'utilisateur renseigne son nom d'utilisateur et son mot de passe. 3. Le système vérifie les informations d'authentification et donne à l'utilisateur accès aux fonctionnalités réservées aux utilisateurs privilégiés.
Alternative	3.a Les informations d'authentification sont incorrectes, le système le notifie à l'utilisateur.

Tableau 5 : Description contextuelle du cas d'utilisation authentification

- La consultation de SIST semi-public (cf. Tableau 6) :

Consultation d'un SIST semi-public	
Objectif	L'utilisateur veut accéder à l'interface de visualisation d'un SIST « privé » ou semi-public auquel il a accès.
Précondition(s)	L'utilisateur est authentifié.
Postcondition(s)	Aucune
Scénario nominal	1. L'utilisateur accède à la liste des SIST "semi-publics" sur lesquels il a l'autorisation d'accès. 2. L'utilisateur sélectionne l'un de ces SIST. 3. L'utilisateur accède à l'interface de visualisation du SIST choisi.

Tableau 6 : Description contextuelle du cas d'utilisation consultation d'un SIST semi-public

Pour les deux derniers cas d'utilisation, nous n'avons pas jugé nécessaire de les présenter de manière détaillée dans un tableau. Nous nous contenterons d'une rapide description textuelle.

- La gestion de son compte

Ce cas d'utilisation regroupe les différents cas d'utilisation liés à la gestion du compte d'un utilisateur :

  - la demande de changement de statut : à partir de la gestion de son compte, l'utilisateur peut effectuer une demande pour la modification de son statut qui sera adressée à l'administrateur GenGHIS-Web, de façon à devenir concepteur.
  - la modification d'informations personnelles : l'utilisateur peut modifier les informations le concernant : email, mot de passe, nom,…

- La déconnexion

L'utilisateur se déconnecte, il n'a ensuite accès à GenGHIS-Web qu'à travers les fonctionnalités dédiées à l'utilisateur « visiteur ».

4.3.2.3. Cas d'utilisation de l'utilisateur « concepteur»

L'utilisateur de type « concepteur » est celui qui a accès aux fonctionnalités les plus complexes du système (création/modification d'un SIST). L'expression de besoins préliminaires a mis en évidence huit cas d'utilisation représentés sur la Figure 43.

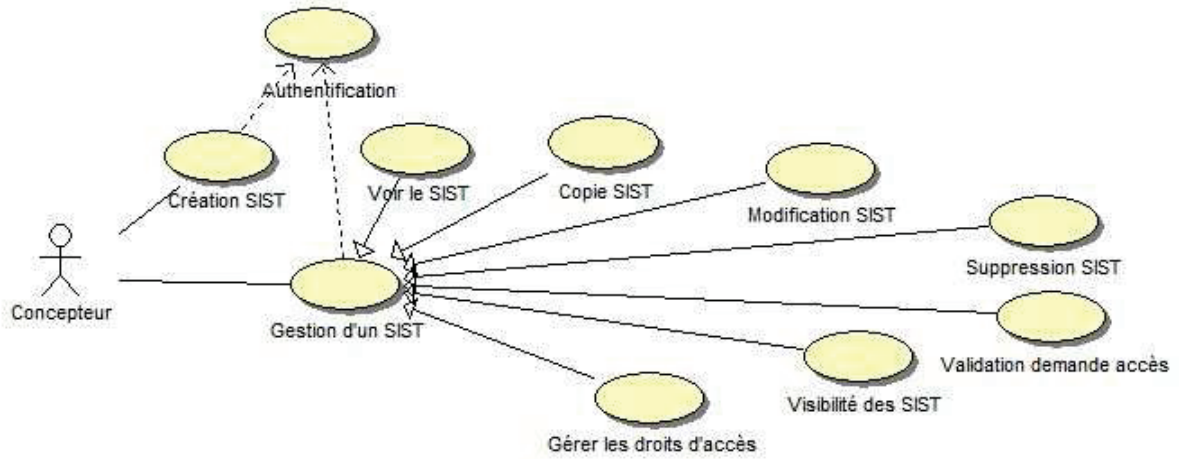


Figure 43 : Cas d'utilisation du concepteur

Les tableaux suivants fournissent une description plus complète des cas d'utilisation pour l'utilisateur « concepteur ».

- La création d'un SIST (cf. Tableau 7) :

Création d'un SIST	
Objectif	L'utilisateur veut générer un nouveau SIST.
Précondition(s)	L'utilisateur est authentifié.
Postcondition(s)	Aucune.
Scénario nominal	1 L'utilisateur crée un nouveau SIST et lui attribue un nom. 2 L'utilisateur fournit une description de son SIST. *Étapes pré-requises : 1* 3 Choix d'un MCD : l'utilisateur choisit un MCD parmi un ensemble de MCD disponibles sur le serveur. *Étapes pré-requises : 1* 4 L'utilisateur choisit des données brutes. Il les importe depuis son poste. *Étapes pré-requises : 1* 5 Création de l'association MCD<-->MDB : l'utilisateur sélectionne des entités du MCD et leur associe une entité du MDB. *Étapes pré-requises : 1, 3 et 4* 6 Choix d'un MP : l'utilisateur choisit un MP parmi un ensemble de MP disponibles sur le serveur. *Étapes pré-requises : 1* 7 Création de l'association MCD<-->MP : l'utilisateur fait correspondre les attributs de l'élément à instancier avec les caractéristiques visuelles d'un objet. *Étapes pré-requises : 1, 3 et 6* 8 Génération de l'interface du SIST : l'utilisateur génère l'interface graphique du SIST qui est sauvegardé. *Étapes pré-requises : 1, 3, 4, 5, 6 et 7*
Alternatives	3.a. L'utilisateur importe un MCD depuis son poste.

	4.a. L'utilisateur utilise un fichier de données brutes présent sur le serveur. 6.a. L'utilisateur importe un MP depuis son poste.
Exigence(s) supplémentaire(s)	Ces étapes sont présentées ici de manière successive et sont numérotées en conséquence. Cette succession correspond à un déroulement « logique » des opérations nécessaires à la création d'un SIST, mais leur ordre de réalisation (pour les étapes 2 à 7) n'est pas contraint par le système. L'utilisateur peut effectuer n'importe quelle étape dans n'importe quel ordre, à partir du moment où les étapes pré-requises ont été effectuées. De même, si l'utilisateur revient sur l'une de ces étapes, les étapes suivantes qui dépendent de celle-ci seront invalidées et l'utilisateur devra les recommencer. Pour permettre de comprendre les dépendances entre ces étapes, pour chacune d'entre elles, nous précisons celles qui sont pré-requises.

Tableau 7 : Description contextuelle du cas d'utilisation création d'un SIST

- La gestion d'un SIST :

Le cas d'utilisation « gestion d'un SIST » est une généralisation des cas énumérés ci-dessous. Sa présence est due au fait que l'utilisateur doit sélectionner parmi la liste des SIST dont il est propriétaire, ou pour lesquels il a un accès privilégié en "lecture/copie", le SIST sur lequel il souhaite travailler. Le SIST sélectionné devient le « SIST de travail ».

- Visualiser un SIST

L'utilisateur consulte l'interface de visualisation du « SIST de travail ».

- Copie d'un SIST (cf. Tableau 8) :

Copie d'un SIST	
Objectif	L'utilisateur veut copier un SIST. Si l'utilisateur en est le propriétaire, cela lui permettra ensuite de travailler sur la copie du SIST sans altérer le contenu du SIST original. Si l'utilisateur n'est autorisé qu'en copie, cela lui permet de devenir le propriétaire du SIST copié et donc par conséquent, s'il le souhaite, le modifier par la suite.
Précondition(s)	La copie d'un SIST est permise uniquement si l'utilisateur est le propriétaire du SIST, ou s'il a obtenu l'autorisation de celui-ci. Un "SIST de travail" a été sélectionné.
Postcondition(s)	Aucune.
Scénario nominal	1 L'utilisateur nomme le nouveau SIST. 2 Toutes les caractéristiques du SIST de travail sont recopiées dans le nouveau SIST, excepté la visibilité (le nouveau SIST est par défaut privé) et la liste des utilisateurs privilégiés (qui est vide).

Tableau 8 : Description contextuelle du cas d'utilisation copie d'un SIST

- Modification d'un SIST (cf. Tableau 9) :

Modification d'un SIST	
Objectif	L'utilisateur souhaite modifier un SIST dont il est le propriétaire.
Précondition(s)	Un "SIST de travail" a été sélectionné et l'utilisateur est le propriétaire de ce SIST.
Postcondition(s)	Aucune.
Scénario nominal	1 L'utilisateur modifie la description de son SIST. 2 Modification des données brutes : L'utilisateur sélectionne un jeu de données sur sa machine, ou sur le serveur, qui est conforme au MDB du SIST (toute entité du MDB du SIST est présente dans le MDB du jeu de données) 3 Modification de l'association MCD<->MP. L'utilisateur modifie les relations décrivant les propriétés graphiques du SIST. Il peut s'il le souhaite choisir un autre MP. 4 Re-Génération de l'interface du SIST : L'étape 2 ou l'étape 3 doit avoir été effectuée en préalable de l'étape 4. L'utilisateur génère l'interface graphique du SIST qui est sauvegardée.
Alternatives	1.a. L'utilisateur supprime la description de son SIST.
Exigence(s) supplémentaire(s)	Il n'est pas prévu dans le cas d'utilisation de modification d'un SIST, de permettre des modifications ayant trait au MCD et MDB (changement du MCD, changement du MDB, changement du mapping MDB-->MCD). Etant donné que l'une de ces modifications oblige à refaire tous le processus de création d'un SIST, nous ne le considérons pas comme une modification mais comme une création. Les étapes 1, 2, 3 sont indépendantes et peuvent être effectuées dans n'importe quel ordre, voire ne pas être effectuées du tout.

Tableau 9 : Description contextuelle du cas d'utilisation modification d'un SIST

- Suppression SIST (cf. Tableau 10) :

Suppression d'un SIST	
Objectif	L'utilisateur souhaite supprimer définitivement un SIST.
Précondition(s)	Un « SIST de travail » a été sélectionné et l'utilisateur est le propriétaire de ce SIST.
Postcondition(s)	Toutes les données correspondant au SIST sont supprimées du serveur.
Scénario nominal	1 L'utilisateur confirme la suppression. Cette suppression sera notifiée à tous les utilisateurs privilégiés et concepteurs qui avaient un accès à ce SIST.

Tableau 10 : Description contextuelle du cas d'utilisation suppression d'un SIST

- Demande d'accès (cf. Tableau 11) :

Validation demande d'accès	
Objectif	Les utilisateurs « visiteur » peuvent effectuer des demandes d'accès à un SIST semi-public (voir cas d'utilisation du « visiteur »). Les propriétaires de ces SIST doivent valider ou rejeter ces demandes.
Précondition(s)	Un « SIST de travail » a été sélectionné et l'utilisateur est le propriétaire de ce SIST.
Postcondition(s)	Une notification de validation ou de rejet est envoyée au demandeur.
Scénario nominal	1 L'utilisateur sélectionne la(les) demande(s) qu'il souhaite valider / rejeter. 2 L'utilisateur valide la(les) demande(s).
Alternatives	2. a. L'utilisateur rejette la(les) demande(s).

Tableau 11 : Description contextuelle du cas d'utilisation validation demande d'accès

- Visibilité du SIST (cf. Tableau 12) :

Modification de la visibilité du SIST	
Objectif	L'utilisateur souhaite modifier la visibilité d'un SIST dont il est le propriétaire afin de le rendre accessible à d'autres utilisateurs ou à l'inverse, de restreindre son accessibilité.
Précondition(s)	Un « SIST de travail » a été sélectionné et l'utilisateur est le propriétaire de ce SIST.
Postcondition(s)	Aucune.
Scénario nominal	1 L'utilisateur modifie le type de visibilité du SIST : public, semi-public (l'utilisateur doit, dans ce cas, avoir rempli au préalable la description de son SIST), privé.

Tableau 12 : Description contextuelle du cas d'utilisation modification de la visibilité

- Droit d'accès (cf. Tableau 13) :

Gérer les droits d'accès au SIST	
Objectif	Le propriétaire d'un SIST peut, de sa propre initiative, accorder ou supprimer des autorisations sur ses SIST.
Précondition(s)	Un « SIST de travail » a été sélectionné et l'utilisateur est le propriétaire de ce SIST.
Postcondition(s)	Une notification est envoyée au(x) utilisateur(s) impactés par les

	modifications faites par le propriétaire.
Scénario nominal	Ajout d'un nouvel utilisateur privilégié. 1 L'utilisateur sélectionne un utilisateur dans la liste des utilisateurs enregistrés sur GenGHIS-Web (et qui ne sont pas déjà autorisés sur le SIST de travail). 2 L'utilisateur attribue un droit de lecture ou de lecture/copie, avec ou sans data à l'utilisateur sélectionné. 3 L'utilisateur valide sa modification.
Alternatives	Modification des droits d'un utilisateur. 1 L'utilisateur sélectionne l'un des utilisateurs privilégiés du SIST. 2 L'utilisateur modifie les droits d'accès pour cet utilisateur : lecture, lecture/copie, ou aucun droit. 3 L'utilisateur valide sa modification.

Tableau 13 : Description contextuelle du cas d'utilisation de la modification des droits d'accès

4.3.2.4. Cas d'utilisation de l'administrateur

Concernant un administrateur du portail GenGHIS-Web, nous avons mis en évidence cinq cas d'utilisation représentés sur la Figure 44.

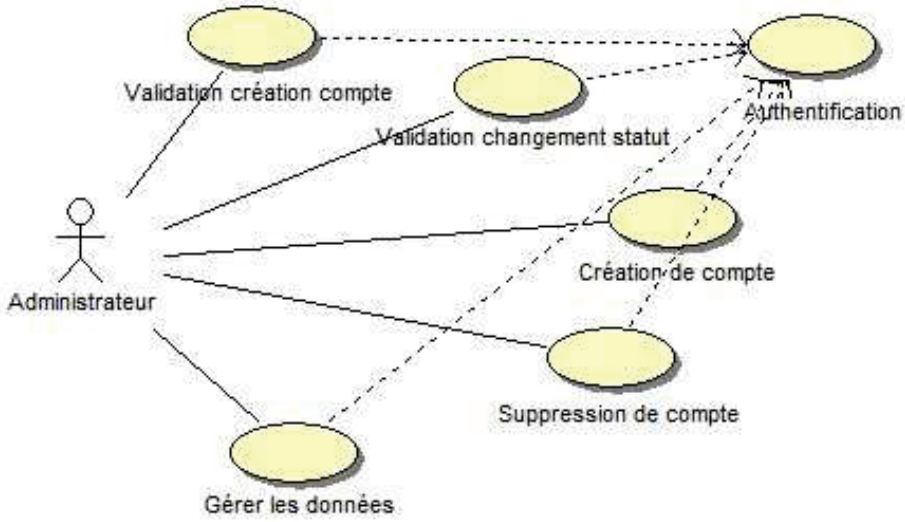


Figure 44 : Cas d'utilisation de l'administrateur

Les tableaux suivants fournissent une description plus complète des cas d'utilisation pour l'utilisateur « administrateur ».

- La validation d'une création de compte (cf. Tableau 14) :

Validation création de compte	
Objectif	L'administrateur est chargé de valider ou rejeter les demandes de création de compte faites par les utilisateurs « visiteurs ».

Précondition(s)	Au moins une demande a été formulée
Postcondition(s)	Une notification est envoyée aux utilisateurs impactés par la validation ou le rejet d'une demande de création de compte.
Scénario nominal	1. L'administrateur accède à la liste des demandes de création de compte. 2. L'administrateur sélectionne la(les) demande(s) qu'il souhaite valider / rejeter. 3. L'administrateur valide la(les) demande(s).
Alternatives	3.a. L'administrateur rejette la(les) demande(s).

Tableau 14 : Description contextuelle du cas d'utilisation validation création de compte

- La validation de changement de statut (cf. Tableau 15) :

Validation changement de statut	
Objectif	L'administrateur a en charge de gérer les demandes de changement de statut, qu'ont pu faire des utilisateurs à accès restreint, pour obtenir le statut de concepteur.
Précondition(s)	Au moins une demande a été formulée
Postcondition(s)	Une notification est envoyée aux utilisateurs impactés par la validation ou le rejet d'une demande de changement de statut.
Scénario nominal	1 L'administrateur accède à la liste des demandes de changement de statut. 2 L'administrateur sélectionne la(les) demande(s) qu'il souhaite valider / rejeter. 3 L'administrateur valide la(les) demande(s).
Alternatives	3.a. L'administrateur rejette la(les) demande(s).

Tableau 15 : Description contextuelle du cas d'utilisation validation changement de statut

- La création de compte (cf. Tableau 16) :

Création de compte	
Objectif	L'administrateur peut de son propre chef créer un compte qui sera immédiatement actif.
Précondition(s)	Aucune
Postcondition(s)	Une notification est envoyée au(x) utilisateur(s) impactés par la création de compte.
Scénario nominal	1 L'administrateur fournit un certain nombre de renseignements concernant le compte à créer.

	2 L'administrateur choisit le type de compte : visiteur, restreint, concepteur. 3 L'administrateur confirme la création du compte.

Tableau 16 : Description contextuelle du cas d'utilisation création de compte

- La suppression de compte (cf. Tableau 17) :

Suppression de compte	
Objectif	L'administrateur peut supprimer un compte
Précondition(s)	Aucune
Postcondition(s)	Une notification est envoyée à l'utilisateur dont le compte a été supprimé.
Scénario nominal	1 L'administrateur accède à la liste de tous les utilisateurs. 2 L'administrateur sélectionne l'identifiant du compte à supprimer. 3 L'administrateur confirme la suppression.
Exigence(s) supplémentaire(s)	Les SIST de l'utilisateur supprimé ne sont pas effacés.

Tableau 17 : Description contextuelle du cas d'utilisation suppression de compte

- Gérer les données (cf. Tableau 18) :

Gérer les données	
Objectif	L'administrateur peut ajouter et supprimer des MP et des MCD de type générique dans la liste disponible sur le serveur.
Précondition(s)	Aucune
Postcondition(s)	Aucune
Scénario nominal	1 L'administrateur accède à l'interface permettant de gérer les données. 2 L'administrateur sélectionne sur son poste un MP ou un MCD. 3 L'administrateur envoie le fichier sur le serveur.
Alternative(s)	2.a. L'administrateur sélectionne un MP ou un MCD dans la liste disponible sur le serveur. 3.a. L'administrateur supprime la sélection.

Tableau 18 : Description contextuelle du cas d'utilisation gérer les données

4.3.3. Planification du projet en itérations

Après ce travail d'identification des cas d'utilisation, nous pouvons les classifier en tenant compte de deux facteurs : la priorité fonctionnelle et le risque technique.
Remémorons nous qu'un des principes du processus unifié consiste à identifier et lever les risques rapidement. Dans le but d'itérer sur nos cas d'utilisation, nous prenons en compte de façon combinée la priorité fonctionnelle et l'estimation du risque.
Le tableau ci-dessous (cf. Tableau 19), précise l'analyse et la planification de notre projet.

Cas d'utilisation	Priorité	Risque	Itération #
Consultation SIST publique	Basse	Moyen	13
Demande d'accès à un SIST semi-public	Basse	Bas	14
Demande de création de compte	Moyenne	Bas	8
Authentification	Moyenne	Moyen	4
Consultation des SIST semi-publics	Basse	Moyen	12
Création d'un SIST	Haute	Haut	1
Copie d'un SIST	Haute	Moyen	3
Modification d'un SIST	Haute	Haut	2
Modification de la visibilité	Moyenne	Bas	9
Modification des droits d'accès	Basse	Moyen	10
Validation création de compte	Moyenne	Moyen	7
Validation changement de statut	Basse	Bas	11
Création de compte	Moyenne	Moyen	5
Suppression de compte	Basse	Bas	15
Gérer les données	Basse	Bas	16
Visualiser un SIST	Moyenne	Moyen	6

Tableau 19 : Planification du projet

Ce découpage préliminaire devra être remis en cause et réévalué à chaque fin d'itération. Pour un souci de lisibilité et de clarté, dans la suite du document, nous nous contentons de modéliser le cas d'utilisation le plus important : création d'un SIST.

Au vu des nombreux cas d'utilisation du projet, dans la suite du document, nous avons pris parti de ne présenter que l'utilisateur « concepteur », puisqu'il correspond à la première itération à développer.

4.3.4. Maquettage de l'IHM

A partir des différents cas d'utilisation que nous avons définis dans la section précédente, nous avons réalisé une maquette de l'interface utilisateur de l'application GenGHIS-Web. Cette maquette a pour avantage de modéliser rapidement l'aspect visuel du site web et permet ainsi de vérifier auprès du maître d'ouvrage, la pertinence des scénarios des différents cas d'utilisation et l'utilisabilité du système.

Pour la réalisation de cette maquette nous avons utilisé l'outil Pencil. Pencil est un logiciel *opensource*[23] qui se présente comme une extension (plug-in) du navigateur FireFox et qui offre une interface graphique pour produire des maquettes HTML. Les pages HTML générées permettent ensuite à l'utilisateur de simuler la navigation entre les différents écrans de l'interface graphique du système maquetté.

Nous n'exposerons ici que quelques captures d'écrans d'un utilisateur « concepteur ».
La Figure 45 présente les captures d'écrans des étapes successives de la création d'un SIST. Ces étapes correspondent au scénario nominal du cas d'utilisation : création d'un SIST (cf. Tableau 7).

[23] Un logiciel open source permet sa libre redistribution et l'accès à son code source.

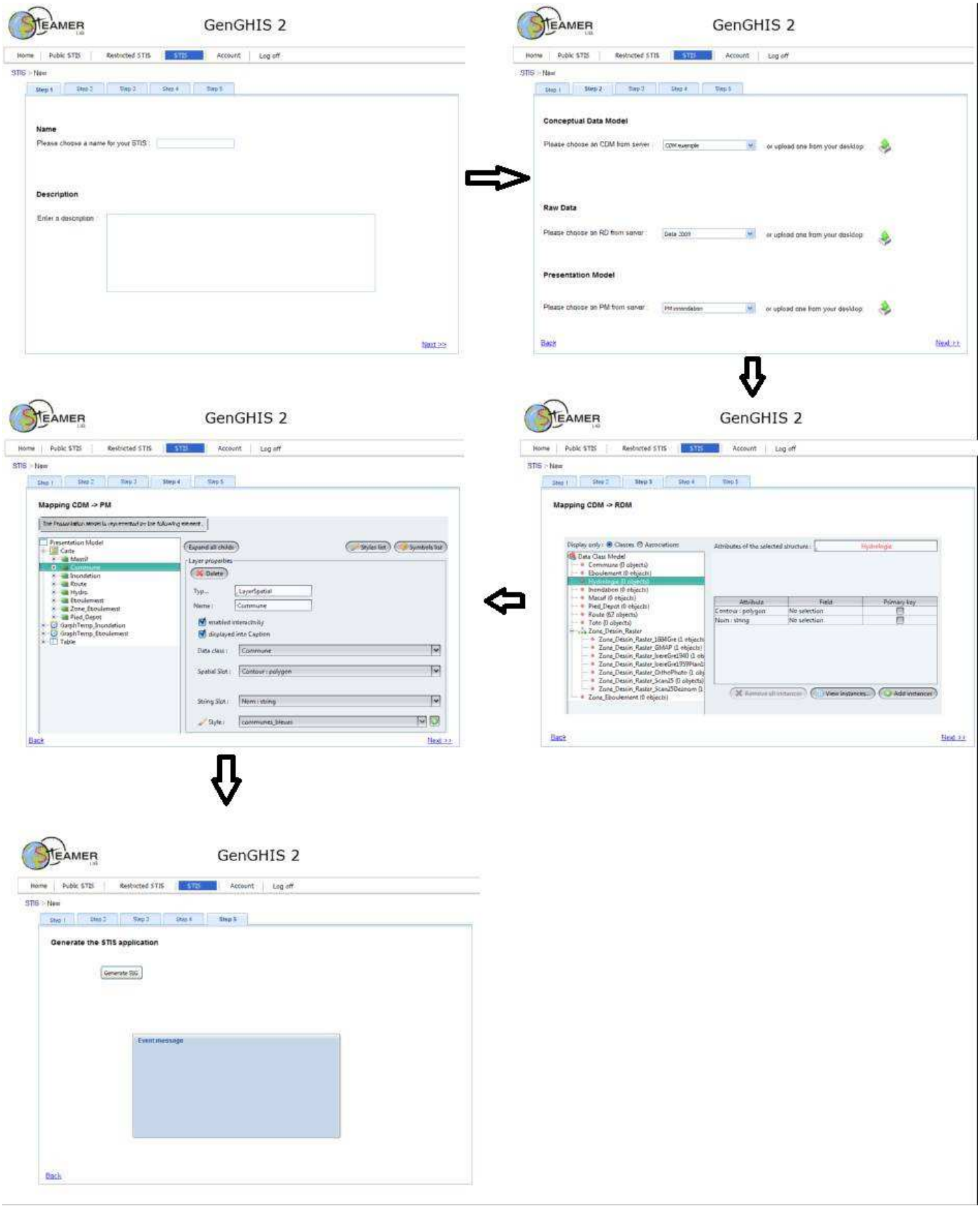


Figure 45 : Maquette de la création d'un SIST

4.3.5. Classes d'analyse

La conception objet demande une description structurelle, statique, du système à réaliser sous forme d'un ensemble de classes logicielles. Avant de passer à la conception, nous devons identifier les principales classes d'Interface Homme Machine (IHM) ainsi que celles qui décrivent la cinématique de l'application. L'étape d'analyse orientée objet est la décomposition d'un domaine en classes conceptuelles, représentant les entités de ce domaine. Avec UML, nous employons un diagramme de classes, dans lequel nous faisons figurer les classes conceptuelles ou les objets du domaine, les associations entre classes conceptuelles et leurs attributs. En vue d'identifier les concepts du domaine nous nous posons la question suivante sur l'ensemble des cas d'utilisation : quels sont les concepts métiers qui participent à ce cas d'utilisation ?

Application au cas d'utilisation création d'un SIST
L'expression préliminaire des besoins, nous indique que l'utilisateur devra nommer le SIST et choisir un modèle conceptuel de données et un modèle de présentations. Nous pouvons en déduire les concepts métiers SIST, MCD et MP. Les concepts métier sont qualifiés d'entités et sont généralement persistants. Nous ajoutons une entité utilisateur, car nous pouvons d'ores et déjà affirmer qu'un lien fort subsistera entre cette entité et l'entité SIST.

La maquette (cf. Figure 45) nous a montré cinq écrans principaux, qui nous conduisent aux classes de dialogue permettant les interactions entre le site web et les utilisateurs :

- l'écran où l'on renseigne le nom et la description du SIST (dialogue *Nom_Description*) ;
- l'écran où l'on choisit le modèle conceptuel de données, le modèle de présentations et le fichier des données brutes (dialogue *Choix_MCD_MP*) ;
- l'écran où l'on instancie le modèle de données (dialogue *Instanciation_MCD*) ;
- l'écran où l'on enrichit le modèle de présentations (dialogue *Instanciation_MP*) ;
- enfin, l'écran où l'on lance la génération du SIST (dialogue *Génération_SIST*).

Nous appliquons les principes de l'approche MVC, pour cela, nous rajoutons à notre analyse une classe contrôle qui contient la cinématique de l'application. Elle fait la transition entre les dialogues et les concepts du domaine, en permettant aux écrans de manipuler des informations détenues par des objets métiers.

Nous retrouvons dans la figure suivante (cf. Figure 46), l'ensemble des classes énumérées auparavant, complétées de leurs associations. Signalons la relation d'agrégation (losange vide) entre l'entité SIST et le MCD/MP, exprimant une relation de contenance. Un SIST est composé d'un MP et d'un MCD, mais les MP et MCD ont une existence en dehors du SIST.

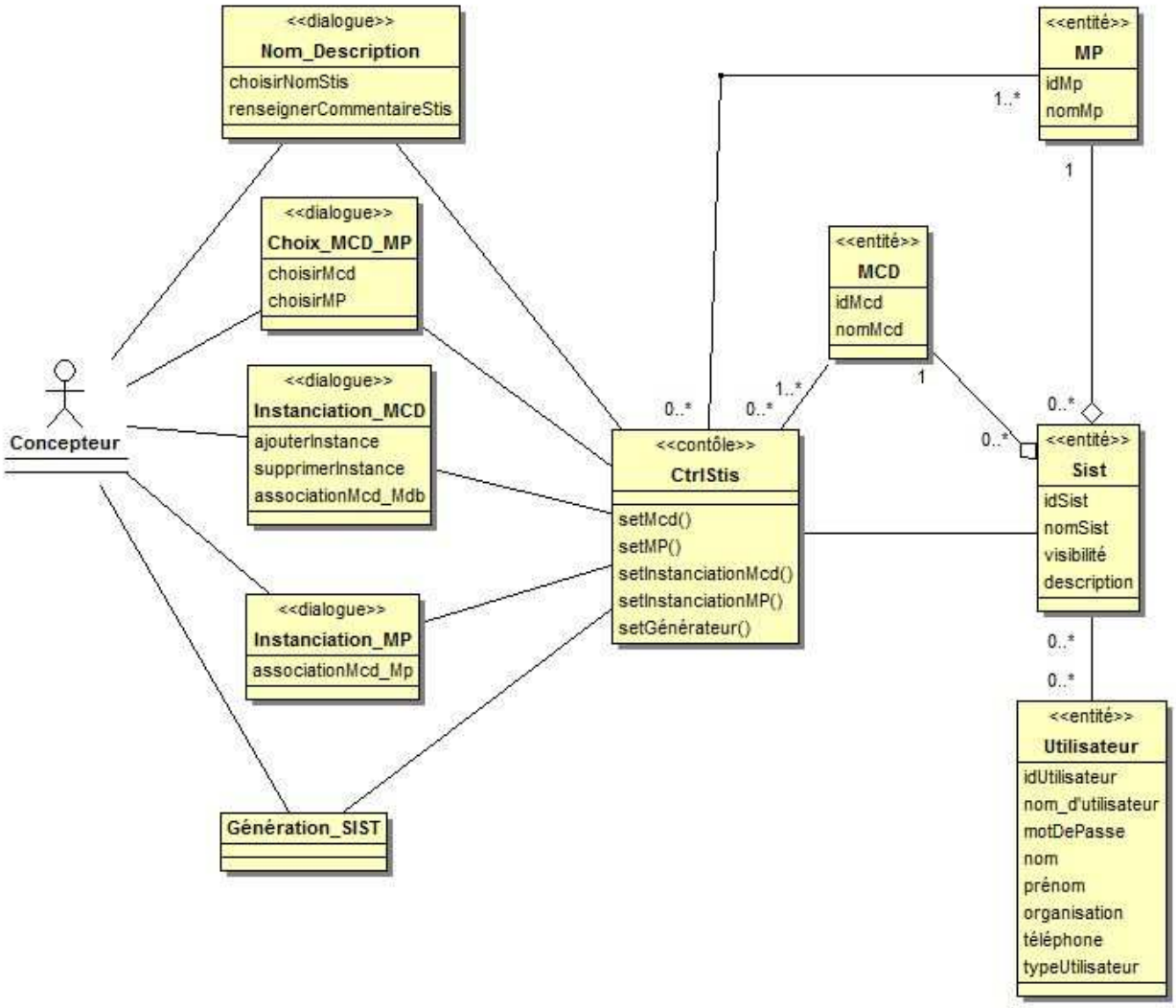


Figure 46 : Diagramme de classes de la création de SIST

Ce diagramme de classes est une étude préliminaire qui sera transformée et enrichie lors de la phase de conception.

4.4. Modélisation de la navigation

Dans la section précédente, nous avons identifié les principales classes d'Interface Homme Machine (dialogues) ainsi que celles qui décrivent la cinématique de l'application (contrôles).
L'étape qui nous conduit à la conception du système est la modélisation de la navigation.
Les IHM modernes, en particulier les sites internet, cherchent à faciliter la communication avec l'utilisateur. Ces IHM sont composées d'un ensemble d'éléments tels que les fenêtres, les liens, et les boîtes de dialogues qui présentent les éléments graphiques ou textuels reflétant l'état courant du système. Ces mêmes interfaces permettent de modifier l'état courant du système en envoyant des messages à celui-ci.

Nous utilisons ci-après (cf. Figure 47) un diagramme d'états UML afin de modéliser la navigation.
Nous apportons quelques particularités supplémentaires pour modéliser plusieurs concepts :
-Une page complète du site (« page ») ;
-Un cadre particulier à l'intérieur d'une page (« cadre »).

La maquette nous a permis de déduire que les étapes successives de création d'un SIST sont situées dans une même page. Nous devons en tenir compte dans la navigation, la résultante en est la décomposition des étapes dans des cadres.
Pour des raisons de structuration et de lisibilité nous indiquons des points de connexion vers d'autres diagrammes qui ne sont pas présentés. Nous ne faisons pas apparaître les exceptions représentant les messages d'erreurs.
A n'importe quelle étape de la création d'un SIST, l'utilisateur peut quitter la page pour y revenir plus tard. Le système devra enregistrer l'étape courante afin de restituer son contenu à l'utilisateur (d'après les besoins fonctionnels). Un état historique permet à un état composite de se souvenir de son dernier sous-état actif avant sa dernière sortie. Nous rajoutons un état historique, à la navigation, représenté par un petit cercle contenant la lettre H. L'état final est la génération du SIST.

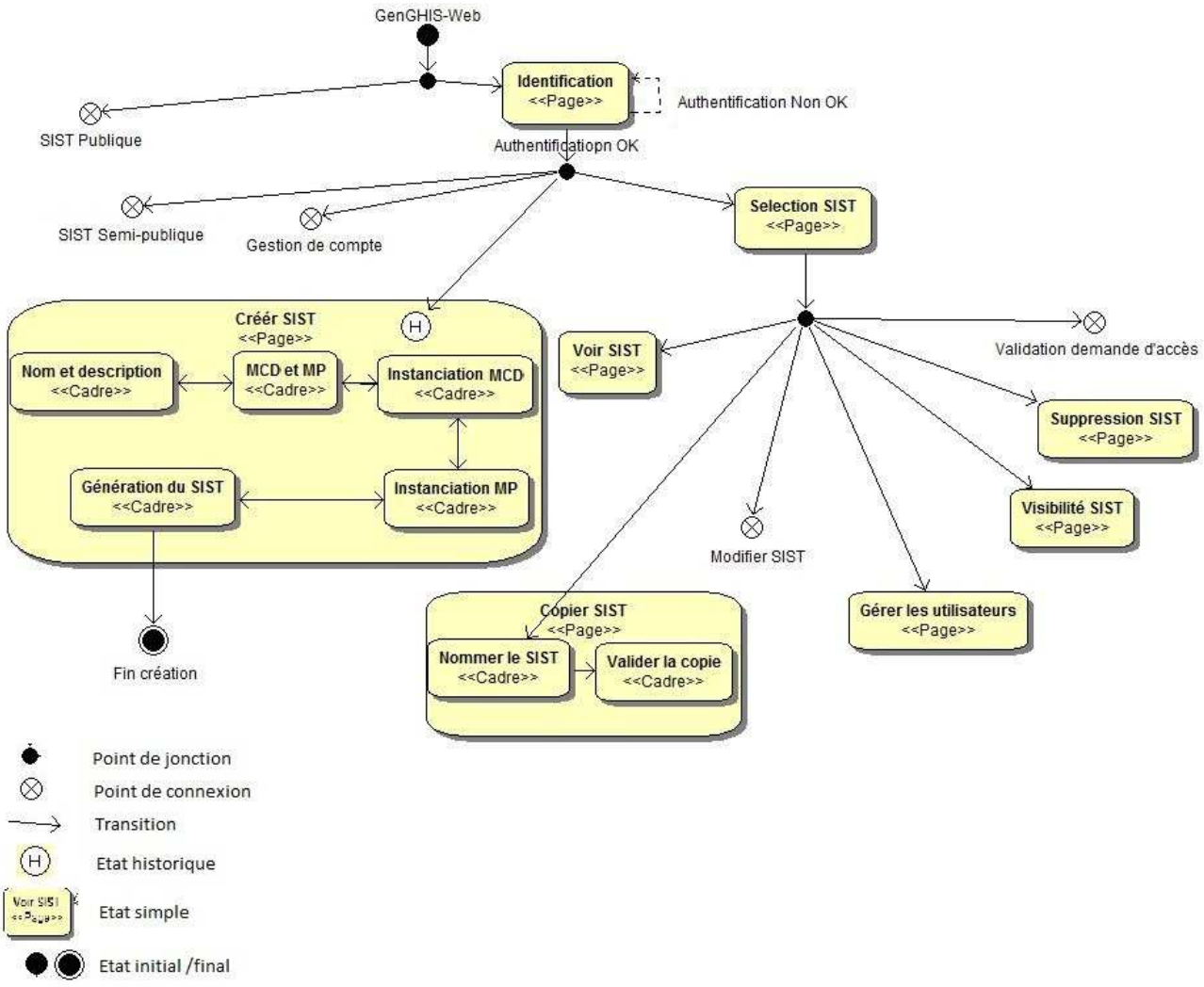


Figure 47 : Diagramme d'états de navigation du concepteur

4.5. Ergonomie

Le cahier des charges impose une ergonomie simple et efficace en tant que besoins non fonctionnels. Dans un premier temps, nous avons effectué des recherches sur les notions d'ergonomie sur l'homme, puis nous avons retenu un certain nombre de caractéristiques à mettre en place dans GenGHIS-Web.

L'ergonomie se caractérise par l'utilisation de connaissances scientifiques relatives à l'homme dans le but d'améliorer son environnement de travail. Ce concept est défini dans le domaine d'internet par la capacité de répondre efficacement aux attentes des utilisateurs et à les guider dans leur navigation.

La « théorie de l'action » (14) modélise les actions et réactions de l'homme. Cette théorie repose sur deux concepts :

- L'homme conçoit des modèles simples pour définir son comportement ;
- L'homme décompose ses actions selon une boucle comprenant 7 étapes :
    - Etablissement d'un but : il s'agit d'une représentation mentale d'un état à atteindre ;
    - Formulation d'une intention : il s'agit de la décision d'atteindre l'objectif fixé ;
    - Mise au point d'un plan d'action : il représente la suite d'actions à mener ;
    - Réalisation du plan d'action : il consiste à activer le système moteur ;
    - Perception de l'état du système, décrivant sous forme de variables psychologiques la perception du changement par l'organisme ;
    - Evaluation de l'état atteint par rapport aux objectifs initialement fixés.

Lorsque l'objectif initial est complexe, il est décomposé en sous-objectifs successifs, jusqu'à ce que le sous-objectif soit réalisable. Le sous-objectif est alors traduit en une action, aboutissant à la définition d'un nouvel objectif (15).

La théorie de l'action définit ainsi deux distances caractérisant l'écart entre le but défini par l'utilisateur et sa réalisation :

- La **distance d'exécution**, représentant l'écart entre le but et l'état du système dans la branche descendante ;
- La **distance d'évaluation**, représentant l'écart entre l'état du système et le but dans la branche ascendante.

Cette liste de caractéristiques psychologiques est loin d'être exhaustive, néanmoins, il faut tirer parti de ces considérations comportementales et des mécanismes d'apprentissage afin de créer des interfaces homme-machine ergonomiques.

Le tableau suivant indique les critères d'ergonomie retenus sur le portail GenGHIS-Web :

Critère	Description
Peu chargé	Les animations sont privilégiées pour afficher des messages forts car elles attirent le regard de l'internaute. L'ensemble des pages contient uniquement les informations essentielles à son

	utilisation.
Facilité de navigation	La « règle des 3 clics », globalement suivie, stipule que toute information doit être accessible en moins de 3 clics.
Repérage	A tout moment, l'internaute doit pouvoir être en mesure de se repérer dans le site. Le logo est présent sur toutes les pages, au même emplacement, et une charte graphique uniforme est appliquée à l'ensemble des pages afin de permettre à l'usager de savoir qu'il est toujours sur le même site. La partie haute du portail est développée dans un fichier en-tête (*header.jsp*) afin d'uniformiser le site.
Liberté de navigation	Le site permet à l'utilisateur de revenir à la page d'accueil et aux principales rubriques par un simple clic, quelle que soit la page sur laquelle il se trouve. Le portail possède une barre de menus permettant d'accéder à la majeure partie des pages.
Temps de chargement	Le temps d'affichage d'une page doit être le plus petit possible.
Interopérabilité	Le respect des standards, en particulier les recommandations d'accessibilité du W3C, permet de garantir un bon niveau d'interopérabilité. Les navigateurs respectant les standards sont compatibles avec le portail.

Tableau 20 : Liste des critères d'ergonomie du portail

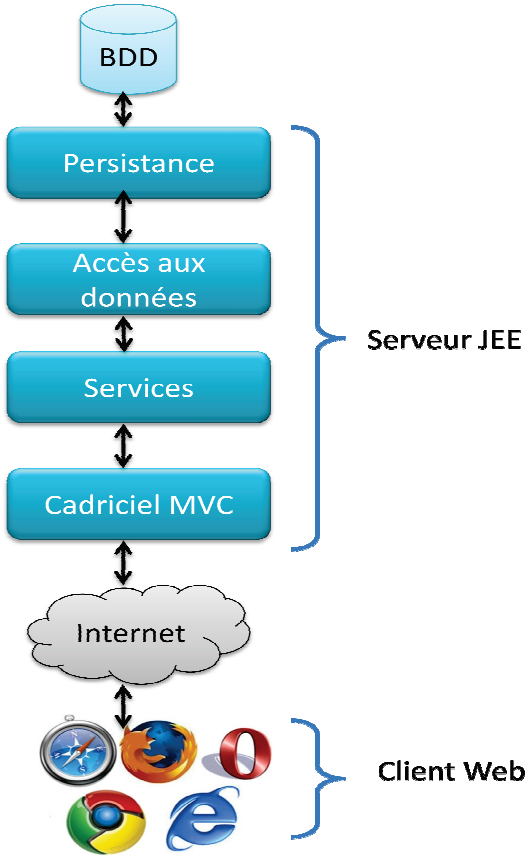
Cette section conclut le chapitre quatre sur la conception de l'application GenGHIS-Web. Dans ce chapitre, nous avons spécifié ce qui doit être réalisé pour atteindre notre objectif. La prochaine étape est la réalisation, objet du prochain chapitre.

Chapitre 5. Réalisations

5.1. Architecture de l'application GenGHIS-Web

GenGHIS-Web est développé sous la spécification Java Entreprise Edition (JEE) qui propose un ensemble d'extensions à la plate-forme standard Java (JSE ou Java Standard Edition) destinées plus particulièrement à la création d'applications réparties. Ces applications sont considérées dans une approche multi-niveaux. L'architecture en couches divise une application en différents modules qui constituent autant de couches. Chaque module a un rôle clairement défini et se doit d'être indépendant des autres. L'objectif est de mettre en avant une meilleure répartition des rôles, la séparation des traitements, ainsi qu'une réduction des dépendances entre les services pour permettre une meilleure évolutivité et faciliter la maintenance du système.

La figure ci-dessous présente les couches développées pour l'application GenGHIS-Web (cf. Figure 48).



48 : Architecture en couches de GenGHIS-Web

Nous distinguons :

- le **système de gestion de bases de données** (SGBD), qui stocke les données utilisées par l'application ;
- la **couche de persistance**, qui gère le mécanisme de sauvegarde et de restauration des données entre le monde Java et le SGBD ;
- la **couche d'accès aux données**, en charge de l'accès aux données et de leur manipulation, indépendamment du SGBD choisi ;
- la **couche de services**, ou couche métier, gère la logique de l'application et les traitements à effectuer sur les données, indépendamment de la provenance des données et de la façon dont elles seront affichées une fois les traitements effectués ;
- le **cadriciel Modèle-Vue-Contrôleur**, façonne l'architecture de l'application web ;
- le **client Web** : un simple navigateur internet.

5.2. Choix technologiques

Avant de réaliser l'implémentation de ces différentes couches pour l'application GenGHIS-Web nous avons dû, pour chacune d'entre elles, procéder à des choix de technologies destinés à faciliter leur mise en œuvre. Nous présentons ces différents choix dans les sections qui suivent.

5.2.1. Cadriciel MVC

De nos jours, il existe de plus en plus d'applications web et celles-ci sont de plus en plus complexes : du simple enchaînement de pages HTML nous sommes passés à des applications proposant une interface graphique digne d'applications classiques (dite "client lourd"). Si les applications web JEE s'appuient sur des technologies de base, les Servlets et pages JSP (Java Server Pages) relativement simples, leur mise en œuvre efficace pour des applications évoluées est loin d'être triviale. C'est pourquoi, afin de faciliter le développement d'application Web de nombreux cadriciels ont été proposés (l'existence de cadriciels pour applications web n'est pas spécifique au monde Java ; on retrouve la même tendance dans les mondes PHP, Python, Ruby…). Ces cadriciels, en prenant en charge tout ou partie de la cinématique des applications web, simplifient[24] le travail du développeur en le déchargeant de la programmation d'un certain nombre de fonctionnalités transversales à toutes les applications web. Ils permettent d'assurer l'évolution et la fiabilité des applications web.

Dans le contexte JAVA / JEE , de nombreux outils ont été développés pour encadrer l'utilisation des Servlets et JSP dont les deux principaux sont Struts et JSF (*Java Server Faces*). Chacun ayant ses points forts et ses points faibles, nous nous sommes donc posés, durant la phase d'étude, la question du choix du cadriciel.

5.2.1.1. Choix du cadriciel MVC

JSF et Struts s'appuient tous les deux sur une architecture de type Modèle Vue Contrôleur (cf. Figure 49). L'interface utilisateur, affichée dans un navigateur web, est construite à partir de pages JSP constituant autant de vues, ces vues sont alimentées à partir de données provenant d'objets Java constituant le modèle. Finalement, la cinématique de l'application est prise en charge par un contrôleur qui assure l'interprétation des requêtes http issues du client et la coordination des modèles et des vues.

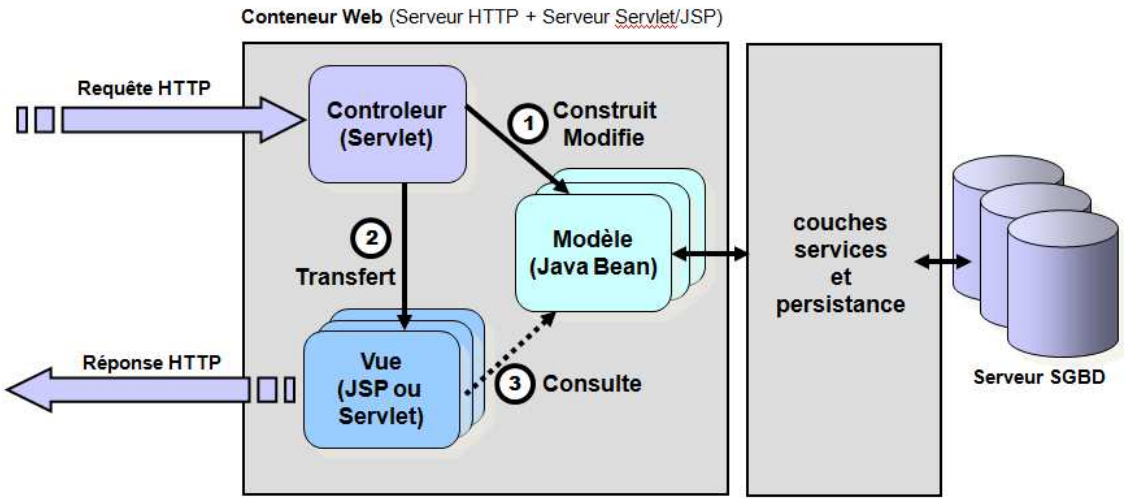


Figure 49 : Architecture MVC d'une application Web Java/JEE

[24] Cette simplification "espérée" par l'utilisation d'un cadriciel, ne doit néanmoins pas masquer la complexité des applications web et la nécessité d'acquérir la maîtrise du cadriciel qui passe nécessairement par une phase d'apprentissage plus ou moins longue.

Dans Struts, on trouve un contrôleur principal de classe *ActionServlet* permettant de traiter les requêtes http qui passent le relais à des mini contrôleurs qui sont des objets héritant de la classe `Action` de Struts. Ces mini contrôleurs délèguent le traitement métier à des classes externes et récupèrent le résultat qui sera placé dans une *Java Server Page* pour pouvoir être affiché à l'utilisateur.

Concernant JSF, la *Face Servlet*, à travers un fichier de configuration, permet de traiter des services communs tels que la navigation ou la gestion des messages. JSF gère les vues sous la forme d'un arbre de composants graphiques (*IUComponents*), pouvant être associés à des classes qu'on appelle *renderers* et qui sont responsables du rendu graphique de ces composants.

Struts et JSF n'ont pas tout à fait le même champ d'action, le premier est orienté *Action Controller* tandis que le deuxième est orienté *View Controler*. JSF donne au développeur un large choix de fonctionnalités sur la création d'une vue, riche de composants graphiques configurables, tandis que Struts qui est, orienté *Action Controler*, gère plus facilement la partie contrôleur que le côté vue.

Le choix de l'un de ces deux cadriciels dépend de la nature du projet que l'on souhaite construire. Struts est plutôt directif, il a une place pour chaque chose (la validation, les actions,…), le développeur n'a pas de mal à l'appréhender et ne fait que construire ses objets suivant une architecture claire et déjà posée par le cadriciel ; il est également très documenté et bénéficie de plus de dix ans d'existence sur le marché. Struts remplit largement les tâches requises pour un projet simple, ne nécessitant pas de fonctionnalités complexes côté interface utilisateur.

A l'inverse de Struts, pour une application d'ergonomie complexe, JSF est mieux adapté, car il offre un ensemble de composants graphiques de haut niveau parmi lesquels le développeur choisit ceux qui lui conviennent. Cet ensemble est extensible, de nombreuses librairies open sources de composants riches et stables sont disponibles. En revanche, le développeur a en charge un certain nombre de tâches, par exemple la validation qu'il a le choix d'intégrer directement dans une JSP ou dans une classe Java spécifique.

L'équipe STeamer possède des compétences concernant le cadriciel Struts, plusieurs applications s'appuyant sur celui-ci ont été développées, mais n'a que très peu d'expérience avec JSF. L'occasion de monter en compétences sur JSF, qui a été intégré dans les spécifications JEE 5, ainsi que la relative complexité de l'application GenGHIS-Web et de ces interfaces, nous ont conduits à choisir ce dernier.

5.2.1.2. Cycle de vie d'une page JSF

Vu de l'extérieur, le cycle de vie d'une page JSF correspond globalement à celui d'une page JSP : un client web émet une requête HTTP vers la page concernée, le serveur répond en renvoyant une page traduite en HTML. En pratique, il est découpé en plusieurs phases présentées sur la figure suivante (cf. Figure 50).

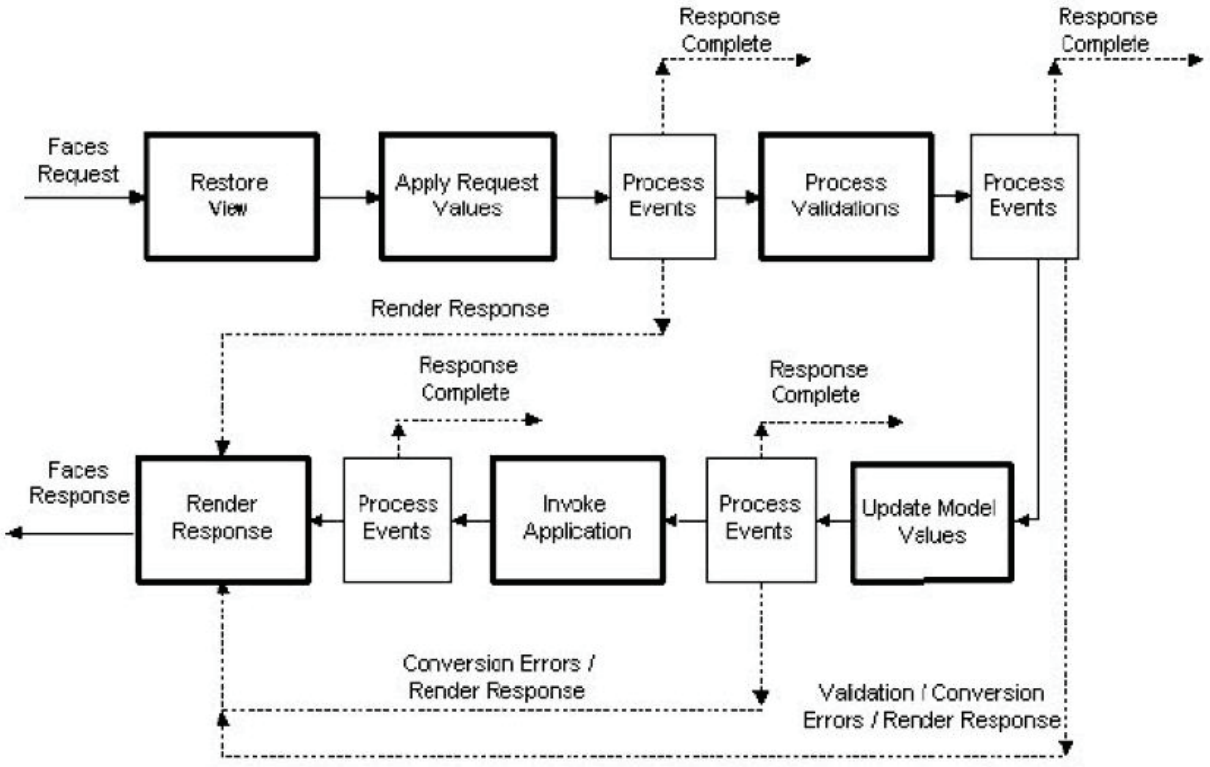


Figure 50 : Cycle de vie d'une page JSF(16)

Il est important de distinguer deux types de requêtes :

- la requête initiale, au cours de laquelle le client accède pour la première fois à la page JSF et où seules les phases de restitution de la vue et la traduction de la réponse sont exécutées.
- une requête dite *postback,* correspondant le plus souvent à la validation d'un formulaire précédemment chargé dans le navigateur du client, là, toutes les phases du cycle de vie de la page sont exécutées, sauf en cas d'erreurs.

Phase de Restitution de la vue (*Restore view*) : C'est cette phase qui est exécutée dès qu'un internaute essaie d'atteindre une page JSF. Elle associe une vue à la page visitée. Dans le cas d'une requête initiale, l'application restitue une page vierge et dans le cas d'une requête *postback*, elle restitue la vue précédemment associée à la page. Les composants graphiques lient à la vue les gestionnaires d'événements et les validateurs requis ; il en découle un arbre de composants. Cette vue est sauvegardée dans l'objet prédéfini appelé `FacesContext`.
Le déroulement du cycle de vie se poursuit en fonction du type de requête. Dans le cas d'une requête initiale, un appel de la méthode `renderResponse` sur l'objet `FaceContext` provoque la traduction de la vue, puis sa restitution au client. Dans le cas d'une requête *postback*, la restitution de la vue s'effectue compte tenu des informations fournies dans la requête.

Phase Application des paramètres de requête (*Apply request values*) : A l'issue de la phase précédente, nous possédons une vue et son arbre de composants associés. Ces composants sont parcourus successivement pour invoquer leur méthode `decode()`, chargée d'analyser la requête, afin d'attribuer au composant la valeur qui doit lui être affectée.

Dans cette phase, JSF permet d'utiliser des valideurs de composants, qui vont par exemple vérifier qu'une valeur numérique saisie dans une zone de texte est bien comprise dans un domaine donné.

Phase de Validation (*Process Validations*) : Dans cette étape, l'application JSF effectue toutes les validations attendues, pour chacun des composants de l'arbre associés à la vue. Pour cela, les attributs de composant définissant les règles de validation sont examinés, puis comparés à la valeur du composant. Lorsque cette valeur est effectivement incorrecte, un message d'erreur est ajouté à l'objet `FacesContext` et on passe directement à la phase de traduction de la réponse : la page web est alors restituée à l'internaute, accompagnée du message d'erreur issu du processus de validation.

Phase Mise à jour du modèle (*Update Model Values*) : Chaque composant d'interface JSF peut être associé à un JavaBean côté serveur. Après s'être assurée de la validité des données saisies lors de la phase de validation, l'application JSF parcourt à nouveau l'arbre de composants, pour affecter la valeur de chaque composant graphique au JavaBean qui lui est associé. Là encore, si un problème de conversion survient au moment de la mise à jour de la propriété du JavaBean, le cycle de vie de la page se poursuit directement par la phase de traduction de la réponse et un message d'erreur adapté est présenté à l'internaute.

Phase Appel de l'application (*Invoke application*) : Lorsque l'internaute valide un formulaire, ou clique sur un lien hypertexte, l'application JSF génère respectivement un objet de type "événement de formulaire" ou un objet de type "événement de commande". Ces objets sont qualifiés d'événements de niveau application : au cours de cette phase, ils sont pris en charge par des gestionnaires spécifiques dont le rôle est de déterminer une URL vers laquelle la navigation est dirigée.

Phase Restitution de la réponse (*Render response*) : Cette phase correspond au moment où l'implémentation Java Server Faces rend la main au conteneur en charge des pages JSP de l'application web. L'état des composants, au moment où survient la phase `Render response`, est présenté dans l'interface de l'utilisateur.

5.2.1.3. Composants graphiques

Dans une page internet, les composants graphiques sont l'un des éléments principaux qui permettent à l'internaute d'interagir avec l'application web. Dans Java Server Faces le support de la partie graphique est assuré par un ensemble de balises JSF insérées dans une page JSP. Après interprétation, ces balises sont transformées en HTML, compréhensible par le navigateur. Ces composants sont représentés par des objets Java issus de classes dérivées de la classe `javax.faces.component.UIComponentBase`. L'imbrication des composants à l'intérieur de la page induisent une structure d'arbre. Ces composants peuvent être associés à diverses interfaces permettant notamment :

- le maintien de l'état entre les pages internet ;
- la prise en charge de la gestion événementielle ;
- la restitution de rendus visuels spécifiques, variables suivant le composant graphique concerné.

Par défaut, il existe deux librairies de balises JSF: HTML et Core. La première sert au rendu graphique et la seconde contient des fonctionnalités de base ne générant aucun rendu.

Les jeux de composants standard de JSF s'avèrent trop limités et insuffisants pour le développement d'applications d'entreprise. Il est possible dès lors de créer ses propres composants ou d'utiliser des jeux de composants additionnels offrant des fonctionnalités plus riches. Dans le cadre du projet, nous avons choisi la seconde solution. Il existe plusieurs cadriciels pour Java Server Face : Myfaces, Icefaces,… Notre choix s'est porté sur le cadriciel RichFaces étant données l'importante documentation et la communauté présentes sur internet.

RichFaces est né du besoin d'associer la technologie Ajax à celle de JSF. Il comporte un ensemble de composants, permettant d'ajouter des fonctionnalités Ajax avancées aux composants standards JSF, sans manipulation de code JavaScript. Depuis mars 2007, le projet RichFaces qui était initialement une solution propriétaire de JBoss est devenu *open-source* et a fusionné avec la bibliothèque a4j (Ajax for JSF) qui permet d'ajouter des traitements Ajax aux pages JSF.

Comme le montre la figure 51, RichFaces se greffe côté serveur en amont de JSF.

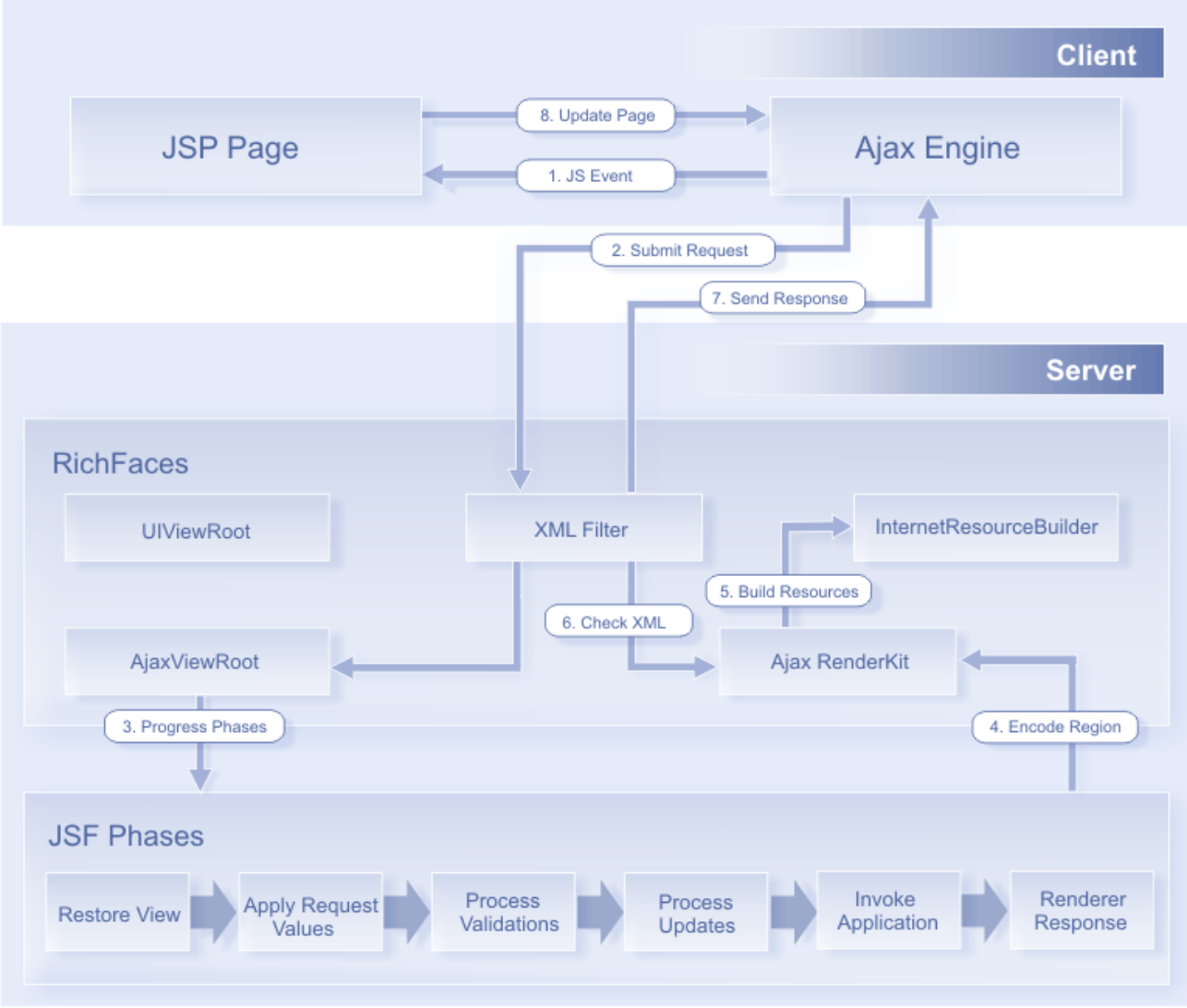


Figure 51 : Cycle de vie d'une requête Ajax avec RichFaces (17)

Lorsque le client envoie une requête Ajax, le moteur Ajax transforme la requête et l'envoie au filtre a4j *ajax engine* (étape 1), ce dernier convertit les données au format XML et transfère la requête à la servlet Faces Servlet (étape 2 et 3) ; ainsi, la requête suit le processus normal d'une requête JSF et passe par les six phases : la phase de Restitution de la vue (*Restore view*), la phase Application des paramètres de requête (*Apply request values*) , la phase de Validation (*Process Validations*), la phase Mise à jour du modèle (*Update Model Values*), la phase Appel de l'application (*Invoke application*), la phase Restitution de la réponse (*Render response*). La bibliothèque a4j gère l'arbre de composants tout au long de la requête et la réponse Ajax.

5.2.2. Persistance des données

La persistance se définit comme le mécanisme de sauvegarde et de restauration de données, que ce soit sur un disque dur ou dans une base de données relationnelles (11). Dans le monde Java, l'une des solutions de persistance est l'utilisation de l'interface de programmation JDBC (*Java DataBase Connectivity*) qui permet la communication avec un SGBDR. Bien que souvent utilisée, elle nécessite l'implémentation par le développeur de l'interaction avec les bases de données et la programmation des requêtes SQL nécessaires à la persistance des données. Tâche souvent fastidieuse, la transcription des objets Java en base de données relationnelles doit être faite "manuellement".

Nous nous sommes libérés de cette problématique en nous orientant sur la notion d'association relationnel-objet (mapping objet-relationnel, en anglais *object-relational mapping* ou *ORM*) qui rend transparent le pont entre le monde orienté objet de Java et le monde relationnel des systèmes de gestion de bases de données. Le tableau ci-dessous (cf. Tableau 21) transcrit les concepts objet avec les concepts relationnels.

Monde objet	**Monde relationnel**
Classe	Table
Attribut	Colonne
Instance	Enregistrement
Association	Clé étrangère ou 1 classe et des clés étrangères

Tableau 21 : Transcription des concepts objet et relationnels

L'interface de programmation Java Persistence API (JPA) introduite dans la version 6 de JSE (Java Standard Edition) se situe au-dessus de la couche JDBC et permet de la rendre transparente. Nous n'implémentons pas les accès directs à la base de données, mais nous appelons des méthodes objet de haut niveau qui prennent en charge la transformation d'objets vers la base de données et inversement, que cela soit pour des lectures ou des mises à jour (création, modification ou suppression).

Pour le projet GenGHIS-Web nous avons porté notre choix sur le cadriciel Hibernate qui est une implémentation de l'interface. Il fait le lien entre l'application Java et la base de données (cf. Figure 52).

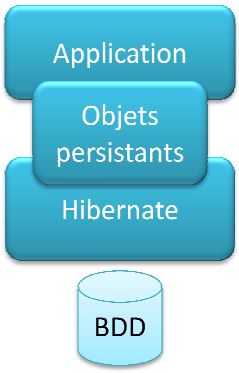


Figure 52 : Architecture d'un projet Hibernate

5.2.3. Stockage des données

Pour le stockage des données (description des utilisateurs, description de SIST…) de l'application GenGHIS-Web nous avons décidé de nous appuyer sur un système de gestion de bases de données relationnelles (SGBDR). Nous souhaitions utiliser un logiciel libre et dont la communauté de développeurs soit conséquente. Parmi les solutions disponibles, nous nous sommes tournés vers *PostgreSQL* disponible selon les termes d'une licence de type BSD. Ce choix fut motivé par la présence de ressources au niveau de l'équipe Steamer ; en effet, un certain nombre des applications qu'elle developpe fonctionne avec ce SGBDR. De plus, PostgreSQL dispose d'un complément, PostGIS (*Geographical Information System*), qui permet la manipulation d'informations de géométrie (points, lignes, polygones), conformément aux standards établis par l'Open Geospatial Consortium. Cette possibilité pourrait s'avérer intéressante dans l'éventualité d'évolutions futures de GenGHIS-Web, dans laquelle le stockage des données spatio-temporelles d'un SIST pourrait être effectué dans une base de données.

5.2.4. Environnement de développement

Après les technologies utilisées pour GenGHIS-Web, il restait encore à choisir les outils à utiliser pour le travail de développement proprement dit. La problématique consistant à rechercher les outils les mieux adaptés pour la réalisation et l'intégration des différentes couches de l'application.

5.2.4.1. L'IDE Eclipse

Eclipse est un projet de la Fondation Eclipse[25] visant à produire tout un environnement de développement libre, extensible, universel et polyvalent.

[25] La fondation Eclipse est une organisation à but non lucratif supervisant le développement de l'IDE open source Eclipse et des projets gravitant autour, et qui aide à cultiver une communauté open source et un écosystème de produits et de services complémentaires autour d'Eclipse.

Le projet Eclipse, développé essentiellement en Java, est bien plus large que le projet Eclipse IDE (*Integrated Devlopment Environment*), environnement de développement intégré (EDI) initialement conçu pour le langage Java. En plus du codage, il offre un éventail très large d'outils gravitant autour de la réalisation de logiciels. Ainsi, outre un support pour de nombreux langages de programmation, il autorise les phases de modélisation, de conception, de test,… Cet ensemble est rendu possible grâce à la notion de greffon (*plug-in*) autour duquel il a été développé.

La base de l'environnement est *Eclipse Platform* composée de (11) :

- *Platform Runtime* démarrant la plate-forme et gérant les plug-ins ;
- SWT, bibliothèque graphique de base offrant un ensemble de composants graphiques s'interfaçant directement avec les bibliothèques graphiques du système hôte (MFC pour Windows, Gnome ou GTK sous linux…) ;
- *JFace*, une bibliothèque graphique de plus haut niveau basée sur SWT ;
- *Eclipse Workbench*, la dernière couche graphique permettant de manipuler des composants, tels que des vues, des éditeurs et des perspectives.

Ces composants de base peuvent être réutilisés pour développer des clients lourds indépendants d'Eclipse grâce au projet Eclipse RCP (*Rich Client Platform*).

L'IDE Eclipse est un exemple d'une telle application RCP ; il intègre un ensemble d'outils de développement Java ajoutés en tant que plugins et regroupés dans le projet *Java Development Tools* (JDT) : éditeur syntaxique avec complétion de code, débogueur, explorateur de classes, etc. En plus de JDT qui propose les outils de base pour le développement d'applications Java standard, il existe également de nombreux plug-in offrant des extensions pour le développement d'application JEE regroupés dans le projet WTP (Web Tools Platform). WTP offre un support complet pour la création et le déploiement de projets JEE.

C'est cette richesse des outils et la qualité générale de l'IDE Eclipse qui font que nous l'avons choisi comme plate-forme de développement pour le projet GenGHIS-Web. En effet, il offre au sein d'un même environnement un ensemble d'outil qui couvrent tous les besoins pour la réalisation des différentes couches de l'application GenGHIS-Web.

5.2.4.2. Moteur de servlets : Tomcat

Pour exécuter une application web JEE, il est nécessaire d'avoir un conteneur web capable d'exécuter des servlets Java et des pages JSP et venant se greffer à un serveur web comme Apache ou Microsoft IIS (Internet Information Services). Eclipse IDE, étendu avec WTP, permet d'intégrer un ou plusieurs conteneurs web afin de pouvoir tester directement les composants Java d'une application web en cours de développement.

De nombreux conteneurs existent, parmi ceux-ci, nous avons porté notre choix sur Apache Tomcat. Apache Tomcat, comme son nom l'indique est un projet de la fondation Apache, il s'agit d'un conteneur d'applications internet Java issu du projet Jakarta[26] de la fondation Apache[27]. Tomcat a été écrit en langage Java. Il peut donc s'exécuter via la machine virtuelle Java sur n'importe quel système d'exploitation la supportant.

[26] Jakarta est un ensemble de projets de logiciels libres, écrits en langage Java.

[27] L'Apache Software Foundation (Fondation Apache) est une organisation à but non lucratif qui développe des logiciels libres.

En tant qu'implémentation de référence, facile à mettre en œuvre et riche en fonctionnalités, Tomcat est quasi incontournable dans les environnements de développements. Les qualités de ses dernières versions lui permettent d'être de plus en plus utilisé dans des environnements de production(18).

Tomcat est souvent utilisé en association avec un autre serveur web, le plus souvent : Apache. Apache englobe toutes les pages web traditionnelles, et Tomcat uniquement les pages d'une application web Java.

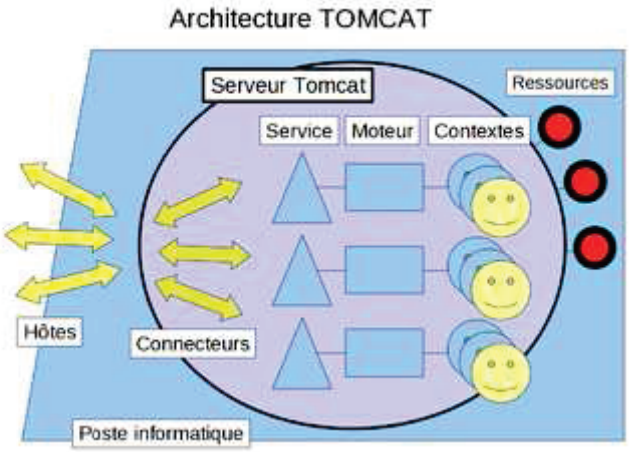


Figure 53 : Architecture de Tomcat (11)

L'architecture du logiciel se compose des éléments suivants (cf. Figure 53) (19) :

- un serveur (*Server*), soit tomcat en cours d'exécution ;
- des services (*Service*), intermédiaires collectant différents canaux de transmissions vers un traitement ;
- un moteur (*Engine*), qui pour chaque service traite les requêtes des collecteurs et renvoie les réponses ;
- des hôtes (*Host*), qui relient une adresse réseau avec le serveur ;
- des connecteurs (*Connector*), qui interprètent un canal et protocole de communication réseau à disposition des clients. Le connecteur HTTP est le plus typique ;
- des contextes (*Context*), qui sont les applications web.

Le contexte est le lieu privilégié pour situer un service informatique que l'on veut rendre sur un réseau. Les autres modules sont mis en oeuvre par le logiciel Tomcat lui même.

5.3. Implémentation

Les technologies utilisées ayant été présentées, nous pouvons maintenant décrire les détails de l'implémentation de GenGHIS-Web. Pour cette description, nous procédons en remontant au travers de couches. Nous partons de la couche de stockage des données à la couche cadriciel MVC (cf. Figure 48).

5.3.1. Stockage des données

Comme nous l'avons indiqué précédemment dans la section des choix technologiques, nous avons utilisé le SGBDR PostgreSQL pour la conservation des données manipulées par l'application GenGHIS-Web. La base de données que nous avons créée contient les tables suivantes (cf. Figure 54) :

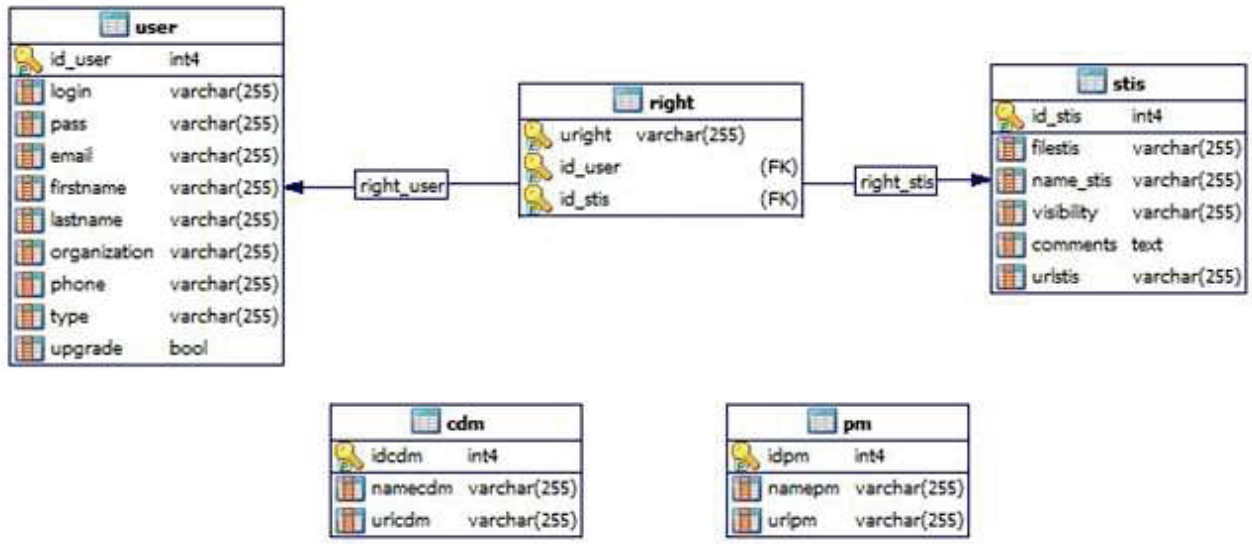


Figure 54 : Modèle physique de la base de données GenGHIS-Web

- Une table *USER* : elle contient toutes les informations relatives à un utilisateur : son nom d'utilisateur (*login*), son mot de passe (*pass*), son courriel (*email*), son nom de famille (*lastname*), son prénom (*firstname*), son appartenance à une organisation (*organization*), son numéro de téléphone (*phone*), son type (concepteur, administrateur,…) et enfin un champ qui indique s'il a fait une demande de changement de statut (*upgrade*).
- Une table *STIS* : elle contient toutes les informations relatives à un SIST : l'emplacement des fichiers du SIST (*filestis*), son nom (*name_stis*), sa visibilité (*visibility*), des commentaires (*comments*), et l'adresse internet pour accéder au SIST généré (*urlstis*).
- Une table *RIGHT* : elle définit pour un couple SIST / Utilisateur le type de droit qui les relie.
- Une table *PM* : elle contient le nom (*namepm*) et l'adresse (*urlpm*) pour accéder à un modèle de présentation.
- Une table *CDM* : elle contient le nom (*namecdm*) et l'adresse (*urlcdm*) pour accéder à un modèle conceptuel de données.

5.3.2. La couche de persistance

Dans la couche de persistance nous reprenons l'architecture du modèle physique de la base de données présentée dans le paragraphe précédent. Il s'agit, à l'aide du cadriciel Hibernate, d'effectuer une correspondance entre des objets Java et les différentes tables de la base de données.

La première étape de mise en place du cadriciel Hibernate est la création des classes persistantes qui représentent les entités que nous voulons stocker dans notre base de données. Ces classes sont de simples classes Java (*Plain Old Java Objects*[28] ou *POJO*) utilisant les conventions de nommage standard des Java Beans (accesseurs *get…* et modifieurs *set…* pour les différents attributs, constructeur sans paramètre). Il est important de souligner la présence d'une propriété `id` contenant la valeur d'un identifiant unique pour une entité particulière. Le code ci-dessous correspond à un fragment de la classe `Stis`.

```
public class Stis implements java.io.Serializable {

    private int idStis;
    private String fileStis;
    private String nameStis;
    private String visibility;
    private String comments;
    private String urlStis;
    private Set<Right> rights = new HashSet<Right>();

    public int getIdStis() {
        return this.idStis;
    }
    ...
}
```

On remarque la similitude entre la classe Stis et la table du même nom de la Figure 54.

La seconde étape consiste à indiquer à Hibernate comment charger et stocker les objets d'une classe persistante : c'est le rôle des fichiers de mapping Hibernate. Ces fichiers indiquent à hibernate à quelle table accéder dans la base de données, ainsi que les colonnes de cette table à utiliser. Ils sont sous la forme de fichiers XML[29].

Nous avons regroupé dans le package `org.STeamer.genghis2.model` les classes liées à la couche de persistance ainsi que les fichiers de mapping. Le code ci-dessous correspond au fichier de mapping de la classe Stis :

```
<hibernate-mapping package="org.STeamer.genghis2.model">
 <class catalog="public" name="Stis" table="STIS">          (1)
  <id name="idStis" type="int">                             (2)
   <column name="id_stis"/>
   <generator class="increment"/>
  </id>
  <property generated="never" lazy="false" name="fileStis" type="string">
   <column name="fileStis"/>
  </property>
  <property generated="never" lazy="false" name="nameStis" type="string">
   <column name="name_stis"/>
  </property>
  <property generated="never" lazy="false" name="visibility" type="string">
   <column name="visibility"/>                              (3)
```

[28] Par le terme de POJO, il est mis l'accent sur le fait que ces classes ne nécessitent pas la réalisation d'interfaces spécifiques comme cela était le cas avec les Entreprise Java Beans dans les versions antérieures de JEE.

[29] Hibernate propose d'effectuer le mapping en utilisant des annotations EJB/JPA. Dans ce cas, les annotations sont présentes dans la classe.

```
  </property>
  <set cascade="all" fetch="join" inverse="true" lazy="false"
   name="rights" sort="unsorted" table="right">
   <key>
    <column name="id_stis" not-null="true"/>
   </key>
   <one-to-many class="org.STeamer.genghis2.model.Right"/>
  </set>
  <property generated="never" lazy="false" name="comments" type="text">
   <column name="comments"/>
  </property>
  <property generated="never" lazy="false" name="urlStis" type="string">
   <column name="urlStis"/>
  </property>
 </class>
</hibernate-mapping>

<hibernate-mapping package="org.STeamer.genghis2.model">

...

</hibernate-mapping>
```

4

Les règles de correspondances indiquant comment faire persister et charger un objet d'une classe d'entité sont encadrées par une balise `<hibernate-mapping>`, un seul fichier peut donc contenir les descriptions pour plusieurs classes d'entités. Nous indiquons ensuite la classe d'entités persistantes et la table associée, dans l'exemple ci-dessus la classe `Stis` est associée à la table `STIS` (1). Nous indiquons le mapping de la propriété de l'identifiant unique vers la clé primaire des tables. De plus, comme nous ne voulons pas nous occuper de la gestion de cet identifiant, nous utilisons une stratégie de génération automatique pour la colonne de la clé primaire subrogée (2). Enfin, nous incluons des déclarations, pour les propriétés persistantes de la classe, dans le fichier de mapping (3), auquel nous rajoutons les associations (4). Dans le cas de l'association de la table `STIS` vers la table `RIGHT`, nous avons une relation « un à plusieurs » (*one to many*).

La clé primaire de la table *RIGHT* est constituée de trois champs dont deux sont soumis à une contrainte de clé étrangère : l'identifiant de l'utilisateur, l'identifiant du SIST et le type de droit (cf. Figure 54). Nous avons choisi une gestion externe de cette clé composée pour des raisons de facilité de gestion, d'évolution et de maintenance. Pour ce faire, nous avons externalisé les propriétés mappées aux champs composant la clé primaire.
La classe RightId se charge de le faire :

```
public class RightId implements java.io.Serializable {
      private String right;
      private long idUser;
      private long idStis;

      public RightId() {
      }
      public RightId(String right, long idUser, long idStis) {
            this.right = right;
            this.idUser = idUser;
            this.idStis = idStis;
      }
…
}
```

La classe *Right* subit une simplification par laquelle les trois anciennes propriétés sont remplacées par une référence à cette classe *RightId*.

Sous Hibernate, le chargement d'objets depuis la base de données, ou la mise à jour de celle-ci avec le nouvel état d'objets "*mappés*" (c'est-à-dire l'instance d'une classe d'entités persistantes), ne peut se faire que via une session active. L'activité d'une session est délimitée par les instructions:

- `org.hibernate.Session session = sessionFactory.openSession()`
- `session.close ()`

Les objets mappés peuvent être dans trois états différents :

- **Persistant** (*persistent*): cet état n'est possible que lorsqu'une session est ouverte, il correspond au fait qu'Hibernate s'occupe complètement de la persistance de l'objet (dès qu'une propriété de l'objet est modifiée la BDD est mise à jour et inversement). L'état de l'objet devient détaché dès que la session est fermée.
- **Détaché** (*detached*): précédemment persistant, mais actuellement non associé à une `Session`, car la session a été fermée (`close()`) (la modification de l'objet n'est pas répercutée au niveau de la BDD et inversement).
- **Ephémère** (*transient*): associé avec aucune `Session`, c'est l'état d'un objet qui vient juste d'être créé par `new`, que ce soit en dehors où dans une session.

Comme le montre la figure suivante (cf. Figure 55), les changements d'état d'un objet mappé se font au sein d'une session.

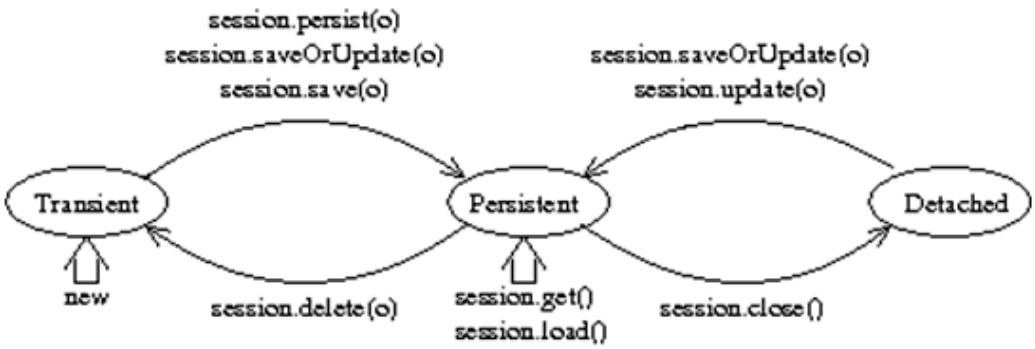


Figure 55 : Transitions d'état possibles durant une session Hibernate

- un objet instancié directement avec new est éphémère : il n'est associé à aucune session. Pendant une session, il devient persistant suite aux opérations : `session.save(obj)`, `session.persist(obj)` ou `session.saveOrUpdate(obj)`.
- un nouvel objet obtenu, pendant une session, depuis la base de données par `session.get()` ou `session.load()` est persistant. Il devient éphémère avec `session.delete(obj)`. Il devient détaché lors de la fermeture de la session `(session.close())`.
- Un ancien objet persistant pendant une session précédente, est initialement détaché lors d'une nouvelle session. Il devient persistant avec `session.update(obj)`, `session.saveOrUpdate(obj)`, `session.lock(obj)`.

Il est donc nécessaire de tenir compte des sessions et des différents états inhérents à Hibernate lorsque l'on manipule les données. Ces opérations sont prises en charge par la couche d'accès aux données présentée dans la section qui suit.

5.3.3. La couche d'accès aux données

Les objets en mémoire vive sont la plupart du temps liés à des données stockées en base de données, dans des fichiers, dans des annuaires,…. Ces données sont persistantes. Ce code de persistance varie en fonction du type de support et/ou des différentes implémentations des fournisseurs de SGBD. De facto, il est difficile de changer de support de persistance en cas d'imbrication du code de persistance et du code métier.
Un objet d'accès aux données (*Data Access Object* DAO) est un patron de conception qui propose de regrouper les accès aux données persistantes dans des classes à part, plutôt que de les disperser dans le code métier. Il vient en complément du modèle MVC (Modèle – Vue – Contrôleur) qui préconise la séparation des classes correspondant aux différentes problématiques : vues, modèle et contrôleurs. Nous séparons les entrées-sorties des classes métier et nous utilisons un objet DAO pour abstraire et encapsuler les accès aux données.

Dans le projet GenGHIS-Web, la couche de persistance est assurée par l'API JPA et son implémentation Hibernate (voir section précédente). Nous avons regroupé dans le package `org.STeamer.genghis2.dao` les classes liées à la couche d'accès aux données. Ainsi, à chaque entité de la couche de persistance correspond :

- une interface : qui décrit les méthodes de manipulations de l'entité dont le programme a besoin (par exemple l'interface IRightDAO pour l'entité Right).
- une classe qui implémente ces méthodes en utilisant Hibernate.

Grâce à cette décomposition, la couche d'accès aux données propose une interface indépendante du système de gestion de persistance, interface qui sera manipulée par les couches supérieures de l'application. Une modification au niveau de la couche de persistance n'impactera ainsi pas les couches de niveau supérieur.

Les tableaux ci-dessous (cf. Tableau 22) explicitent les méthodes des interfaces des entités de DAO de l'application GenGHIS-Web.

Interface	**Méthodes**	**Description**
ICdmDAO	findallCdm()	Renvoie la liste de tous les modèles conceptuels de données (MCD).
	returnCdm(int cdmId)	Retourne le MCD correspondant à un id.
	persistCdm(Cdm entity)	Rend persistant le MCD.
	deleteCdm(Cdm entity)	Efface le MCD.

	getCdm(String nameCdm)	Renvoie le MCD correspondant au nom fourni en paramètre.
IPmDAO	findallCdm()	Renvoie la liste de tous les modèles de présentations (MP).
	returnPm(int pmId)	Retourne le MP correspondant à un id.
	persistPm(Pm entity)	Rend persistant le MP.
	deletePm(Pm entity)	Efface le MP.
	getPm(String namePm)	Renvoie le MP correspondant au nom fourni en paramètre.
IStisDAO	findAll()	Renvoie la liste de tous les Systèmes d'Information Spatio-Temporelle.
	makePersistent(Stis entity)	Enregistre le SIST en base de données.
	newStis(Stis entity)	Crée un nouveau SIST.
	findPublicStis()	Retourne tous les SIST publics.
	findRestrictedStis()	Retourne tous les SIST à accès restreints.
	findMyStis(int idUser)	Retourne les SIST d'un utilisateur.
	getFromId(int id)	Retourne le SIST correspondant à un id.
	deleteStis(int id)	Efface le SIST dont l'id est fourni en paramètre.
IUserDAO	getFromLogin(String login)	Renvoie l'utilisateur possédant le login passé en paramètre.
	makePersistent(User entity)	Enregistre l'utilisateur passé en paramètre.
	findAll()	Renvoie tous les utilisateurs
	findUserUpgrade()	Renvoie les utilisateurs qui ont procédé à une demande d'élévation de leur compte.
	deleteUser(User entity)	Supprime l'utilisateur passé en paramètre.
	findUserStis(int idStis)	Retourne la liste des utilisateurs d'un SIST.

Tableau 22 : Méthodes des interfaces des entités de DAO

Toutes ces méthodes utilisent le langage d'interrogation d'Hibernate, afin de requêter la base de données. Le langage HQL (Hibernate Query Language) possède une syntaxe proche du SQL[30] et permet d'écrire des requêtes dans un langage simplifié et adapté à la manipulation orientée objet. Dans les requêtes HQL, il n'y a aucune référence aux tables ou aux champs car ceux-ci sont implicitement référencés respectivement par la classe et les propriétés des objets impliqués. C'est Hibernate qui se charge de générer la requête SQL à partir de la requête HQL, en tenant compte du contexte (la configuration du mapping).

L'exemple ci-dessous montre une requête HQL qui interroge la table USER de la base de données en sélectionnant un utilisateur dont le nom et le mot de passe sont passés en paramètre.

```
            Query q = s.createQuery("from User u where u.login= :login and
u.pass= :pass").setString("login", login).setString("pass", password);
```

Un objet de type Query est obtenu en invoquant la méthode createQuery() de la classe `Session` avec comme paramètre la requête HQL.
Dans cette requête, les paramètres sont précisés avec un identifiant précédé du caractère « : » (ici `:login` et `:pass`). L'interface Query propose de nombreuses méthodes setXXX() pour associer à chaque paramètre une valeur, en fonction du type de la valeur (XXX représente le type). Chacune de ces méthodes possède deux surcharges permettant de préciser le paramètre (à partir de son nom ou de son index dans la requête) et sa valeur. Un alias nommé « u » est défini pour la classe User.

5.3.4. La couche services

La couche de services fait le lien entre la couche d'accès aux données et les couches supérieures. Elle va donc être chargée, par exemple, lors de la connexion d'un utilisateur, de vérifier que le nom d'utilisateur et le mot de passe saisis, correspondent bien à un utilisateur existant dans la base de données. Elle sera également chargée de demander à la couche DAO, de renvoyer la liste de tous les utilisateurs enregistrés, lorsque nous voudrons les afficher.

5.3.5. La couche JSF.

5.3.5.1. Structuration et fichiers de configuration

Une application JSF est similaire à une application internet classique mise en œuvre par un conteneur de servlets car elle doit obligatoirement respecter les spécifications relatives aux servlets et aux pages JSP.

[30] **SQL** (*Structured Query Language*) est un langage informatique normalisé qui sert à effectuer des opérations sur des bases de données.

La figure suivante montre la structure de l'application GenGHIS-Web, telle qu'elle se présente dans l'explorateur de projets de l'environnement de développement Eclipse.

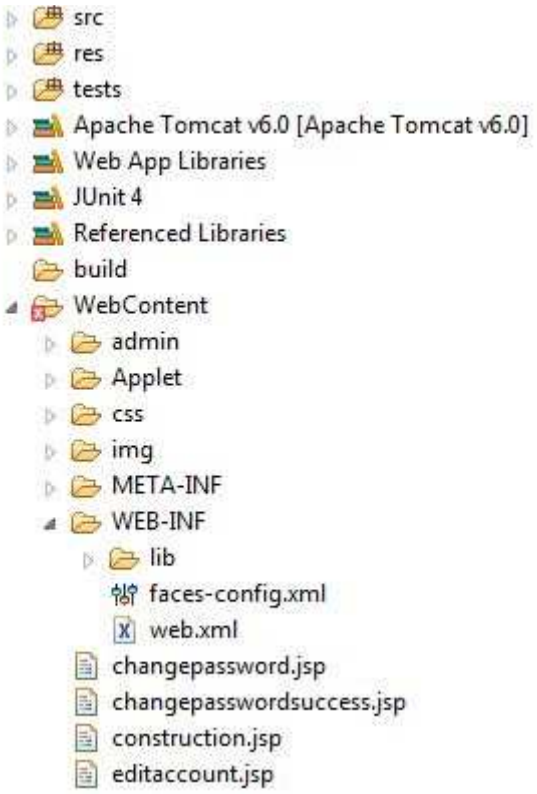


Figure 56 : Arborescence de l'application GenGHIS-Web

L'arborescence du projet (cf. Figure 56) est composée de plusieurs dossiers dont les principaux sont :

src : ce dossier contient le code source des classes Java utilisées par l'application.

res : ce dossier contient toutes les ressources employées dans l'application, telles que les images.

tests : ce dossier regroupe les classes de tests.

build : ce dossier regroupe le code compilé des classes Java du répertoire src.

WebContent : ce dernier dossier contient les ressources destinées à être publiées par le moteur de servlets (Tomcat) : les pages JSP/JSF, les styles CSS, les images.... Son contenu respecte la spécification JEE pour les applications Web. En particulier, il contient un répertoire WEB-INF où se trouvent les fichiers de configuration de l'application web, à savoir le descripteur de déploiement (`web.xml`), ainsi que le fichier de configuration des ressources de l'application (`faces-config.xml`).

Le code source ci-dessous est un extrait du fichier de configuration de ressource `faces-config.xml` de GenGHIS-Web. Ce fichier XML contient une balise racine `<faces-config>` encadrant d'autres balises enfants représentant les ressources gérées. Dans l'extrait donné ici, sont déclarés des beans managés (1), des règles de navigation (2) et un

validateur personnalisé (3). Ces notions sont étudiées plus en détail dans la suite de ce chapitre.

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
     http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
    version="2.0">
     <managed-bean>
          <managed-bean-name>loginForm</managed-bean-name>
          <managed-bean-class>org.STeamer.genghis2.servlets.LoginForm
          </managed-bean-class>
          <managed-bean-scope>session</managed-bean-scope>
     </managed-bean>
     <managed-bean>
          <managed-bean-name>userForm</managed-bean-name>
          <managed-bean-class>org.STeamer.genghis2.servlets.UserForm
          </managed-bean-class>
          <managed-bean-scope>request</managed-bean-scope>
     </managed-bean>
     <navigation-rule>
          <display-name>login</display-name>
          <from-view-id>/login.jsp</from-view-id>
          <navigation-case>
               <from-action>#{LoginForm.CheckValidUser}</from-action>
               <from-outcome>failure</from-outcome>
               <to-view-id>/login.jsp</to-view-id>
          </navigation-case>
     </navigation-rule>
     <navigation-rule>
          <display-name>login</display-name>
          <from-view-id>/login.jsp</from-view-id>
          <navigation-case>
               <from-action>#{LoginForm.CheckValidUser}</from-action>
               <from-outcome>success</from-outcome>
               <to-view-id>/homepage.jsp</to-view-id>
          </navigation-case>
     </navigation-rule>
     <navigation-rule>
          <display-name>register</display-name>
          <from-view-id>/register.jsp</from-view-id>
          <navigation-case>
               <from-action>#{userForm.NewUser}</from-action>
               <from-outcome>failure</from-outcome>
               <to-view-id>/register.jsp</to-view-id>
          </navigation-case>
     </navigation-rule>
     <validator>
          <validator-id>PasswordValidator</validator-id>
          <validator-
class>org.STeamer.genghis2.validator.PasswordValidator
           </validator-class>
     </validator>
…
</faces-config>
```

1

2

3

5.3.5.2. Les beans managés

Une application JSF permet une gestion des beans[31] s'exécutant côté serveur. Dans le cas où ils sont associés avec des composants graphiques présentés dans les formulaires web, ils sont qualifiés de beans managés : c'est le moteur d'exécution qui se charge de la gestion de leur cycle de vie. Ces beans managés prennent en charge les actions suivantes :

- la validation de la donnée saisie dans un composant graphique ;
- la prise en charge des événements générés par un composant ;
- la réalisation du traitement permettant de définir vers quelle page la navigation doit se poursuivre après la validation du formulaire.

Chaque bean managé doit être déclaré dans le fichier de configuration `faces-config.xml`(cf. section précédente).

La balise `<managed-bean-name>` attribue une clé (un identificateur) au bean. Cette clé valable sur la portée prévue pour le bean managé, pourra être utilisée dans toutes les pages web (pages JSP) de l'application pour référencer le bean managé.
La balise `<managed-bean-class>` indique le nom de la classe Java à partir de laquelle doit être instancié le bean managé.
Enfin, la balise `<managed-bean-scope>` précise la portée dans laquelle le bean managé est instancié. Elle peut prendre comme valeur : *page*, *request*, *session* ou *application*. Un bean de portée *page* à une durée de vie correspondant au traitement de la page où il est utilisé (il sera réinstancié chaque fois que l'application reviendra sur cette page), une portée *request* signifie que son existence est liée au traitement de la requête http, une portée *session* lie le bean à la session de l'utilisateur (une fois instancié il est stocké dans la session, ensuite lorsqu'il est référencé dans une page, JSF ira le chercher dans la session). Pour finir, la portée *application* lie le bean non plus à la session de l'utilisateur, mais au contexte de l'application (le bean une fois instancié est accessible à toutes les sessions en cours).

Le tableau ci-dessous (cf. Tableau 23) récapitule la liste des beans managés de l'application GenGHIS-Web, ainsi que leur rôle respectif :

Nom du bean managé	Rôle
`loginForm`	Gère l'authentification et la déconnection sur le portail web.
`pendingRequest`	Gère les demandes en attente des utilisateurs (demande de création de compte, mise à niveau du compte,..).
`manageUsers`	Utilisé par l'administrateur ; il gère l'ensemble des utilisateurs.
`servletStis`	Gère les SIST d'un utilisateur.

[31] Un bean ou Java Bean est un objet Java dont la classe respecte un certain nombre de conventions d'écriture : méthodes get… et set… pour la consultation et modification des propriétés, constructeur sans paramètres… Grâce à ces conventions et aux mécanismes d'introspection du langage Java, ces objets peuvent être ensuite manipulés sans que le code les manipulant ait besoin de connaître leur classe effective.

newStis	Permet de créer un nouveau SIST.
publicStis	Gère les SIST publics.
mgtUserStis	Gère les droits accordés aux utilisateurs des SIST.
restrictedStis	Gère les SIST à accès restreint.
fileUploadBeanCdm	Gère les modèles de données.
fileUploadBeanPm	Gère les modèles de présentations.
modifyStis	Permet de modifier un SIST.
userForm	Gère l'enregistrement et la modification des données sur les utilisateurs.

Tableau 23 : Liste des beans managés de GenGHIS-Web

Un bean managé s'exécutant côté serveur doit disposer d'un constructeur sans argument et doit également définir autant de propriétés que de champs de saisie auxquels il peut potentiellement être lié. Ces propriétés doivent être de portée privée, et disposer d'accesseurs en lecture et en écriture, pour rendre leur accès possible depuis les pages web de l'application. Tous les beans managés de GenGHIS-Web respectent ces conventions, comme par exemple le bean `userForm` qui permet, entre autres, à un utilisateur de s'enregistrer sur le site. Les propriétés du bean managé sont liées aux composants graphiques (cf. Figure 57) :

Figure 57 : Correspondance entre les composants graphiques et les propriétés du bean userForm

5.3.5.3. Validation de données

Avec Java Server Faces, des objets spécifiques, nommés validateurs, ont pour rôle la validation des données saisies dans un formulaire. Un validateur est obligatoirement associé à

un composant graphique JSF et implémente l'interface Validator qui définit une méthode `validate(…)`. Lors de la soumission d'un formulaire contenant le composant graphique auquel le validateur est associé, cette méthode est automatiquement invoquée (cela correspond à la phase de validation du cycle de vie d'une page JSF) dans le but de confirmer (ou infirmer) la valeur mentionnée dans le composant graphique associé.

Dans ce qui suit nous donnons un exemple de validateur personnalisé que nous avons mis en place dans GenGHIS-Web. Utilisé sur la page de demande de création de compte, il vérifie la syntaxe de l'adresse du courrier électronique renseigné qui doit être sous la forme n'importequoi@quelquechose.deslettres. Un autre validateur est utilisé également sur cette page pour vérifier si les saisies du mot de passe correspondent.

```
public class EmailValidator implements Validator {

      //La methode validate est appelée durant la phase de validation
      public void validate(FacesContext facesContext,
        UIComponent uIComponent, Object object)throws ValidatorException {

      //Déclaration et récupération de l'adresse e-mail renseignée
        String enteredEmail = (String)object;

         //Définition d'une expression régulière décrivant une adresse
         // mail syntaxiquement valide
         Pattern p = Pattern.compile(".+@.+\\.[a-z]+");

         //On vérifie la correspondance entre l'e-mail et
         // l'expression régulière
         Matcher m = p.matcher(enteredEmail);
         boolean matchFound = m.matches();

         // Dans le cas où il n'y a pas de correspondance, on définit le
        // message à communiquer au client et on crée une exception.
         if (!matchFound) {
              FacesMessage message = new FacesMessage();
              message.setDetail("Email not valid");
              message.setSummary("Email not valid");
              message.setSeverity(FacesMessage.SEVERITY_ERROR);
              throw new ValidatorException(message);
         }
      }
}
```

Chaque validateur est déclaré dans le fichier de configuration `faces-config.xml`. Ainsi, durant la phase de validation, JSF vérifie la présence ou non d'une `ValidatorException`. Si c'est le cas, JSF renvoie au client la même page agrémentée d'un ou plusieurs messages. Dans le cas contraire, JSF passe à la phase suivante.

5.3.5.4. Navigation dynamique

Java Server Face propose un modèle de navigation permettant de définir rapidement et facilement l'ordre de succession des pages visitées. Ce modèle de constitution souple, garantit

l'aisance de modification de la navigation entre les pages en cas d'évolution de la logique métier de l'application ou de la structure du site elle-même.

Le code source ci-dessous est un extrait du fichier faces-config.xml correspondant à la navigation de la page d'authentification.

```
<navigation-rule>
      <display-name>login</display-name>
      <from-view-id>/login.jsp</from-view-id>
      <navigation-case>
            <from-outcome>failure</from-outcome>
            <to-view-id>/login.jsp</to-view-id>
            <from-action>#{loginForm.checkValidUser}</from-action>
      </navigation-case>
</navigation-rule>
<navigation-rule>
      <display-name>login</display-name>
      <from-view-id>/login.jsp</from-view-id>
      <navigation-case>
            <from-outcome>success</from-outcome>
            <to-view-id>/homepage.jsp</to-view-id>
            <from-action>#{loginForm.checkValidUser}</from-action>
      </navigation-case>
</navigation-rule>
```

Ce code source présente deux règles de navigation, définies grâce aux balises `<navigation-rule>`. Pour chacune de ces règles sont précisées les informations suivantes :

- Le nom de la page web à partir de laquelle l'internaute provoquera la navigation : cette page source est mentionnée par l'intermédiaire de la balise `<from-view-id>`.

- La description du cas de navigation, caractérisé par une balise `<navigation-case>` et trois balises enfants, `<from-outcome>`, `<to-view-id>` et `<from-action>`.
La première de ces balises enfants (balise `<from-outcome>`) encadre une chaîne de caractères identifiant le nom du cas de navigation en tant que tel. C'est cette chaîne qui est recherchée au moment où l'internaute active un bouton, ou un lien hypertexte, pour identifier la règle de navigation à exploiter. Le choix de la chaîne des caractères identifiant un cas de navigation spécifique (balise `<from-outcome>` peut être fait librement (20). Il existe cependant quelques valeurs communément utilisées dans les applications web :
  - **success** : employé lorsque toutes les vérifications sont réalisées avec succès.
  - **failure** : employé lorsqu'une erreur au moins est identifiée par le processus métier chargé de réaliser les contrôles. La navigation est alors renvoyée vers une page d'erreur.

La deuxième balise enfant, `<to-view-id>`, précise le nom de la page web vers laquelle la navigation est orientée en cas d'utilisation du cas de navigation.

La troisième balise enfant, `<from-action>` indique un bean managé qui est chargé de générer dynamiquement la chaîne identifiant le cas de navigation adapté au contexte. Concrètement, c'est une méthode spécifique qui est identifiée pour réaliser les traitements nécessaires et renvoyer le cas de navigation attendu.

L'exemple de code source présenté plus haut montre que les deux règles de navigation sont applicables à partir d'une unique page web source nommée « login.jsp ». La navigation est rendue possible depuis cette page, vers la page « homepage.jsp » lorsque le cas de navigation « *success* » est utilisé, ou vers la page « login.jsp » si le cas de navigation « *failure* » est employé. Pour cela nous faisons appel à la méthode `checkValidUser` du bean managé `loginForm` dont un extrait du code se trouve ci-dessous. Cette méthode vérifie si le couple nom/mot de passe renseigné par l'utilisateur est présent dans la table USER de la base de données. Si ce couple existe, la méthode renvoie « *success* » (dans ce cas la navigation continue vers la page homepage.jsp) sinon elle renvoie « *failure* » (la navigation retourne sur la page login.jsp accompagnée d'un message d'erreur).

```
public String checkValidUser()throws Exception {
      UserServices uServices = new UserServices();

      // Si le couple nom d'utilisateur/ mot de passe existe
      // on renvoie success
      if (uServices.connectUser(userName, password) != null) {
            returnString = "success";
      }
      //Sinon on renvoie failure et un message explicite.
      else {
            setText("User Name or Password is incorrect.");
            returnString = "failure";
      }
}
```

Sous l'environnement de développement Eclipse, ces deux cas de navigation peuvent être visualisés graphiquement, comme le montre la figure ci-dessous (cf. Figure 58).

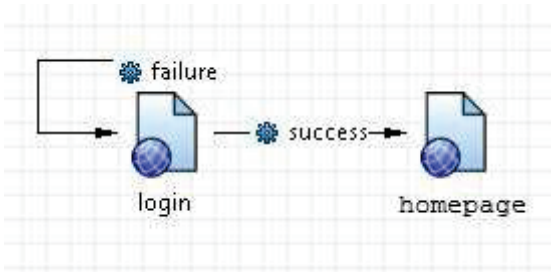


Figure 58 : Visualisation graphique de cas de navigation

5.3.5.5. Composants graphiques

Comme nous l'avons indiqué lors de la présentation des technologies JSF, nous utilisons la bibliothèque de composants graphiques RichFaces. Parmi les très nombreux composants proposés par cette bibliothèque, ceux que nous avons le plus utilisé dans l'application GenGHIS-Web sont ceux permettant l'itération de données. Nous avons, en particulier, employé deux composants : le `DataTable` et l'`ExtendedDataTable`.

- Le composant `DataTable`

`DataTable` permet de représenter un tableau de données. Il offre la possibilité d'indiquer une ligne d'en-tête ainsi qu'un filtre sur les colonnes, permettant de trier celles-ci.

Concrètement, lors de l'affichage de la page, le composant `DataTable` envoie une requête à un bean managé en lui indiquant de retourner dans une variable un objet contenant les données à afficher. Pour l'affichage, nous indiquons au composant de mettre tel attribut de la variable dans telle ligne de telle colonne.

La figure suivante (cf. Figure 59) est une capture d'écran de la page énumérant les SIST publics. Le composant DataTable permet d'itérer dans un tableau les champs `nom` et `description` des SIST.

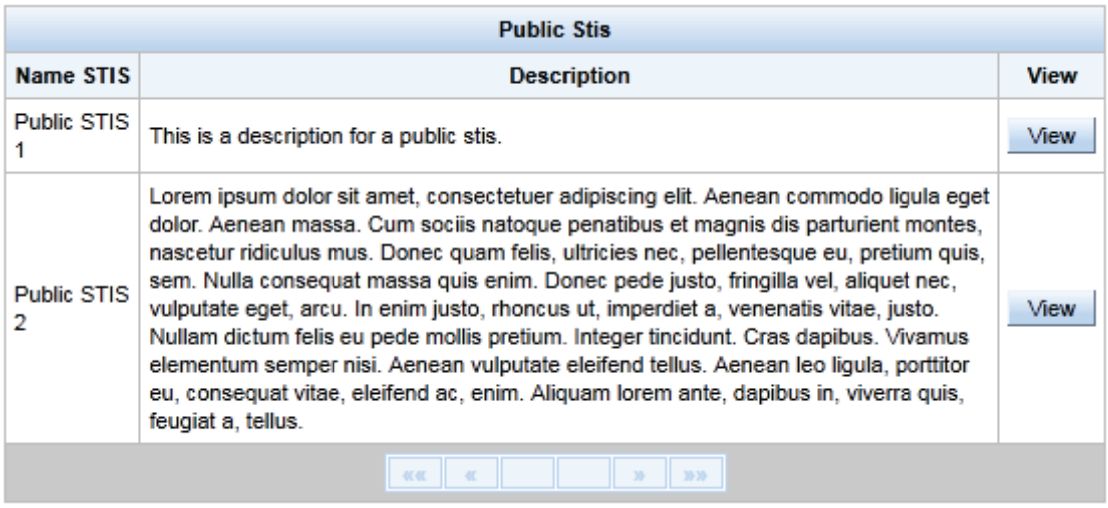


Figure 59 : Capture d'écran du composant richFace DataTable

- Le composant `ExtendedDataTable`

`ExtendedDataTable` fonctionne de la même façon que `DataTable`, a ceci près qu'il offre la possibilité d'affiner le résultat du tableau de données en réémettant une requête au serveur pour mettre à jour l'extension du tableau.

La figure suivante (cf. Figure 60) est une capture d'écran de la page énumérant les SIST d'un utilisateur. Le composant `ExtendedDataTable` permet d'itérer dans un premier tableau les champs `nom` et `privilège` des SIST. Si l'utilisateur sélectionne l'un des SIST, une requête est envoyée au serveur afin d'afficher les boutons qui correspondent aux actions autorisées par son statut.

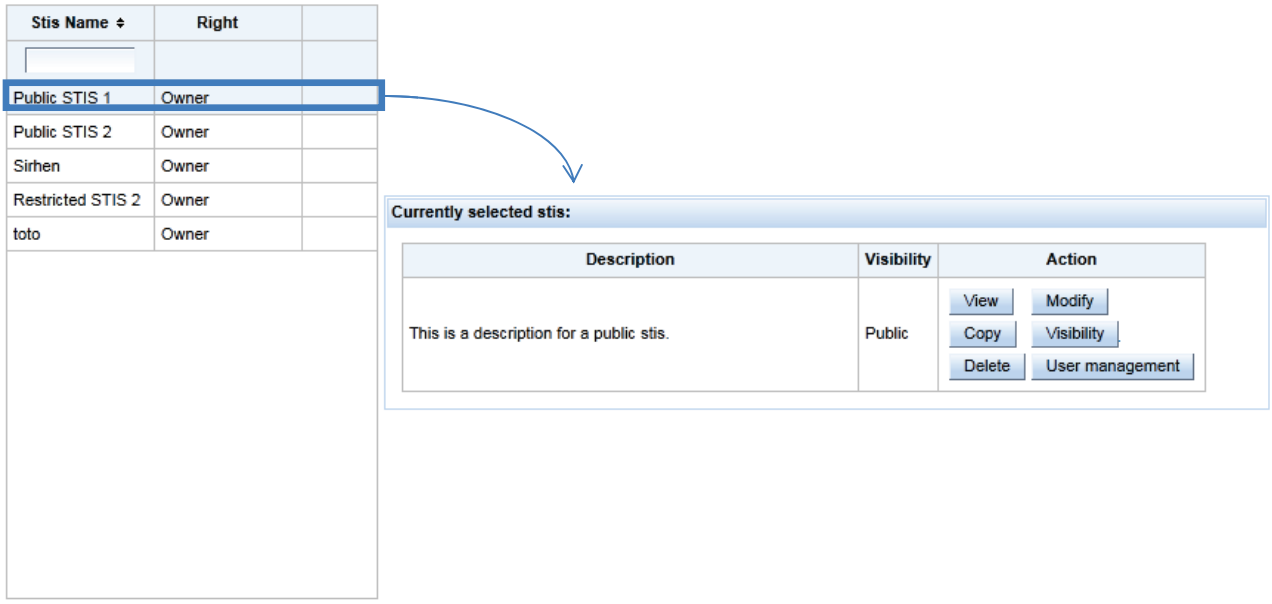


Figure 60 : Capture d'écran d'un composant ExtendedDataTable

Dans l'exemple ci-dessus, l'utilisateur est le propriétaire de tous les SIST de la liste. En sélectionnant le SIST dénommé « Public STIS 1 », la fenêtre est construite dynamiquement et s'affiche. Notre utilisateur étant le propriétaire du SIST, il peut :

- voir le SIST (bouton *View*) ;
- modifier le SIST (bouton *Modify*) ;
- copier le SIST (bouton *Copy*) ;
- modifier la visibilité du SIST (bouton *Visibility*) ;
- supprimer le SIST (bouton *Delete*) ;
- gérer les accès des utilisateurs sur ce SIST (bouton *User Management*).

Nous avons utilisé d'autres composants graphiques RichFaces que nous énumérons dans le tableau ci-dessous, accompagnés de leur description.

Composant graphique	Description
<rich:dropDownMenu>	Ce composant est utilisé pour la barre de menus de GenGHIS-Web. Il organise un menu hiérarchiquement de façon semblable à celui d'une application traditionnelle.
<rich:modalPanel>	Une fenêtre contextuelle apparaît au premier plan de l'écran bloquant ainsi les opérations sur la page parente.
<rich:inplaceInput>	Cette balise représente un champ de saisie de type texte qui affiche une valeur prédéfinie.
<rich:fileUpload>	Ce composant fournit des fonctionnalités pour le chargement et le transfert de fichiers sur le serveur.

5.3.6. Intégration de l'application GenGHIS

Comme nous l'avons présenté dans les cas d'utilisation au chapitre 4, l'application GenGHIS-WEB doit proposer à un utilisateur concepteur des fonctionnalités similaires à l'application GenGHIS présentée au chapitre 3. Dans cette section nous présentons la manière dont nous l'avons intégrée au portail GenGHIS Web pour la rendre accessible aux utilisateurs de ce dernier.

5.3.6.1. Les solutions disponibles

L'étude de l'existant nous a montré que GenGHIS est une application autonome (ou *stand-alone*). Son utilisation nécessite l'installation de l'application sur le poste de l'utilisateur, et le traitement des données est local. Tout en conservant au maximum le code existant, nous devions transformer GenGHIS en environnement client-serveur. L'interface de l'application doit s'exécuter dans le navigateur du client mais les données sont conservées au niveau du serveur GenGHIS-Web.

L'environnement client-serveur désigne un mode de communication organisé par l'intermédiaire d'un réseau et d'une interface internet entre plusieurs ordinateurs. Parmi les types de clients existants (client lourd, client léger, client riche) nous avons choisi le client riche. Le client riche est une interface graphique plus évoluée qui permet de mettre en œuvre des fonctionnalités comparables à celles d'un client "lourd". Les traitements sont effectués majoritairement sur le serveur, la réponse "semi-finie" étant envoyée au poste client, où le client "riche" est capable de la finaliser et de la présenter.

Dans cette optique de client riche, nous nous sommes penchés sur trois solutions :

Java Web Start
Développée avec la plate-forme Java 2, la technologie Java Web Start (JWS) permet le déploiement d'applications lourdes à travers le web. Une application JWS peut se lancer de trois manières distinctes :
- à partir d'un navigateur internet, en cliquant sur un lien ;
- à partir de raccourcis (icones) présents sur le poste de l'utilisateur ;
- à partir du gestionnaire d'applications intégré de Java Web Start (ce gestionnaire assure le suivi des applications récemment utilisées et permet d'accéder rapidement à celles-ci).

Lors du lancement de l'application, Java Web Start se connecte au serveur pour vérifier si une nouvelle version est disponible. Dans l'affirmative il télécharge celle-ci, sinon la dernière version téléchargée et installée localement, est exécutée. Ce système a pour avantages de permettre un accès direct à l'application, même en l'absence d'une connexion réseau, et de garantir la présence de la dernière version du logiciel (vérification faite au démarrage de JWS). Si elle est très facile à mettre en œuvre, cette solution ne convient néanmoins pas à notre projet car nous ne souhaitons plus que le poste client ait en charge le traitement métier.

Un client « Full Web »
Dans l'internet d'aujourd'hui, les applications riches sont réalisées en AJAX (*Asynchronous Javascript and XML*). Elles reposent sur les qualités intrinsèques des navigateurs du marché et s'appuient sur le langage JavaScript, les feuilles de styles CSS et le balisage XML. C'est une stratégie de type essentiellement « client/serveur ». La partie client est implémentée en

JavaScript. La partie serveur utilise une des plates-formes du marché (par exemple Java). Toute l'interface homme machine (présentation, logique) est hébergée par le navigateur. Le serveur n'exécute que les points de service dont le client a besoin (et qu'il peut invoquer de manière asynchrone). Si ces technologies permettent de réaliser une interface, directement affichable dans un navigateur web et offrant les mêmes fonctionnalités que celles de GenGHIS, cela nous obligerait à reprendre la quasi intégralité du code. Ceci n'étant pas compatible avec le temps consacré à ce projet, nous avons écarté cette solution.

Les applets Java

Une applet est une application Java intégrée dans une page HTML. Elle s'affiche à l'intérieur de la page dans la fenêtre du navigateur qui a téléchargé cette dernière. Une applet est fournie sous la forme de bytecode java[32], s'exécutant grâce à une machine virtuelle Java. Ce bytecode est multiplate-forme, les applets peuvent donc être exécutés sur différentes plates-formes telles que Windows, Linux, Mac OS, à la condition que le module d'extension nécessaire (Java Plugin) ait été installé avec le navigateur web.

Contrairement aux applications dites classiques, les programmes comportant des applets ne sont donc pas autonomes. Bien que les créations respectives d'applets et d'applications Java soient soumises à deux ensembles distincts de règles et de procédures, rien ne les oppose. La différence fondamentale entre applet et application est l'ensemble des restrictions auxquelles est soumise l'exploitation des applets pour des raisons de sécurité. Etant donné que les applets peuvent être chargées depuis n'importe quel site Web pour être exécutées sur un système client, un certain nombre de restrictions sont mises en place afin d'empêcher qu'une «applet malfaisante» n'inflige des dommages au système utilisateur ou n'en viole la sécurité. Les restrictions portent sur les points suivants :

- les applets ne peuvent ni lire, ni écrire sur le système de fichiers de l'utilisateur ;
- les applets ne peuvent communiquer avec aucun autre serveur que leur serveur d'origine ;
- les applets ne peuvent faire fonctionner aucun programme sur le système de l'utilisateur.

Introduites avant l'apparition des technologies AJAX, les applets permettaient de fournir, au sein d'applications web, des fonctionnalités interactives qui n'étaient pas disponibles avec le langage HTML. Si les technologies du Web 2.0 les ont quelque peu éclipsées, les applets restent une solution pour des développements en Java. L'application GenGHIS étant écrite en Java, sa transformation en applet permet de conserver un maximum du code existant, c'est pourquoi nous avons retenu cette solution pour l'intégration de cette application dans GenGHIS-Web.

5.3.6.2. Réingénierie de l'application GenGHIS

La technologie des applets ayant été choisie, il s'agit maintenant d'adapter l'existant de manière à le transformer sous forme d'applets qui seront intégrées aux pages de l'application GenGHIS-Web.

Un point qui a été mis en avant lors de la conception de GenGHIS-Web et de la définition des cas d'utilisation d'un utilisateur concepteur est le besoin de permettre outre la création, la

[32] Le bytecode Java est l'ensemble des instructions exécutables par une machine virtuelle java.

modification d'un SIST. La modification signifie que l'utilisateur doit pouvoir reprendre chacune des étapes du processus de construction d'un SIST. De cela, il découle que l'interface utilisateur de GenGHIS-Web devra être découpée en applets pouvant être exécutées indépendamment les unes des autres. Une seconde conséquence de cela est qu'il est apparu nécessaire de pouvoir mémoriser la correspondance Modèle de Données Brutes (MDB) / Modèle Conceptuel de Données (MCD) construite dans la première étape du module de données. En effet, nous souhaitons que l'utilisateur puisse reprendre la construction d'un SIST à partir de n'importe laquelle des étapes du processus de construction. De même, il nous semble important qu'un utilisateur puisse construire plusieurs SIST partageant les mêmes caractéristiques (MCD, MDB, MP, correspondances MCD<-> MDB et MCD <-> MP) exceptées les valeurs de données brutes. Dans la version actuelle cela n'est pas possible, si un utilisateur désire construire deux SIST pour des jeux de données similaires (par exemple les inondations dans le département de l'Isère et les inondations dans le département de la Drôme), il doit redéfinir à chaque fois la correspondance entre ses données et un modèle AROM.

Une autre constatation qui a été faite lors de l'analyse de l'application GenGHIS existante est la forte dépendance du code applicatif vis-à-vis d'AROM. En effet AROM et AROM-ST sont utilisés pour définir les modèles servant à l'intégration des données, modèles qui sont ensuite exploités lors de la phase de génération du code de l'application SIST. Bien qu'adoptant une architecture MVC, le code de l'interface graphique fait certains appels à l'API-AROM ce qui n'est plus souhaitable pour plusieurs raisons. Premièrement, cela nécessiterait de faire migrer les modèles AROM vers le poste client pour pouvoir les instancier, puis, de les renvoyer sur le serveur GenGHIS-Web pour les stocker. Deuxièmement, il est important d'isoler au maximum les dépendances vis-à-vis d'AROM en vue de pouvoir, plus tard, migrer vers un autre système pour la représentation des modèles. De cela résulte une seconde orientation qui a guidé notre travail de réingénierie de l'application GenGHIS : supprimer les dépendances entre le code de l'interface utilisateur pris en charge par les applets et le code de manipulation des modèles AROM que nous avons décidé d'exécuter sur le serveur.

Dans les sections suivantes, nous présentons plus en détail le travail de réingénierie effectué sur le module de données de GenGHIS.

5.3.6.3. L'association MDB <->MCD

Comme nous l'avons dit précédemment, dans GenGHIS, à chaque fois que l'utilisateur charge un fichier de données, il doit faire correspondre les entités du modèle conceptuel avec les attributs définis dans le modèle de données brutes. S'il charge un fichier contenant exactement la même structure que le précédent, il doit à nouveau établir les relations, qu'elles soient identiques ou non.

Afin de pallier cette problématique, nous avons amélioré l'application en explicitant l'association MDB<->MCD à l'aide de structures de données qui sont enregistrées sur le serveur. Ainsi, lorsque l'utilisateur voudra modifier un SIST l'association MDB<->MCD pourra être restaurée. L'utilisateur pourra alors la modifier et/ou la réutiliser pour des données différentes.

Nous avons conçu un modèle objet pour représenter l'association MDB <->MCD. Le diagramme de classes suivant (cf. Figure 61) présente les classes de ce modèle ainsi que les différentes relations entre celles-ci. Nous remarquons que ces classes ont des associations non

symétriques qui expriment des couplages forts et des relations de composition. Les cycles de vie des éléments (les "composants") et de l'agrégat sont liés : si l'agrégat est détruit, ses composants le sont aussi. A un même moment, une instance de composant ne peut être liée qu'à un seul agrégat. Si nous supprimons un *ModelMapping*, l'ensemble des *ConceptMappings* liés seront supprimés.

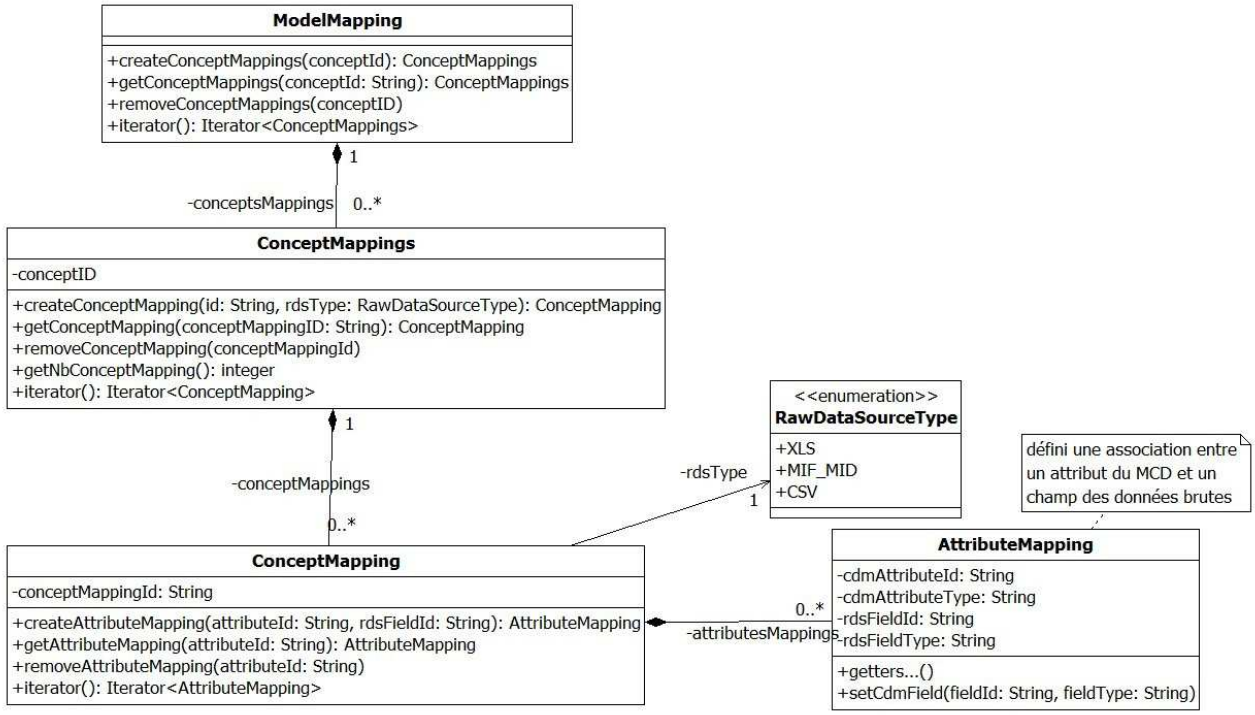


Figure 61 : Diagramme de classes de l'association MDB <-> MCD (21)

Pour l'enregistrement et la restitution d'un `Model Mapping`, nous utilisons la bibliothèque *XStream* qui permet de sérialiser et dé sérialiser des objets Java dans des fichiers XML. C'est donc dans un format XML, que l'association MDB <-> MCD d'un SIST est enregistrée sur le serveur GenGHIS-Web.

5.3.6.4. Séparation du module de données en applet client et code serveur.

L'analyse de l'existant nous a montré (cf. Figure 24) que l'entité *KB Handler* permet l'interaction avec AROM et que le contrôleur principal `ControlerGenGHIS` fait la correspondance entre le `KB Handler` et le contrôleur de modèle de données. Si nous voulons extraire la couche métier de notre applet, nous devons transférer ces deux composants sur le serveur et ne conserver dans l'applet que le code d'interface utilisateur. Il faut également que les échanges de données entre l'applet et les composants demeurant sur le serveur s'effectuent indépendamment d'AROM.

Différentes techniques permettent aux applets de communiquer avec des serveurs (connection par socket, servlet,...). L'une d'entre elles est l'appel de méthodes distantes (*Remote Method Invocation* ou RMI), c'est celle que nous avons choisie d'adopter pour le

module de données. Nous présentons les principes du RMI dans ce qui suit, puis nous détaillons ensuite comment il est mis en œuvre dans le module de données.

Principes du RMI

RMI (*Remote Method Invocation)* permet le développement d'applications réparties en Java en rendant possible l'invocation des méthodes d'un objet situé sur une machine distante. Un client et un serveur peuvent ainsi communiquer en utilisant un paradigme-objet de haut niveau : les requêtes deviennent des invocations de méthodes, les réponses sont reçues sous forme d'objets sérialisés (cf. Figure 62). La technologie RMI se charge de rendre transparents la localisation de l'objet distant, son appel et le renvoi du résultat.

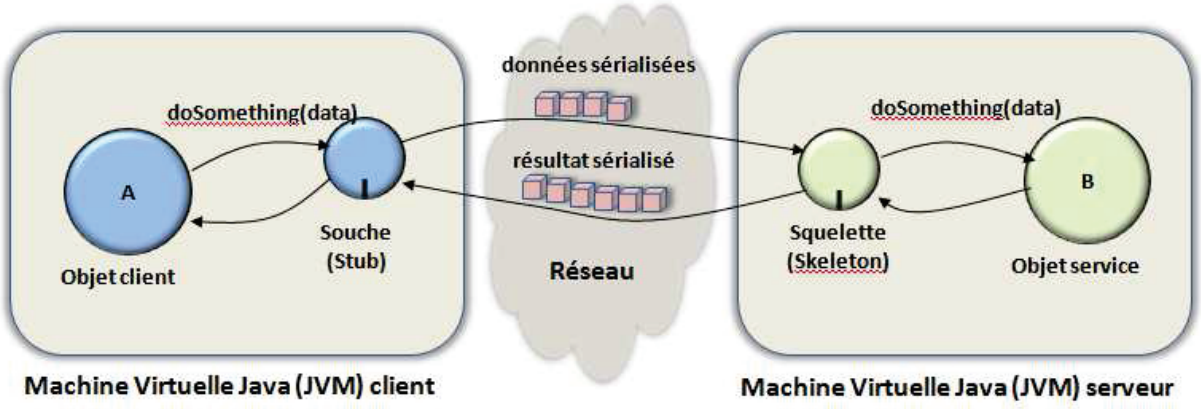


Figure 62 : Interaction entre un client et un serveur via RMI

Ainsi, si un objet instancié côté client veut accéder à des méthodes d'un objet distant, il doit en premier lieu, localiser l'objet distant grâce au registre RMI. Il en résulte une image virtuelle de l'objet distant : la souche (stub). Cette souche possède exactement la même interface que l'objet distant. La souche sérialise[33], en une suite d'octets, l'appel de la méthode distante puis les transmet au serveur. Le squelette (skeleton) « dé sérialise » les données envoyées par la souche, puis appelle en local la méthode de l'objet serveur. Le squelette récupère les données renvoyées par la méthode puis, les retransmet au client en effectuant le chemin inverse.

Une application client-serveur basée sur RMI met donc en œuvre trois composantes(22) :

- une application client implémentant la souche (*stub*) ;
- une application serveur implémentant le squelette (*skeleton*) ;
- une application médiatrice : le registre RMI.

Les connexions et les transferts de données dans RMI sont effectués par Java en se greffant sur TCP/IP[34] grâce au protocole normalisé par l'OMG[35] (*Object Management Group*) : RMI-IIOP (*Internet Inter-Orb Protocol*). La transmission des données entre la souche et le

[33] La sérialisation est un processus visant à encoder l'état d'une information qui est en mémoire sous la forme d'une suite d'informations plus petites(11).
[34] La suite TCP/IP est l'ensemble des protocoles utilisés pour le transfert des données sur Internet.
[35] L'Object Management Group est une association américaine dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes(11).

squelette s'effectue à travers un système de couches similaires au modèle OSI[36] (*Open Systems Interconnection*) afin de garantir une interopérabilité entre les programmes et les versions de Java. La figure ci-après (cf. Figure 63) détaille les couches utilisées.

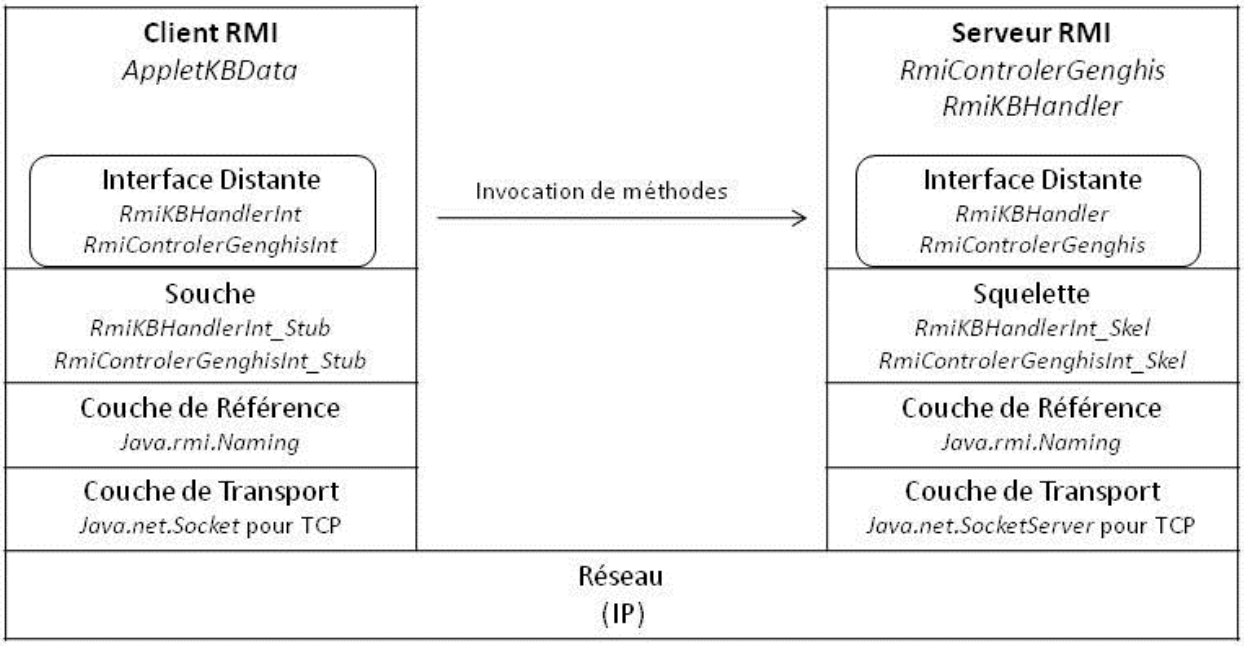


Figure 63 : Architecture de l'invocation des méthodes

La souche (*stub*) et le squelette (*skeleton*), respectivement sur le client et le serveur, assurent la conversion des communications avec l'objet distant.
La couche de référencement (appelée généralement registre RMI), est assurée par le package `java.rmi.naming`. Elle a en charge le système de localisation, afin de fournir aux objets un moyen d'obtenir une référence à l'objet distant.
La couche de transport, est assurée par le package `java.net.Socket` et `java.net.SocketServer`. Elle permet d'écouter les appels entrants ainsi que d'établir les connexions et le transport des données sur le réseau, par l'intermédiaire du protocole TCP.

Mise en œuvre de RMI côté serveur

RMI nécessite de définir une interface héritant de l'interface *java.rmi.Remote* et contenant toutes les méthodes susceptibles d'être appelées à distance. Ces méthodes doivent être en mesure de lever l'exception `java.rmi.RemoteException` dans le cas de l'échec de l'invocation.

Dans cette optique, nous avons inventorié toutes les méthodes utilisées dans le module de données afin de les regrouper dans deux interfaces : la première interface `RmiKBhandlerInt` correspond au `KB Handler` qui permet d'interagir avec les modèles de type AROM. Les méthodes de cette interface permettent de charger, sauver un modèle ou d'accéder aux noms des différentes entités d'un modèle. Seules des chaînes de caractères sont

[36] Le modèle OSI est un modèle de communications entre ordinateur.

échangées avec le client qui ne doit pas être dépendant du fait que le modèle utilise AROM ou une autre technologie. Un certain nombre de méthodes de `RmiKBhandlerInt` sont énumérées dans le code ci-dessous :

```
public interface RmiKBHandlerInt extends Remote{

      public static RmiKBHandler.TypeStruct STRUCT_CLASS = TypeStruct.CLASS;
      public static RmiKBHandler.TypeStruct STRUCT_ASSOCIATION =
          typeStruct.ASSOCIATION;
      public boolean readKB(File file) throws
          RemoteException,FileNotFoundException;
      public ArrayList<String> getSubStructureNames(String structName) throws
          RemoteException;
      public int getNbInstStructure(String structName) throws RemoteException;
      public boolean IsKbLoaded() throws RemoteException;

      …
}
```

La seconde interface correspond au `ControlerGenGHIS`, nous la nommons `RmiControlerGenghisInt`, l'intégralité de son code se trouve ci-dessous :

```
public interface RmiControlerGenghisInt extends Remote {

      /** Renvoie le modèle au contrôleur. Le type QueriesKBGenghis
       fournit à la fois le MDC et le MP **/
      public QueriesKBGenghis getModel() throws RemoteException;

      /** Communique l'emplacement du modèle de données et procède à sa
          lecture. **/
      public String ConfigureUrl(String url) throws
             RemoteException,FileNotFoundException;

      /** Récupère l'enregistrement de l'association MCD<-> MDB */
      public ModelMapping getModelMapping(String path) throws RemoteException,
             FileNotFoundException;

      /** Enregistre l'association MCD<-> MDB. **/
      public ModelMapping setModelMapping(ModelMapping mapping, String path)
             throws RemoteException, FileNotFoundException;

}
```

La seconde étape consiste en l'écriture des classes `RmiKBHandler` et `RmiControleurGenghis` correspondant aux objets s'exécutant sur le serveur et implémentant les interfaces précédentes. Ces classes doivent obligatoirement hériter de la classe `UnicastRemoteObject` qui contient les différents traitements élémentaires pour un objet distant, dont l'appel par un client unique. La souche ne peut obtenir qu'une seule référence sur un objet distant héritant de la classe `UnicastRemoteObject`.

La dernière étape est l'écriture de la classe `RmiServer` pour instancier les objets sur le serveur et les enregistrer dans le registre RMI. Nous créons un objet de la classe de l'objet distant, puis nous l'enregistrons dans le registre de noms en lui affectant un nom. L'enregistrement se fait en utilisant la méthode `rebind` de la classe `Naming`. Elle attend en paramètre le nom de l'objet et l'objet lui même. C'est ce nom qui sera utilisé dans une URL par le client, pour obtenir une référence sur l'objet distant. La classe `RmiServer` suivante possède deux objets ayant pour dénomination « RmiController » et « RmiKBHandler ».

```
public class RmiServer {

      public static void main(String[] args) throws RemoteException {
            try {
                  RmiControlerGenGHIS controler = new RmiControlerGenGHIS();
                  Naming.rebind("RmiControler", controler);

                  RmiKBHandlerInt kb = new RmiKBHandler();
                  Naming.rebind("RmiKBHandler", kb);

                  System.out.println("RMI Server OK");

            } catch (Exception e) {
                  e.printStackTrace();
            }

     }
```

Mise en œuvre de RMI côté client

Afin d'obtenir une référence sur l'objet distant à partir de son nom, RMI nécessite l'utilisation de la méthode statique `lookup()` de la classe `Naming`. Cette méthode attend en paramètre une URL indiquant le nom qui référence l'objet distant. Cette URL est composée de plusieurs éléments : le préfix `rmi://`, le nom du serveur (*hostname*) et le nom de l'objet tel qu'il a été enregistré dans le registre, précédé d'un slash. La méthode `lookup()` recherche dans le registre du serveur l'objet et retourne un objet `stub`. L'objet retourné étant de type `Remote`, il faut réaliser un typage vers l'interface qui définit les méthodes de l'objet distant. Pour plus de sécurité, on vérifie que l'objet retourné est bien une instance de cette interface(18).
Ci-dessous le code de la mise en œuvre de RMI dans la classe `AppletKBData` :

```
//Initialisation de RMI
obj = Naming.lookup("rmi://" + getCodeBase().getHost()+ ":1099/RmiControler");
controlerGenghis = (RmiControlerGenghisInt) obj;

obj = Naming.lookup("rmi://" + getCodeBase().getHost()
                            + ":1099/RmiKBHandler");
RmiKBHandler rmiKBHandler = null;
if (obj instanceof RmiKBHandlerInt) {
     rmiKBHandler = (RmiKBHandlerInt) obj;
     ...
}
```

Il suffit ensuite d'appeler les méthodes des classes `controlerGenghis` ou `RmiKBHandler`.

5.3.6.5. Le module de présentation

L'étude du code du module de présentation a révélé une forte imbrication entre la vue et la couche métier. L'extraction du code lié à AROM nécessitant une réingénierie trop

importante, nous n'avons pas pu l'effectuer dans le temps qui nous était imparti. En revanche, nous avons décidé d'avoir néanmoins le module de présentation fonctionnel dans GenGHIS-Web, pour cela il nous a fallu transformer le code existant pour l'intégration dans une applet.

GenGHIS est décomposé en plusieurs contrôleurs dont l'un, nommé `ControlerEditKBPresentation` (cf. Figure 24), s'occupe des flux de traitement de l'interface gestionnaire du modèle de présentations. La mise en place en applet a consisté à transformer le constructeur du contrôleur en une méthode propre aux applets Java appelée : `init()` qui est lancée lors du chargement ou du rechargement de l'applet. L'initialisation d'une applet inclut généralement la lecture des paramètres de celle-ci, la création d'objets auxiliaires dont elle peut avoir besoin, la mise à l'état initial, le chargement d'images, de polices,…

Après avoir créé la classe contenant l'applet et l'avoir compilée, nous l'intégrons à une page internet à l'aide d'une balise HTML. Nous précisons des paramètres à l'applet qui sont les liens vers le modèle de données et le modèle de présentations à charger.

5.3.7. Test

La réalisation d'un logiciel exige la manipulation d'un grand nombre de concepts et d'objets, ce qui génère de nombreuses confusions. Pour cette raison, lors de l'élaboration d'une application, nous mettons en place des tests afin de vérifier que les fonctionnalités font ce qu'on attend d'elles. Les tests d'une application sont une phase très importante dans les cycles de développement et de maintenance d'une application. Ils permettent de s'assurer que l'application répond au cahier des charges et aux spécifications, de détecter des bugs, de vérifier qu'il n'y a pas de régression lorsqu'une modification est apportée au code.

Ces tests peuvent prendre différentes formes (18) :

- tests unitaires : le test unitaire est un procédé permettant de s'assurer du fonctionnement correct d'une portion d'un programme. Les tests unitaires automatisés sont un des mécanismes les plus importants pour améliorer la qualité et tenter de garantir la fiabilité du code d'une application ;
- tests de recette : le but est de vérifier que l'application réponde aux spécifications fonctionnelles. Ces tests sont faits par les utilisateurs qui devraient fournir un procès-verbal de recette ;
- tests d'intégration : ils consistent, une fois que les développeurs ont chacun validé leurs développements ou leurs correctifs, à regrouper leurs modifications ensemble dans le cadre d'une livraison ;
- tests de charge (robustesse, performance, montée en charge, ...) ;
- tests d'acceptabilité : C'est la dernière étape avant l'acceptation du système et sa mise en œuvre opérationnelle. On teste le système dans les conditions définies par le futur utilisateur, plutôt que par le développeur ;
- …

Dans le cadre du projet, nous avons mis en place des tests unitaires dont la mise en œuvre est facilitée par l'utilisation d'outils tels que JUnit.

JUnit est un outil de gestion des tests unitaires pour les programmes Java. C'est une bibliothèque pour le développement et l'exécution de tests unitaires automatisables. Cette bibliothèque a été créée par Erich Gamma et Kent Beck. Son principal intérêt est de s'assurer que le code réponde toujours aux besoins, même après d'éventuelles modifications. JUnit fait partie de la famille des xUnit, descendant de SUnit, qui fut créé pour le langage SmallTalk. Les cas de tests sont regroupés dans des classes Java qui contiennent une ou plusieurs méthodes de tests. Ces cas de tests peuvent être regroupés sous la forme de suites de tests. Les cas de tests peuvent être exécutés individuellement ou sous la forme de suites de tests. JUnit permet le développement incrémental d'une suite de tests (18). Avec JUnit, l'unité de test est une classe dédiée qui regroupe des cas de tests.

Ces cas de tests exécutent les tâches suivantes :

- création d'une instance de la classe et de tout autre objet nécessaire aux tests
- appel de la méthode à tester avec les paramètres du cas de test
- comparaison du résultat attendu avec le résultat obtenu : en cas d'échec, une exception est levée

Le cadriciel JUnit a permis, dans le cadre du projet, de gérer les tests se référant aux développements des applets, mais aussi des couches de persistance, d'accès aux données et de service. En revanche, pour la mise en œuvre de JSF, nous avons pris le parti de rajouter un cadriciel de test propre à cette technologie : JSFUnit.

Les tests JSFUnit sont, comme tous les tests JUnits, des méthodes d'une classe Java. Dans le cas particulier de JSFUnit, il est nécessaire que cette classe hérite de `ServletTestCase` afin de lui faire prendre en compte les spécificités de JSF. JSFUnit permet de valider :

- les beans managés : Quel est l'état du bean à un instant donné ;
- la navigation : cette requête me renvoie-t-elle à la bonne page ;
- les composants graphiques : est-ce que l'arbre de composants relié à la vue contient les composants demandés et sont-ils dans l'état désiré ?

Le code ci-dessous montre le test de la page d'identification du portail GenGHIS-Web, il teste à la fois la navigation, un bean managé, et un composant graphique.

```
public void testLogin() {

// émulation d'un navigateur afin de tester HTTP
JSFClientSession client = new JSFClientSession("/login.faces");

//émulation de la liaison http avec le serveur
JSFServerSession server = new JSFServerSession(client);

//on donne une valeur au parametre userName
client.setParameter("userName", "testlogin");

//le client émule le clique sur le bouton d'identification
client.submit("login_button");

//vérification de la navigation vers la page suivante
assertEquals("/hello.jsp", server.getCurrentViewID());

//Vérification de la presence du composant graphique contenant le texte
//« welcome, testlogin »
```

```
assertTrue(client.getWebResponse().getText().contains("Welcome,
testlogin"));

//test du bean manage loginForm
assertEquals("testlogin
",server.getManagedBeanValue("#{loginForm.username}"));
                                                    }
```

Lorsque l'on utilise un tel outil, il est nécessaire de borner les tests à certains éléments précis, au risque sinon de se perdre dans des lignes de code inutiles testant l'intégralité des pages JSP.

Cette section conclut le chapitre cinq. Ce chapitre a présenté la phase de réalisation du projet dans laquelle nous avons développé l'application GenGHIS-Web.

Chapitre 6. Conclusion

6.1. Rappel des objectifs

Dans un but d'amélioration de GenGHIS et de sa mise en valeur sur internet, l'objectif de notre travail était triple :

- concevoir un portail collaboratif, GenGHIS-Web, permettant non seulement la diffusion et le partage d'applications SIST[37]-GenGHIS mais également leur construction et modification ;
- intégrer l'application GenGHIS existante au portail GenGHIS-Web, afin que GenGHIS prenne en charge les fonctionnalités de création/modification de SIST ;
- œuvrer sur la modularité de l'application GenGHIS et sa transformation en application client/serveur en vue de son intégration au portail GenGHIS-Web.

6.2. Synthèse

En vue de répondre aux objectifs énoncés ci-dessus, nous avons effectué une analyse et une conception complète du portail GenGHIS-Web. Le travail d'analyse nous a permis de mettre en évidence les différents types d'utilisateurs de cette application : simple internaute, utilisateur avec accès privilégiés à certains SIST, utilisateur concepteur de SIST. Dans cette optique, nous avons défini une gestion des droits d'accès aux SIST gérés par le portail GenGHIS-web qui permet de contrôler leur visibilité (publics, semi-publics, privés) et les

[37] SIST : Systèmes d'Information Spatiale et Temporelle

types d'opérations autorisées (lecture seule, lecture/copie, modification). Un ensemble complet de cas et scénarios d'utilisation a été défini couvrant l'ensemble des besoins exprimés pour les différents types d'utilisateurs et types de SIST.

Après une étape de maquettage qui a permis de valider nos différentes propositions (en particulier l'organisation de l'interface utilisateur du portail GenGHIS-Web) nous avons procédé au choix des différentes technologies pour la mise en œuvre de l'application GenGHIS-Web. Nous avons fait le choix d'une architecture Java Entreprise Edition (JEE) multicouches :

- la persistance des données est prise en charge par le SGBD relationnel PostgreSQL en conjonction avec l'implémentation Hibernate de l'API Java JPA (Java Persitence API) qui fait le pont entre le modèle relationnel de la base de données et le monde objet Java ;
- l'accès aux données s'effectue par une couche correspondant au patron de conception DAO (Data Acces Objects) qui permet de s'abstraire des choix techniques effectués pour la couche de persistance ;
- l'interface Web de l'application est réalisée à l'aide du cadriciel Java Server Faces qui propose une architecture Modèle Vue Contrôleur (MVC).

Pour la réalisation des fonctionnalités de création et modification de SIST proposées par le portail GenGHIS-Web, nous avons procédé à un travail de réingénierie de l'application GenGHIS existante. L'analyse détaillée de son fonctionnement, nous a permis de faire évoluer celle-ci en une application client/serveur modulaire (en particulier la génération des cartes, est exécutée sur le serveur, ce qui laisse au client des tâches moins coûteuses en ressources). L'interface utilisateur du module de données de GenGHIS a ainsi été déportée dans des applets communiquant via RMI (Remote Method Invocation) avec une partie s'exécutant sur le serveur GenGHIs-Web. Cette séparation nous a, entre autres, permis de dissocier complètement le code de l'interface utilisateur (applet) du code prenant en charge les modèles AROM utilisés pour l'intégration des données puis la génération du code des SIST. Nous avons également amélioré le processus de création/modification d'un SIST, en explicitant les associations entre le modèle des données brutes (MDB) fournies en entrée et le modèle conceptuel des données (MCD) AROM. Ces associations sont l'un des constituants d'un SIST GenGHIS, explicitées à l'aide d'un modèle objet. Elles sont mémorisées avec le SIST et peuvent être ainsi réutilisées et/ou modifiées par la suite.

L'ensemble des fonctionnalités de GenGHIS-Web ont été implémentées et testées (des tests automatisés ont été mise œuvre pour certaines).

Tout au long de notre travail, nous avons apporté un soin tout particulier à l'architecture logicielle, l'accent étant mis sur la modularité afin de faciliter la maintenance et les évolutions futures de l'application GenGHIS-Web.

6.3. Perspectives

Des évolutions sont possibles sur certaines couches de notre architecture :

La représentation de l'information : Pour la description des données utilisées par le générateur GenGHIS, l'application existante s'appuie sur le système de représentation de connaissances AROM. AROM est un système issu du monde de la recherche qui n'est plus développé ni maintenu. Le travail de "modularisation" que nous avons effectué sur GenGHIS en isolant le code AROM et en le séparant complètement du code de l'interface utilisateur est un premier pas vers le remplacement d'AROM par une autre technologie.

Pour représenter d'une manière formelle et structurée la connaissance terminologique d'un domaine d'application, les langages développés autours du Web Sémantique, en particulier le langage d'ontologies OWL (Ontology Web Language), enrichis de constructions spécifiques aux applications spatio-temporelles (à la manière des modèles conceptuels de MADS (24)), sont une alternative crédible à AROM et AROM-ST qui devra être étudiée.

Les SIST générés : Actuellement les SIST générés utilisent des technologies ne permettant que peu la manipulation des données et des cartes. D'une certaine manière, le SIST généré est statique : toutes les données sont intégrées aux fichiers HTML et SVG produits. Les scripts JavaScript permettent d'interagir avec leur représentation tabulaire, cartographique et/ou temporelle mais il n'est plus possible à ce niveau de modifier leur valeur et/ou la manière de les visualiser. Il faut régénérer un nouveau SIST. Même si le travail de réingénierie que nous avons effectué sur GenGHIS permet de modifier un SIST, certaines de ces modifications devraient pouvoir être effectuées directement sur le SIST généré. Les évolutions de technologies web (AJAX, HTML 5.0) permettent d'envisager le support d'une telle interactivité dans l'interface d'une application Web. C'est une piste d'étude qui doit elle aussi être développée.

Les sources de données : Pour le moment GenGHIS permet d'intégrer des données au format Excel ou MapInfo. L'extension vers d'autres formats de fichiers ne pose pas de difficultés. Nous pouvons envisager d'exploiter d'autres types de sources de données, en particulier des services Web, tels les WMS et WFS de l'OGC, à l'image de ce que fait CARMEN (voir chapitre 2).

Module de présentation : GenGHIS n'ayant pas de restriction dans la création des présentations, aucune vérification de la sémiologie n'est faite. Il serait nécessaire d'intégrer les règles de sémiologie graphique dans le processus de création de cartes. Le concepteur pourrait être guidé dans ses choix graphiques en fonction du domaine et du type des cartes construites.

Par ailleurs, concernant le module de présentation, on pourrait s'inspirer de WINDMash (voir chapitre 2) qui construit de manière totalement visuelle l'agencement des composants de l'interface graphique de l'application générée ainsi que les interactions utilisateur avec ces composants. Cela rendrait l'application de génération de SIST encore plus accessible à des utilisateurs non informaticiens.

Comme on peut le voir, ce projet offre donc de nombreuses perspectives, à la fois en termes de développement logiciel mais aussi en termes de recherche, sur les représentations d'informations spatiales et temporelles.

6.4. Bilan personnel

La rédaction de ce mémoire d'ingénieur CNAM est l'aboutissement d'une démarche commencée en 2005, lors de ma première inscription au CNAM de Grenoble. Mon entrée dans le monde du travail m'a permis de prendre conscience de certaines limites dans mes connaissances, ce qui m'a motivé à poursuivre mes études pour élever le niveau de mes compétences.

Durant ces dernières années, le CNAM m'a permis de remettre en question mon approche du monde professionnel et particulièrement de l'ingénierie.

Le stage de fin de cursus est une étape essentielle, elle permet de mettre en pratique les savoirs et les compétences acquises et offre une mise en situation correspondant au choix d'évolution professionnelle vers le métier d'ingénieur.

Le contact avec d'autres professionnels de la recherche m'a enrichi. Leur présence, leur disponibilité et leur savoir-faire m'a été d'un grand bénéfice. La collaboration étroite entre les membres de l'équipe est une des clés de la réussite du projet.
Le stage dont j'ai bénéficié m'a apporté des connaissances pratiques et théoriques : j'ai découvert le monde de la programmation, notamment le langage Java, le développement d'application Web, les technologies associées, les processus de conception de logiciels… Par les responsabilités qui m'ont été attribuées et la grande autonomie dont j'ai bénéficiée, j'ai pu prendre des décisions pour de nombreuses tâches et acquérir une rigueur scientifique.

D'un point de vue plus personnel, la rédaction de ce mémoire m'a permis d'améliorer mes capacités en expression écrite.

Pour conclure, l'évolution de mes compétences, me permet de mieux appréhender mon travail d'aujourd'hui et m'ouvre de nouvelles perspectives professionnelles.
Les années de partage et les enseignements dispensés par les acteurs du CNAM (professeurs et élèves, maîtres de stage), me permettent aujourd'hui d'aborder avec davantage de maturité mes futures missions.

Bibliographie

2. **Moisuc B, 2007.** *Conception et Mise en OEuvre de Systèmes d'Information.* Thése de l'Université Joseph Fourrier - Grenoble.

3. **Gayet L, 2009.** *Conception d'un générateur d'applications de Systèmes d'Information Spatio-Temporelle : GenGHIS.* Mémoire d'ingenieur - CNAM Grenoble.

5. **Alkante, 2009.** *Spécifications technique de Carmen.* Rapport.

6. **Barbe E, 2008** *Application de cartographie numérique internet du MEEDDAT.*

7. **Luong N, Etcheverry P, Nodenot T, 2010.** *WINDMash : A Visual Mashup Environment Dedicated to the Design of Web Interactive Applications***.** Article de l'Université de Pau.

9. **Miron A, 2009.** *Decouverte d'associations semantiques pour le Web Semantique Geospatial: le framework ONTOAST.* Thése de l'Université Joseph Fourrier - Grenoble.

10. **Consortium OpenGIS, 1999** *OpenGIS Simple Features Specification for SQL.*

13. **Blanc X,2006.** *UML 2 pour les développeurs.* Livre aux éditions Eyrolles.

14. **Norman D, 1986.** *User Central System Design.* Rapport de l'Université de Californie – San Diego.

17. **JBOSS, 2010.** *Richfaces documentation*. Documentation du cadriciel Richfaces.

18. **Doudoux J.-m, 2010.** Dévéloppons en java.

20. **Chennesal A.** *Java Server Face.*

23. **Marsden, R.** *John Snow and the Broad Street Cholera Outbreak of 1854*. Winwaed Software Technology LLC.

24. **Alliance, 2008.** *Struts vs JSF.* Lettre d'expertise.

Sites Internet

1. http://steamer.imag.fr/ Site web STEAMER. 06 08 2011

4. http://carmen.naturefrance.fr. Site d'accompagnement de l'application de cartographie en ligne. 15 07 2011

8. http://erozate.iutbayonne.univ-pau.fr/forbes2007/exp/. Journal de James David FORBES. 09 02 2011

11. http://fr.wikipedia.org Wikipedia.

12. http://agilemanifesto.org/iso/fr/ agilemanifesto

15. http://www.commentcamarche.net/contents/web/ergonomie.php3 Ergonomie Web. 11 06 2011

16. http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html. Java Server Faces. 02 02 2011

19. http://tomcat.apache.org/tomcat-6.0-doc/architecture/overview.html. Apache Tomcat Architecture. 27 07 2011

22. http://www.commentcamarche.net/contents/rmi/rmiarchi.php3. Architecture de RMI.

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

GenGHIS-Web : un portail collaboratif pour la création et le partage de Systèmes d'information Spatio-Temporelle

Benjamin BRICHET-BILLET **Grenoble, le 19 décembre 2011**

Résumé

L'avènement d'internet et du World Wide Web a offert au plus grand nombre la possibilité d'acquérir et de diffuser de l'information géographique. Dans ce contexte, à partir de l'application GenGHIS (Generator for Geographic and Historical Information Systems), nous avons procédé à la conception d'un portail collaboratif permettant la diffusion et le partage de Systèmes d'Information Spatio-Temporelle. En se basant sur l'état de l'art de différentes solutions et grâce à l'expression initiale des besoins, notre démarche de conception nous a conduits à la spécification des exigences mais aussi à la modélisation de la navigation. Cette application se base sur une architecture Java Entreprise Edition multicouche. Ainsi, elle fait appel à un certain nombre de technologies telles que Hibernate pour la persistance des données mais aussi le cadriciel Java Server Face pour l'architecture Modèle-Vue-Contrôleur.

Mots-clés : GenGHIS-Web, Systèmes d'Information Spatio-Temporelle, Système d'information géographique ; Hibernate, JSF.

Abstract

Since the creation of internet and of the World Wide Web, a large number of people have the ability to obtain and to diffuse geographic informations. In that context, from the present application "GenGHIS" (Generator for Geographic and Historical Information Systems), it has been necessary to create a web portal that enables the diffusion and the share of Spatio Temporal Information System applications. After a state of the art review and the initial expressions of need, our approach of conception led us to the requirements specifications and to the modeling of navigation. GenGHIS-Web Application is based on Java Enterprise Edition (JEE) architecture. It uses different technologies such as Hibernate for the data persistence and Java Server Face for the Model View Controller architecture.

Keywords : GenGHIS-Web, Spatio Temporal Information System, Geographic information system, Hibernate, JSF