UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

THIERRY KOSKAS

24 JUIN 2014

MASTER 2 G2M

RAPPORT DE STAGE

LE WEBMAPPING EN OPEN SOURCE DANS UNE PME D'EXPERTISE ECOLOGIQUE

REMERCIEMENTS :

Je tiens à remercier Monsieur Thomas Sauzon pour ses conseils. Malgré un emploi du temps chargé, il a su se rendre disponible pour m'orienter dans des domaines techniques nouveaux pour moi. Qu'il en soit remercié.

Je remercie également Monsieur Kovacs qui a accepté une organisation souple du projet tenant compte à la fois des contraintes de mon emploi du temps ainsi que celles de l'entreprise ; je travaillais en effet une journée par semaine en télétravail.

Enfin, ce projet n'aurait pas pu aboutir sans l'apport des cours dispensés à l'Université de Paris VIII notamment par Madame Isis Truck. Je tiens aussi à remercier vivement toute l'équipe pédagogique qui m'a encadré durant l'année.

SOMMAIRE

INTRODUCTION

Ingénieur en informatique de formation, j'ai souhaité étendre mes domaines de compétences aux SIG. C'est dans cette perspective que je me suis inscrit au Master G2M de l'université Paris VIII Vincennes-Saint-Denis. La formation dispensée comporte deux volets bien distincts : un enseignement théorique de 6 mois et un stage en entreprise de 4 à 6 mois visant à mettre en pratique les connaissances acquises.

Intéressé par le développement durable, et les aspects développement informatique autour du WebMapping, j'ai postulé auprès de plusieurs entreprises dont l'activité était en lien avec ces domaines. J'ai reçu plusieurs propositions et mon choix s'est porté sur la société Ecosphère.

Ce document est un **rapport d'étape de stage**. Il est en effet rédigé après quatre mois d'activité alors que le stage couvre une période de six mois. La société Ecosphère m'a confié la mission d'améliorer l'application de WebMapping qui utilise différentes couches logicielles. (OpenGeo suite).

Mon expérience récente en informatique me préparait peu à ce type de mission. Durant les dernières années, j'ai plutôt travaillé autour des notions du décisionnel et des bases de données. Cela est fort différent du travail demandé ici de développement orienté objet que je méconnaissais.

Après une présentation de la société et de ma mission de stage, nous présenterons chacun des aspects techniques qui concourent à la réalisation de ces applications. Puis nous décrirons mes réalisations en détail en rentrant dans des extraits de code informatique. En Annexe, nous décrirons des techniques ou technologies qui, bien qu'indispensables au projet, nous ont semblé plus secondaires (localhost, proxy…).

Extrait de l'offre d'emploi parue sur georezo :

Offre de stage

Développement webmapping

L'Entreprise
Ecosphère est un bureau d'études en environnement (faune/flore) spécialisé dans l'expertise écologique, le conseil et l'aménagement des milieux naturels basé à St Maur des Fossés (94, Ile de France).

http://ecosphere.fr

Cadre du Stage

La géomatique a pris une place centrale au sein des études faune/flore. D'abord exploités à des fins de cartographie, les SIG ont progressivement été mieux utilisés afin d'en faire des outils d'analyse et d'aide à la décision. Aujourd'hui, les technologies ont mûries et les besoins évolués : à travers le webmapping, la géomatique se veut plus ouverte, plus proches des utilisateurs finaux, aux non-géomaticiens.

C'est dans ce cadre qu'Ecosphère est en train de mettre en place une plateforme de webmapping, ouverte à ses clients et partenaires mais aussi à l'ensemble de ses salariés.

Le stage

L'objectif principal de ce stage est de développer, à partir d'une application existante, une application web permettant la consultation, l'édition, l'analyse (géotraitements) et la gestion de données SIG.

Le stage s'appuiera sur :

- PostgreSQL/PostGIS + Geoserver

- le langage JavaScript pour le développement de l'application : OpenLayers 3, GeoExt/Ext js
- un serveur ASP.NET en interne (Windows Server 2008 R2 avec SQL Server) en production
- un serveur Linux en interne (Ubuntu) en production
- un cahier des charges rédigé conjointement par le stagiaire et son maître de stage en début de stage

Selon les capacités du stagiaire, d'autres missions pourront lui être confiées, en accord avec les objectifs de son stage et les priorités d'Ecosphère.

Profil recherché

- Bac+3/+5 en Géomatique ou informatique
- Maîtrise du développement Web : JavaScript, CSS, HTML...
- Python et/ou ASP.NET apprécié
- Connaissance des technologies de webmapping
- Connaissance de l'ensemble PostgreSQL/PostGIS + Geoserver
- Compétences en .net et/ou python appréciées
- Autonome, curieux et rigoureux

Modalités

- Le stage pourra commencer dès possible pour une durée de 4 à 6 mois
- Rémunération en accord avec la législation et selon profil
- Localisé à Saint Maur des Fossés (94, Ile de France)

Contact

Thomas Sauzon
Responsable SIG et Informatique
Tél : 01 45 11 24 30
Mél : thomas.sauzon@ecosphere.fr
Candidature exclusivement par voie électronique (CV et texte l'accompagnant par mél)

ECOSPHERE : UNE PME DYNAMIQUE DANS LE DOMAINE DE L'EXPERTISE ECOLOGIQUE

Créée en 1988 par trois ingénieurs écologues, ECOSPHERE est devenue une **Société Anonyme** le 1er Janvier 1990.

C'est aujourd'hui un groupe dont le siège social est localisé à **Saint-Maur-des-Fossés** (94), à moins de 10 km du centre de Paris.

Il comprend une filiale, dénommée Ecothème, située à Cuvilly, près de Compiègne (60), créée en 1992 et acquise en 2000 et plusieurs agences ouvertes en avril 2005 à Vienne (38), transférée depuis sur la commune voisine de Sainte-Colombe (69), en octobre 2006 à Strasbourg (67), en juillet 2007 à Mérignac (33), en Juin 2010 à Aubagne (13) et en Janvier 2012 à Orléans (45).

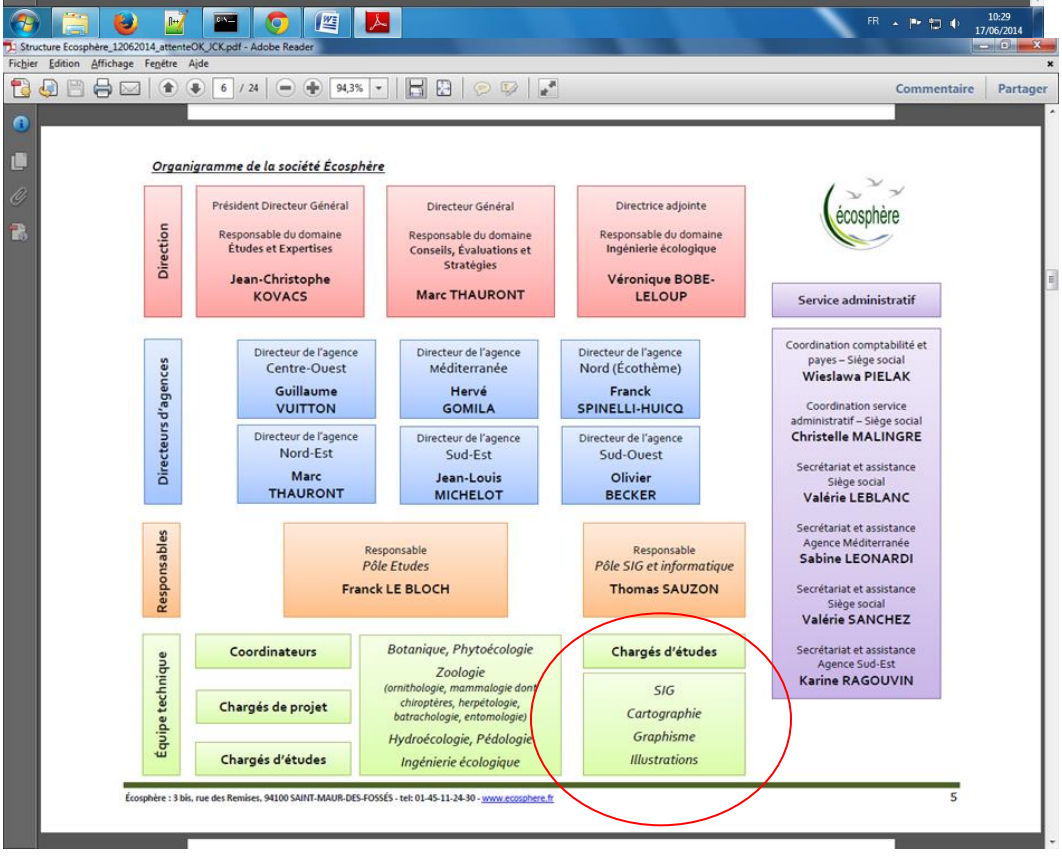
La société n'est liée à aucun groupe ou entreprise extérieure ce qui garantit aux clients une totale **indépendance d'esprit**. Le refus de recettes ou de solutions toutes faites permet de pratiquer une approche spécifique de chaque projet pour l'insérer au mieux dans son environnement naturel.

La société jouit d'une excellente réputation dans son domaine. Les affaires arrivent du fait de la **forte notoriété** ou en réponse à un appel d'offre.

Cela est sans doute dû à une **organisation souple en réseau**. A la différence de la structure hiérarchique elle laisse la place à l'initiative individuelle et à la une plus grande créativité. Cela nécessite toutefois une très grande **autonomie** de chaque acteur. Chaque chargé d'étude, spécialisé faune ou flore, est responsable de son périmètre et de ses études.

Présentons à présent la société au travers de son organigramme.

ORGANIGRAMME



Le siège de l'entreprise à St Maur (environ 20 personnes/ 70 salariés) ne détient pas tous les pouvoirs. Cela est sans doute dû à l'histoire de la société ainsi qu'à son activité intrinsèque près du terrain qui nécessite une vaste décentralisation.

Le siège intègre toutes les fonctions de l'entreprise. Les antennes régionales travaillent de manière quasi autonome.

La hiérarchie des équipes techniques bien que souple comporte des degrés d'expérience différents entre chargé d'étude, chargé de projet, et coordinateur.

En ce qui concerne la géomatique, le siège est composé du responsable ainsi que de deux sigistes. Dans chaque agence existe un correspondant SIG.

Les domaines d'activité reprennent tout ce qui est envisageable en études de l'écologie ; faune et flore.

DOMAINES D'ACTIVITE : UN PANEL DE SERVICES AUTOUR DE L'ECOLOGIE

Un projet typique d'Ecosphère pourrait être celui d'un grand chantier type rail ou autoroute. Cela nécessite de se préoccuper de la faune et la flore existant sur le tracé. Une phase d'inventaire sera suivie d'une étude d'impact. Avec éventuellement une étude sur des aménagements à prévoir pour renforcer la faune/ flore environnante ou déplacer certaines espèces. Les prestations d'Ecosphère vont jusqu'au suivi de chantier d'aménagement.

Les domaines d'activité sont de cinq ordres : (extraits d'un dossier interne de présentation de la société)

- **Études d'impact et expertises techniques de projets**
- **Expertises écologiques**
- **Ingénierie écologique**
- **Conseils, évaluation et stratégies**
- **Communication, sensibilisation et formation**

ÉTUDES D'IMPACT ET EXPERTISES TECHNIQUES DE PROJETS

Le groupe ÉCOSPHÈRE intervient dans des projets de natures très diverses : mines et carrières, centres de stockage des déchets, infrastructures diverses (routes et autoroutes, voies de chemin de fer, canaux, gazoducs, lignes à haute tension, éoliennes, aérodromes...), lotissements, Z.A.C., projets industriels, stations touristiques (golfs, parcs de loisirs...), stations d'épuration, champs captant pour la production d'eau potable, aménagements hydrauliques (aires de sur stockage, canaux et rivières...).

Le groupe ÉCOSPHÈRE est aussi régulièrement amené à réaliser des **expertises de sites, d'infrastructures et d'aménagements** à la demande des administrations, collectivités et clients privés :

- Évaluation du patrimoine naturel de sites préalable à des projets
- Suivis et bilans environnementaux d'équipements (infrastructures) ou d'activités
- Expertise de sites dégradés ou remis en état
- Bilan et expertise des politiques de conservation et d'aménagement

EXPERTISES ECOLOGIQUES

- Inventaires scientifiques de sites (ZNIEFF, Natura 2000, ENS…) ;
- Élaboration des trames vertes et bleue (TVB) dans le cadre notamment de la mise en œuvre des Schémas Régionaux de Cohérence Ecologique (**SRCE**) et de l'analyse des incidences des grands projets d'infrastructures linéaires (LGV, routes et autoroutes…) ;
- Définition des politiques et stratégies de conservation
- Réalisation des dossiers de conservation
- Expertises d'espèces végétales et animales en particulier sur les espèces
- Identification, caractérisation et cartographie des zones humides.

INGENIERIE ECOLOGIQUE : LE POLE AMENAGEMENT

Dès sa création en 1990, ECOSPHERE a cherché à être **opérationnel et innovant** en matière d'**Aménagement et de Gestion Ecologique d'Espaces Naturels et/ou Modifiés** en développant une spécialité en **ingénierie écologique,** avec un service et une équipe dédiée.

Les **prestations** ou **missions** menées se rattachent à **4 grandes catégories** :

1. Conception de projets ;
2. Encadrement des travaux de génie écologique ;
3. Recherche appliquée ;
4. Suivis écologiques et audits techniques**.**

Dans le détail, les prestations se déclinent en :

- Réalisation d'études **d'aménagement** et de gestion d'espaces à vocation naturelle
- Conception et mise en œuvre de mesures de réduction et de compensation écologique
- Assistance à Maîtrise d'Ouvrage dans le cadre de projets d'aménagement et de gestion d'espaces naturels
- Maîtrise d'œuvre
- Actions en faveur des trames vertes et bleues (passages et aménagements pour la faune, réseaux de mares…) ;

CONSEILS, EVALUATION ET STRATEGIES

Le groupe ÉCOSPHÈRE intervient dans **la définition, la mise en place, le suivi et l'évaluation de politiques ou de projets de gestion des ressources naturelles** ayant trait à différents domaines :

- **Les audits et l'évaluation de politiques sectorielles et de projets spécifiques** dans le domaine la conservation du patrimoine naturel et du développement rural
- **L'aménagement du territoire** : Chartes de l'environnement, projets de territoires, chartes de PNR, SCOT, POS et PLU, SDAGE, Schémas de carrière…
- **Le développement durable** : Agendas 21, Analyse Environnementale sur l'Urbanisme (AEU)…
- **La gestion des espaces ruraux, agricoles et forestiers** (mesures agri-environnementales, sylvo-environnementales...) ;
- **La gestion de la ressource en eau**

COMMUNICATION, SENSIBILISATION ET FORMATION

Le groupe ÉCOSPHÈRE attache une importance particulière à la stratégie de communication des dossiers auxquels il participe. Pour cela il est nécessaire de présenter les **enjeux dans un langage clair et compréhensible par tous** :

- **Définition des problématiques liées à la conservation des milieux et espèces sauvages** dans différents cadres (aide à la décision dans le cadre d'arbitrages entre divers intérêts : aménagement, conservation...) et proposition de stratégies

- **Évaluation des incidences juridiques** (espèces protégées, ZNIEFF, ZICO, Zones Natura 2000, aires protégées, conventions internationales...) ;
- **Animation de réunions et gestion de la communication** avec divers partenaires (industriels, collectivités locales, administrations, associations...).

Dans ce contexte, la conception et l'élaboration de **documents pédagogiques** (plaquettes, panneaux d'information...) ou de **guides et brochures techniques** peuvent constituer des outils privilégiés.

Compte tenu de son expérience, ÉCOSPHÈRE est aussi régulièrement sollicitée pour l'organisation et l'animation de **sessions de formation** dans les domaines suivants :

- Politiques Natura 2000
- Nature en ville (écosystèmes, faune, flore, gestion différenciée des espaces verts)
- Agendas 21 et autres documents de planification locaux
- Infrastructures linéaires (routes, voies ferrées…), Mines et carrières…

UN ENVIRONNEMENT DE TRAVAIL AGREABLE

Le siège d'Ecosphère est situé à Saint-Maur-des-Fossés tout près du RER, ce qui est fort pratique pour les salariés

Il occupe un petit immeuble de deux étages, anciennement d'habitations. Les locaux accueillent une vingtaine de personnes. Chaque bureau est occupé par deux ou trois personnes. J'étais seul dans un bureau de deux. Cette configuration est nettement plus agréabe que celle de travailler dans un Open space. L'ambiance est très bonne, à la fois studieuse et décontractée.

J'étais sous la responsabilité de Thomas Sauzon, le responsable informatique / Géomatique. C'est la seule personne susceptible de m'encadrer ; les autres géomaticiens connaissent peu le WebMapping. Ces derniers développent leur activité principalement avec l'outil ArcGis. Présentons ici leur activité.

Le géomaticien d'Ecosphère : au cœur de l'activité

Toutes les études diffusées par Ecosphère comportent une partie cartographie. De plus toutes les données de terrain mainiées par la société sont à composante géographique par essence.

Les trois géomaticiens du siège partagent leur temps entre deux activités principales : l'amélioration des cartes et participer à des projets SIG. Nous présenterons ici ces deux volets. Les géomaticiens en agence partagent leur temps de manière identique.

UNE FONCTION SUPPORT : LA CARTOGRAPHIE

Par essence, les métiers de l'écologie sont propices à l'utilisation de carte. Pour définir par exemple des périmètres de présence d'espèces.

Nous donnons ici un exemple de carte concernant les chauves-souris depuis la saisie papier sur le terrain à la carte finale rendue dans l'étude.

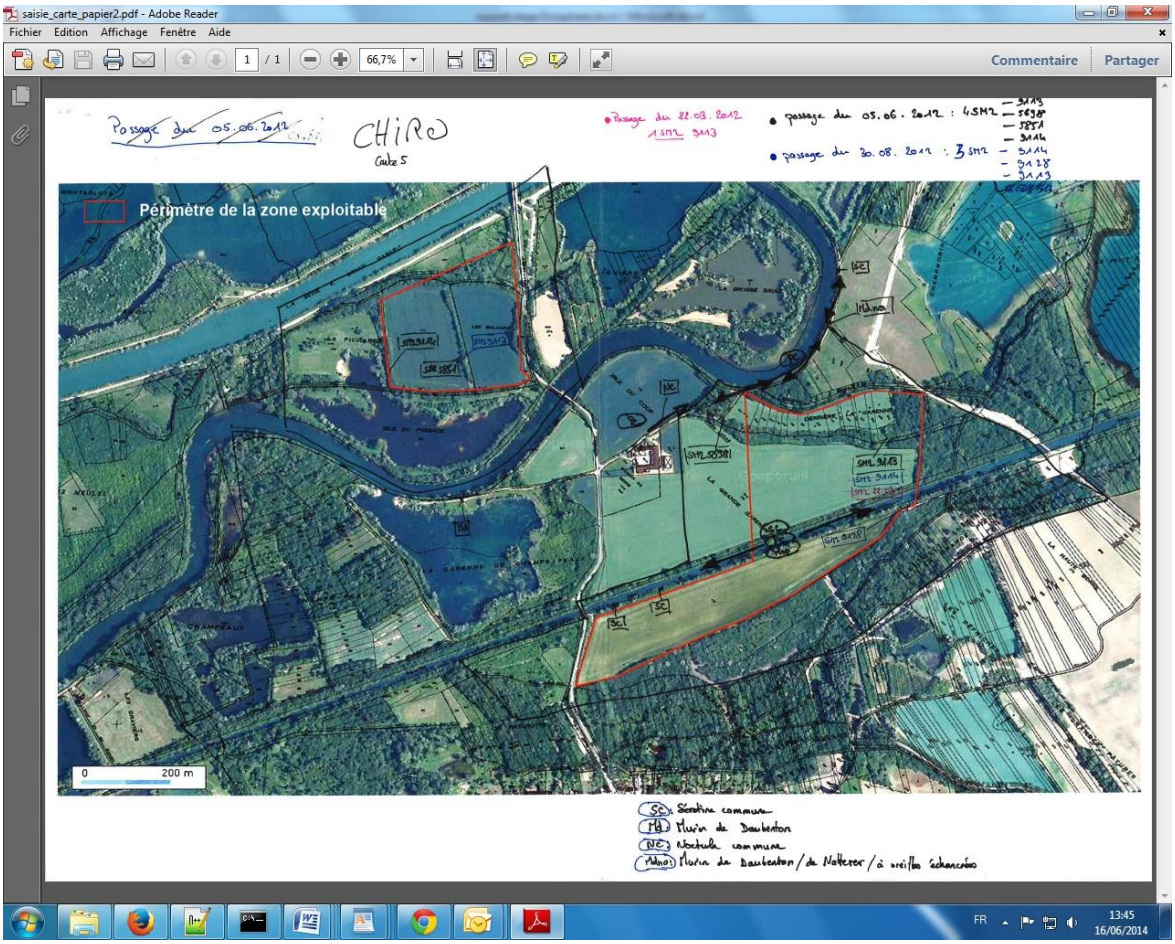
Les chargés d'étude vont sur le terrain. A partir d'une ortho-photo papier, Ils saisissent différentes informations :

- Des points de relevé de présence d'espèce. Cela se fait à partir d'empreintes ou en observant l'espèce à l'œil nu ou parfois à la jumelle.
- Des polygones de mares ou foret ou autres
- Des points qui signalent le placement d'un dispositif de capture d'information. Ces dispositifs servent à repérer différents types d'espèces présentes sur le terrain.

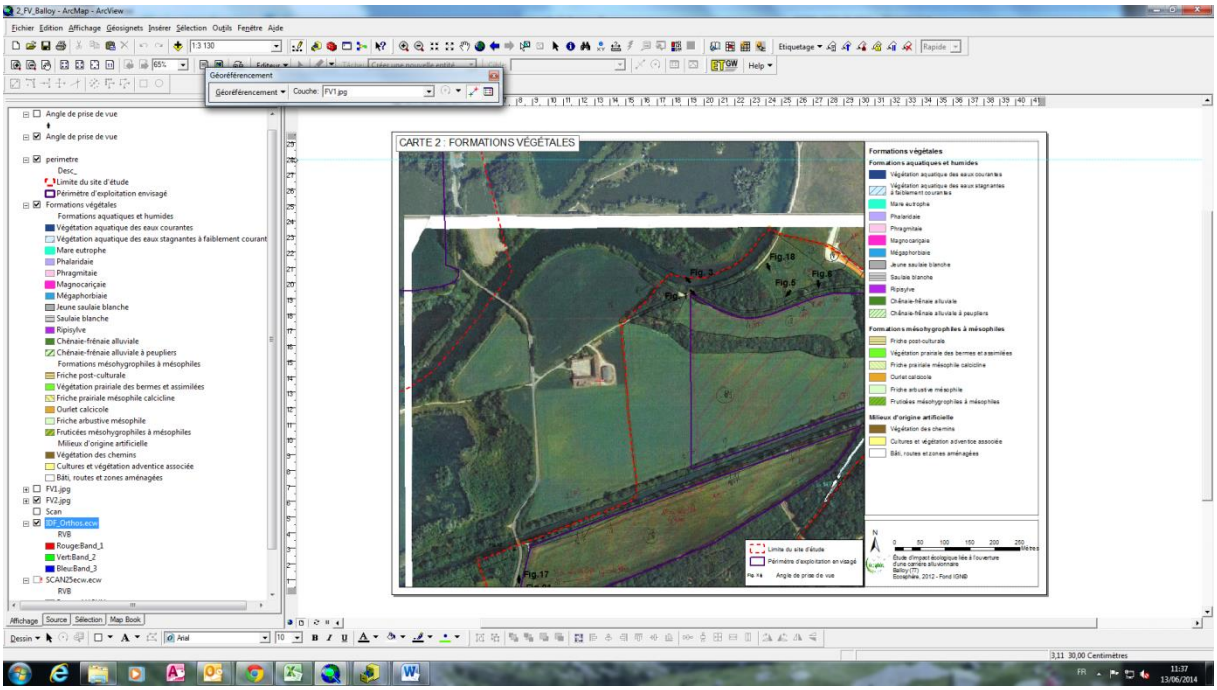
Ces données peuvent être géo localisées au GPS ou non. Dans ce cas un repérage dans l'espace en rapprochement de la carte papier est nécessaire. Ce qui peut engendrer des erreurs de quelques mètres.

Le fond de carte enrichi de ces informations est remis au cartographe. Différentes phases vont suivre :

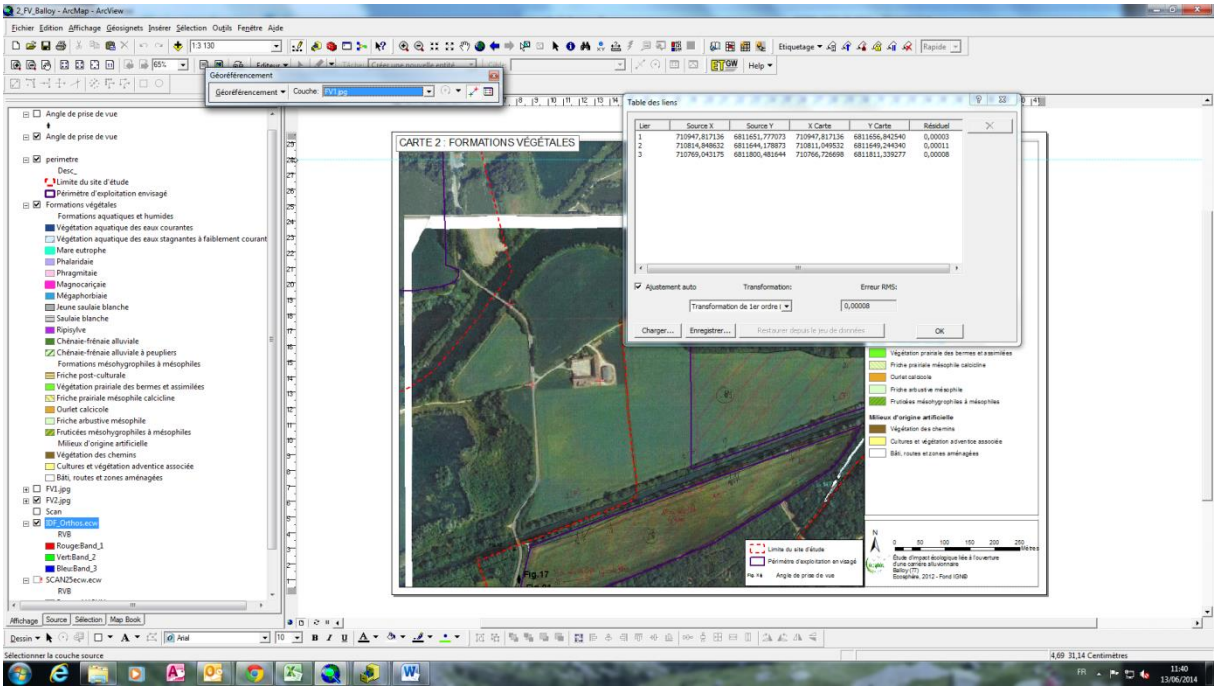
1. La carte papier de la zone étudiée est scannée avec les informations manuelles des chargés d'étude



2. Une fois scannée, la carte papier est supperposée à un fond ortho-photo géoréférencé.

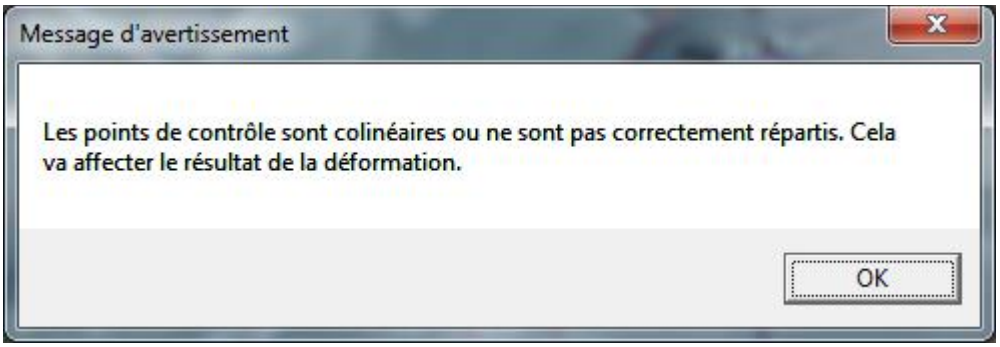


3. Puis on procède au géoréferencement.

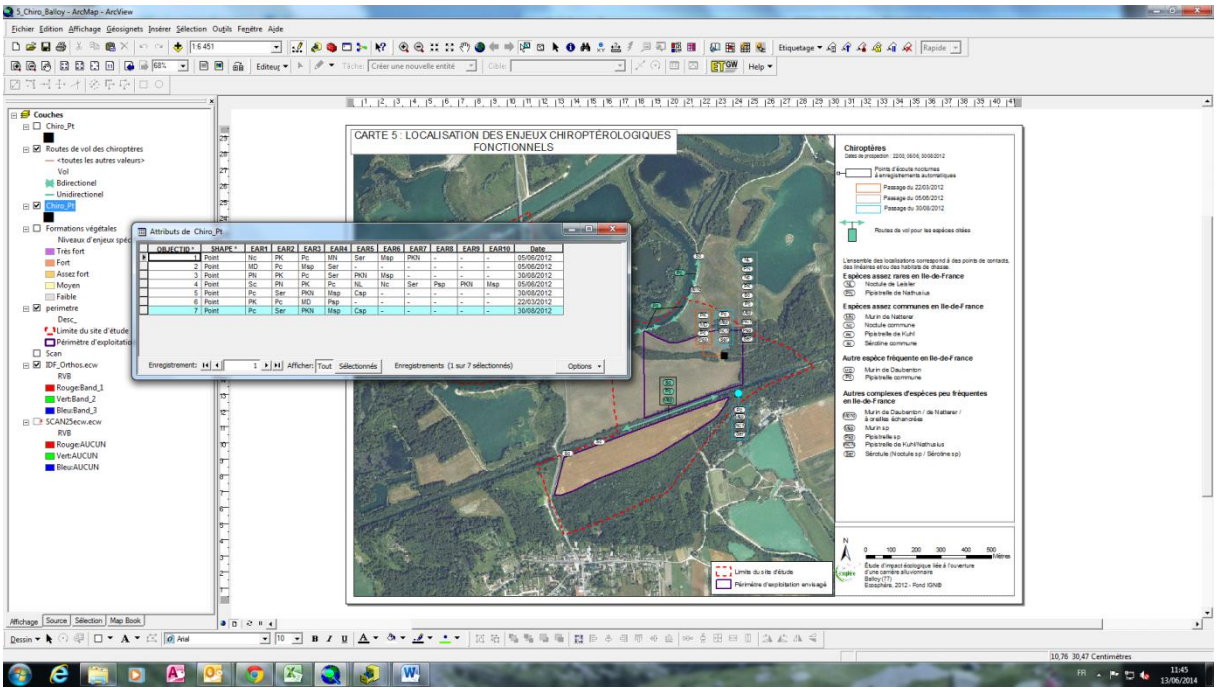


En théorie 3 points sont suffisants. Dans la pratique 5, ou 6 points sont nécessaires.

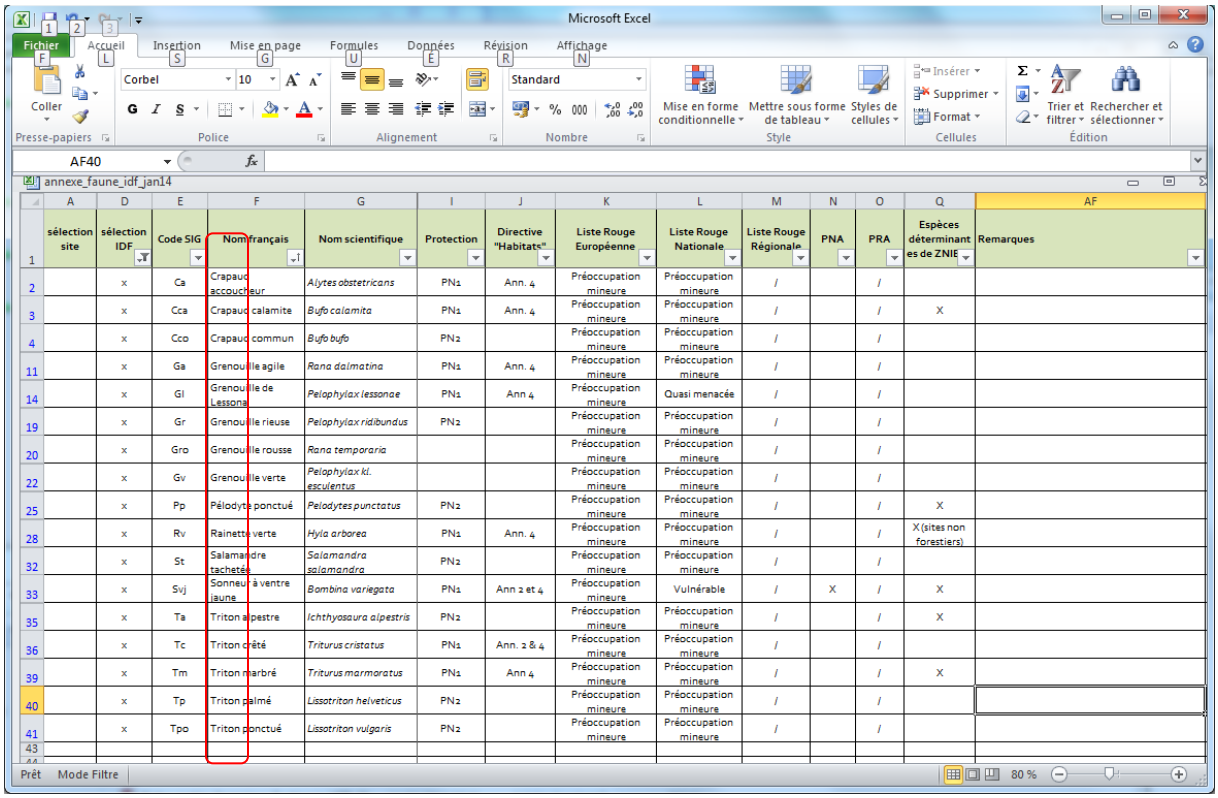
Si ces points sont trop proches les uns des autres nous obtenons le message suivant :



1. Récupération des données



Chaque point saisi est une ligne dans la table attributaire. Il comportera d'autant de colonnes que d'espèces présentes sur ce point. Une espèce est caractérisée par son identifiant sur 3 caractères. Ces 3 caractères se retrouvent sur un fichier Excel socle qui récence toutes les espèces (code SIG):



	A	D	E	F	G	I	J	K	L	M	N	O	Q	AF
1	sélection site	sélection IDF	Code SIG	Nom français	Nom scientifique	Protection	Directive "Habitats"	Liste Rouge Européenne	Liste Rouge Nationale	Liste Rouge Régionale	PNA	PRA	Espèces déterminant es de ZNIE	Remarques
2		x	Ca	Crapaud accoucheur	*Alytes obstetricans*	PN1	Ann. 4	Préoccupation mineure	Préoccupation mineure	/		/		
3		x	Cca	Crapaud calamite	*Bufo calamita*	PN1	Ann. 4	Préoccupation mineure	Préoccupation mineure	/		/	X	
4		x	Cco	Crapaud commun	*Bufo bufo*	PN2		Préoccupation mineure	Préoccupation mineure	/		/		
11		x	Ga	Grenouille agile	*Rana dalmatina*	PN1	Ann. 4	Préoccupation mineure	Préoccupation mineure	/		/		
14		x	Gl	Grenouille de Lessona	*Pelophylax lessonae*	PN1	Ann 4	Préoccupation mineure	Quasi menacée	/		/		
19		x	Gr	Grenouille rieuse	*Pelophylax ridibundus*	PN2		Préoccupation mineure	Préoccupation mineure	/		/		
20		x	Gro	Grenouille rousse	*Rana temporaria*			Préoccupation mineure	Préoccupation mineure	/		/		
22		x	Gv	Grenouille verte	*Pelophylax kl. esculentus*			Préoccupation mineure	Préoccupation mineure	/		/		
25		x	Pp	Pélodyte ponctué	*Pelodytes punctatus*	PN2		Préoccupation mineure	Préoccupation mineure	/		/	X	
28		x	Rv	Rainette verte	*Hyla arborea*	PN1	Ann. 4	Préoccupation mineure	Préoccupation mineure	/		/	X (sites non forestiers)	
32		x	St	Salamandre tachetée	*Salamandra salamandra*	PN2		Préoccupation mineure	Préoccupation mineure	/		/		
33		x	Svj	Sonneur à ventre jaune	*Bombina variegata*	PN1	Ann 2 et 4	Préoccupation mineure	Vulnérable	/	X	/	X	
35		x	Ta	Triton alpestre	*Ichthyosaura alpestris*	PN2		Préoccupation mineure	Préoccupation mineure	/		/	X	
36		x	Tc	Triton crêté	*Triturus cristatus*	PN1	Ann. 2 & 4	Préoccupation mineure	Préoccupation mineure	/		/		
39		x	Tm	Triton marbré	*Triturus marmoratus*	PN1	Ann 4	Préoccupation mineure	Préoccupation mineure	/		/	X	
40		x	Tp	Triton palmé	*Lissotriton helveticus*	PN2		Préoccupation mineure	Préoccupation mineure	/		/		
41		x	Tpo	Triton ponctué	*Lissotriton vulgaris*	PN2		Préoccupation mineure	Préoccupation mineure	/		/		
43														

Remarque 1: Dans l'exemple décrit, les chargés d'étude saisissent sur une carte papier leur donnée. De plus en plus, ces données sont saisies sur terminal informatique de type Gtack ou tablette.

Remarque 2 : Pour alléger les tâches de création de cartes des géomaticiens, a été conçue l'application Géosphère, qui est une application souple et simple de diffusion, et de saisie des cartes. Cette application permet aux chargés d'étude eux-mêmes de saisir directemet les cartes. C'est sur cette application de type WebMapping, à améliorer, que se porterons nos efforts.

Une fonction plus intégrée : Les SIG

Les géomaticiens d'Ecosphère réalisent aussi leurs propres études à plus forte valeur ajoutée que la cartographie.

Exemple du **SRCE** de la région IDF :

Ce Schéma Régional de Cohérence Ecologique est une Obligation depuis le Grenelle de l'environnement pour toutes les régions. Carte des trames vertes (foret…) Et bleues (milieu d'eau, mares…) TVB

Une utilisation massive des géotraitements d'ArcGis est nécessaire pour arriver à la carte finale qui comprendra des aires de migration simulées ainsi que des corridors ou axes de déplacement principaux.

On part d'une **carte d'occupation des sols**. Il n'y a pas de travail de terrain pour établir cette carte.

On repère les points de départ des espèces, soit les réservoirs de biodiversité. En utilisant les outils ArcGis « extension spatiale Analyst » « Distance » « distance de coût) On obtient une **carte des couts de friction**

Remarque : cette carte est de type raster. Chaque pixel contient une information : code habitat (foret, culture, habitat,…)

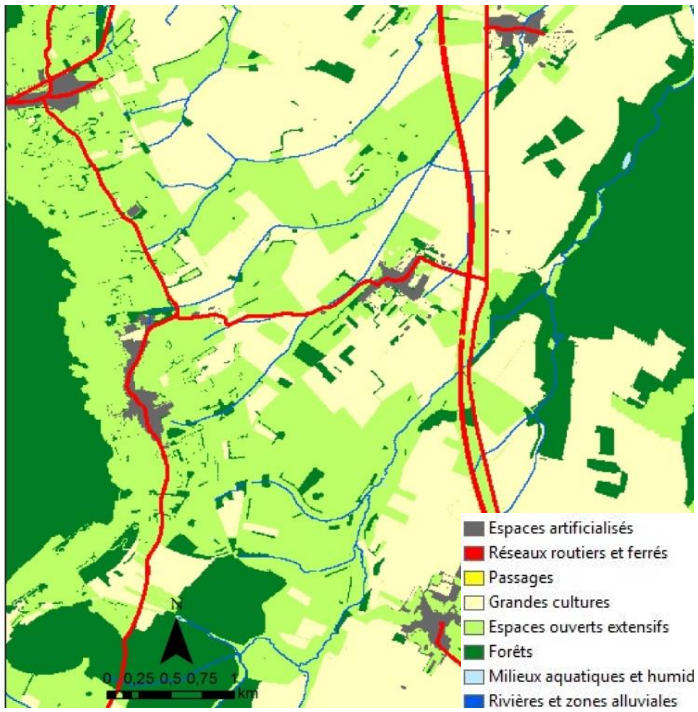
Puis on obtient des **aires de migration simulée**.

Puis on obtient les **corridors** ou *voies de déplacement effectives*. A partir de l'outil « chemin de coût »

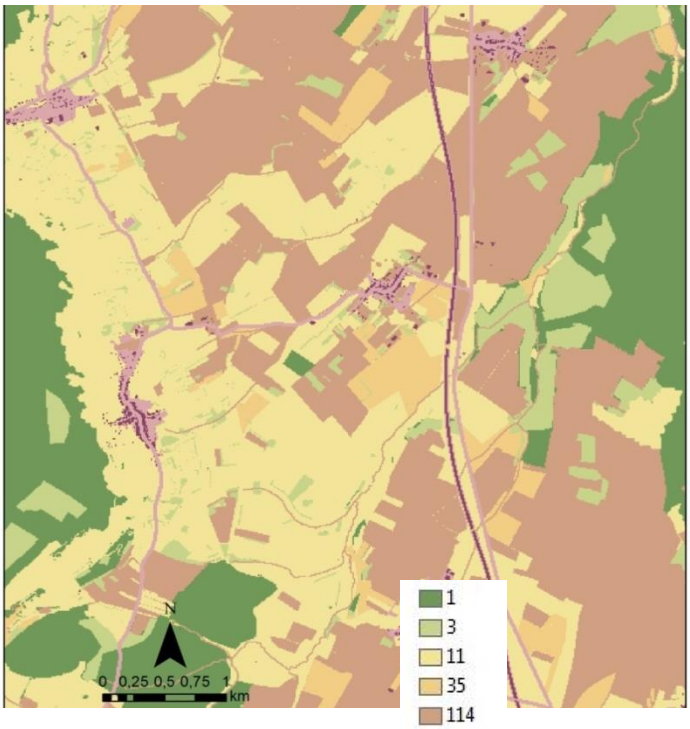
Puis en croisant les routes et les corridors on obtient les **zones de conflit.**

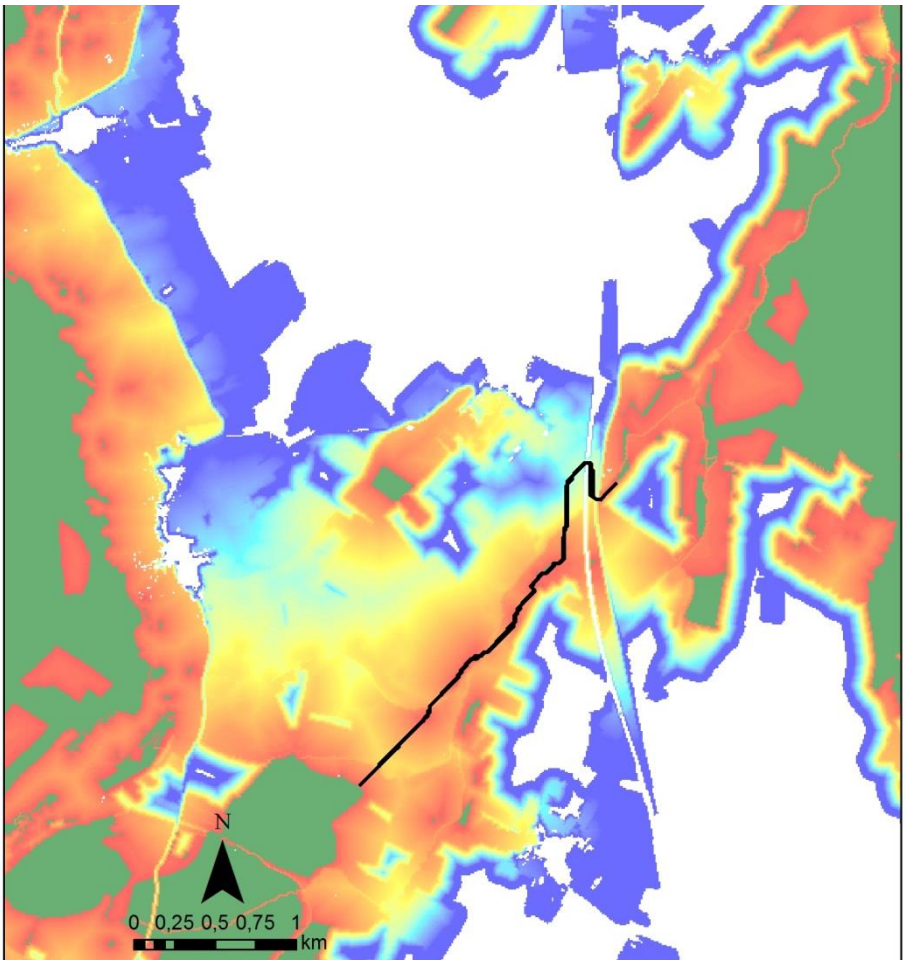
Cette analyse purement théorique demande une validation des naturalistes experts

Occupation des sols :



Coûts de friction : énergie dépensée pour se déplacer





Aires de migrations simulées

AMS avec réservoir Biodiversité en vert (dispersion des espèces dans Migration simulées) + Corridors en noir

Plus c'est rouge plus il est aisé de se déplacer

L'EXISTANT ET SES AMELIORATIONS A APPORTER

J'ai commencé mon travail par un receuil des besoins utilisateur. J'ai interviewé Sébastien, chargé d'études, qui utilise l'application depuis un certain temps pour avoir un retour sur l'application WebMapping existante. Cette étape est indispensable si nous voulons coller au mieux les développements aux besoins réels des utilisateurs. Par expérience, beaucoup de projets dérapent par négligence de cette étape.

La fiche d'entretien correspondante est en annexe.

Parmi tous les points à améliorer, l'usage du « snaping » a été jugée prioritaire. C'est sur ce point que se sont portés mes efforts.

Ce travail de développement repose sur un socle de logiciels. C'est l'architecture de ces locigiels que nous allons décrire maintenant.

L'ARCHITECTURE TECHNIQUE ET LOGICIELLE

Avant de présenter les différents logiciels utilisés, nous présentons ici un résumé de ce qu'est le WebMapping.

NOTIONS DE WEBMAPPING

Le WebMapping (ou WebSIG) est la mise en ligne du système d'information géographique et plus largement de cartes permettant de diffuser celui-ci et celles-ci à travers le web (réseau internet). En d'autres termes, le WebMapping ou la cartographie en ligne est l'usage des technologies de l'Internet pour le **stockage** et la **diffusion** de l'information géographique.

Une application de WebMapping devrait pouvoir offrir les fonctionnalités ci-après :

- cartographier des données géographiques à la demande selon le choix des couches et de l'emprise géographique;

- afficher des cartes dans un navigateur ;

- effectuer des mesures sur des cartes ;

- accéder à des bases de données métiers et sémantiques ;

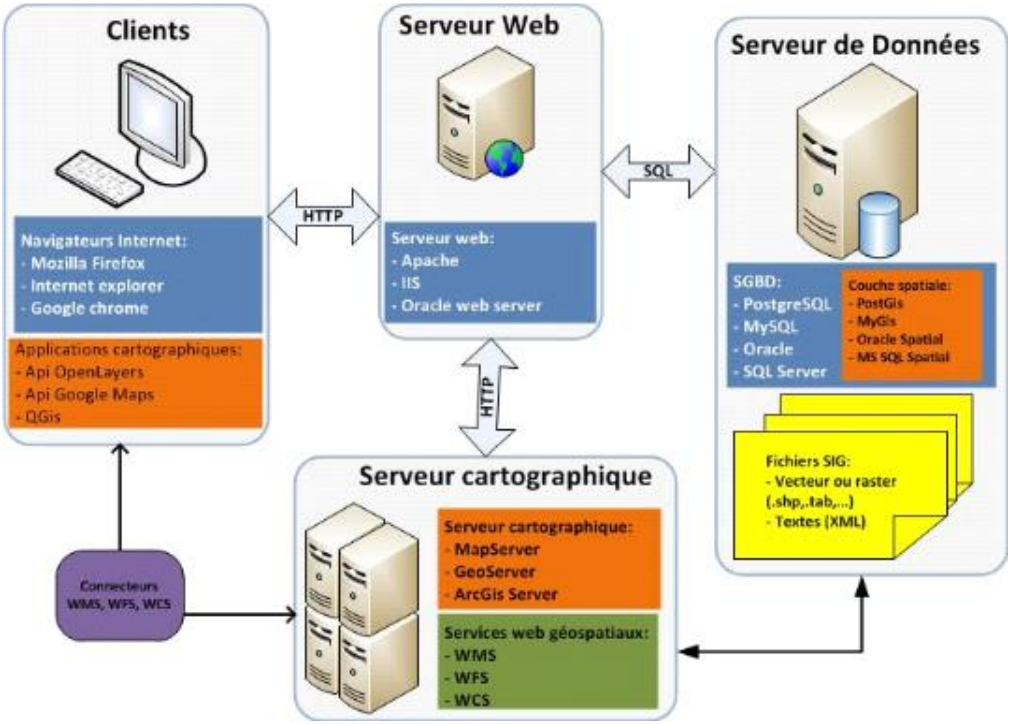
- faire des recherches portant sur la sémantique ou la géométrie des données cartographiées ;

- saisir de l'information pour alimenter la base de données sur le serveur ;

- effectuer des traitements complexes comme le calcul des itinéraires ;

- imprimer des cartes en ligne ;

Aujourd'hui avec l'avènement du Web 2.0, le WebMapping n'est plus au stade du simple affichage et de visualisation de cartes dans un navigateur. Mais, il se caractérise de nos jours par une forte interactivité et des contenus géolocalisés générés par les utilisateurs. Ainsi, chaque utilisateur peut créer des cartes personnalisées, les partager à d'autres utilisateurs. Et on parle de WebMapping 2.0 orienté grand public.



Le client, envoi une requête au serveur web (Apache dans notre cas) par le protocole http. Il utilise pour cela un navigateur quelconque. Cette requête est encapsulée dans l'API **OpenLayers**. Le serveur Web envoie une requête SQL au serveur de données (**PostGis** dans notre cas). PostGIS renvoie des fichiers SIG ou textes (XML) au serveur cartographique (**GeoServer** dans notre cas). Enfin GeoServer renvoie une information **WMS** ou **WFS** au client. La carte s'affiche alors sur le navigateur du poste client.

Tous les mots clé en gras sont explicités ci-dessous.

RAPPELS SUR JAVASCRIPT

(Souvent abrégé JS) est un langage de programmation de scripts principalement utilisé dans les pages web interactives mais aussi côté serveur. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés. En outre, les fonctions sont des objets de première classe.

Les possibilités de programmation objet y sont très limitées. Celles-ci seront présentés dans des API que nous étudierons plus loin tels Ext JS ou OpenLayers.

Dans le source HTML ou dans un fichier externe

```
<!DOCTYPE html>
<html>
<head>
<title>Example HTML Page</title>
</head>
<body>
<!-- content here -->
<script type="text/javascript" src="example1.js"></script>
<script type="text/javascript" src="example2.js"></script>
</body>
</html>
```

Ce langage peut être enrichi par des API qui le structurent plus. Nous utiliserons quant à nous OpenLayers, Ext Js, Gxp, GeoExt. Ces Api sont décrites plus bas.

On sera gêné tout au long du développement du fait que les variables ne sont pas typées. De ce fait **l'autocomplétion** n'existe pas. Cela aurait évité de devoir se référer à la documentation en permanence pour connaitre, par exemple, les méthodes s'appliquant à un objet.

Après ce rappel et ces quelques remarques, nous analyserons le cœur des logiciels utilisés, notamment OpenGeo Suite.

OPENGEO SUITE : UNE SUITE DE WEBMAPPING INTEGREE

Cette suite open sources logicielles permet de publier des données geospatiales sur le web. Elle est composée de différentes couches :

- **Postgis** est une base de donnees robuste qui gere les données geographiques. Ce socle est transparent pour notre usage. Nous nous contenterons d'y importer des shape file en local.
- **Geoserver** est un serveur de cartes et de donnees geospatiales et de cartes. Nous detaillerons certaines notions connexes plus bas.
- **Geoexplorer** est un outil visible sur un navigateur qui permet de visualiser, naviguer et gérer les données de la suite en local ou à distance

- **Geowebcache** est un outil qui accelere l'affichage des cartes en pre-affichant et en gerant un cache des images de carte. Cela fonctionne de maniere transparente et nous n'intervenons pas dessus.
- **Geoext** : openlayers + ext js. Partie programmation qui nous interesse plus. Nous y reviendrons.

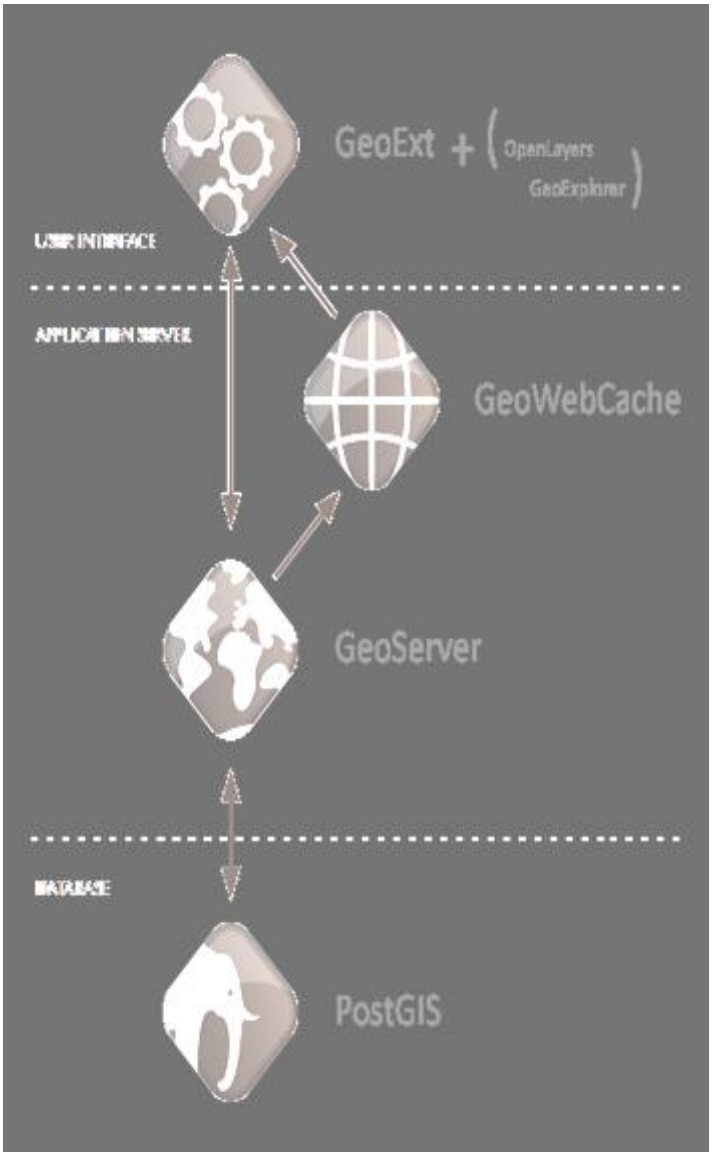
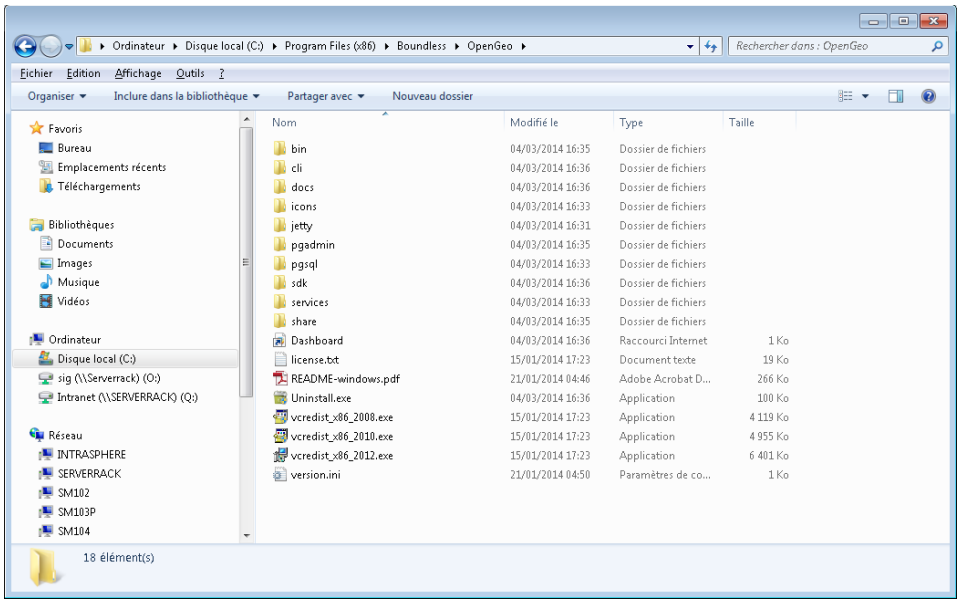


Schéma représentant les différentes couches de l'OpenGeo Suite :

Après une présentation des notions de serveur web, de serveur web cartographique, de WMS et de WFS de l'OGC, nous zoomerons sur notre activité qui se situe dans le haut du

schéma : la programmation de la suite avec différentes extensions de JavaScript : OpenLayers, GeoExt, GXP, Ext Js. Cette partie de codage occupera la quasi-totalité de stage.

REPERTOIRES APRES INSTALLATION



On reconnait les répertoires pgadmin, pgsql, Sdk.…

Le répertoire Sdk comporte le fichier suite-sdk.cmd qui faudra invoquer pour débuguer les programmes.

POSTGIS : LA BD OPENSOURCE QUI GERE L'INFORMATION GEOGRAPHIQUE

Nous ne nous étendrons pas sur ce logiciel étudié en cours, à l'université de Paris VIII-Vicennes-Saint-Denis.

Nous utiliserons, en cours de projet, son module d'importation de Shape file pour y intégrer de nouvelles couches sous GeoServer. Etant donné que nous sommes dans la suite OpenGeo, tout communique. Les Shape file importés dans PostGis sont accessibles par GeoServer.

GEOSERVER

Ce logiciel sert à partager et à éditer des données geospatiales suivant le Protocole OGC

C'est un serveur de données geospatiales à travers le web.

Dans cette partie, nous introduisons la notion de web server et son fonctionnement. J'ignorai tout de ces notions et j'ai souhaité les introduire pour une meilleure compréhension de l'architecture.

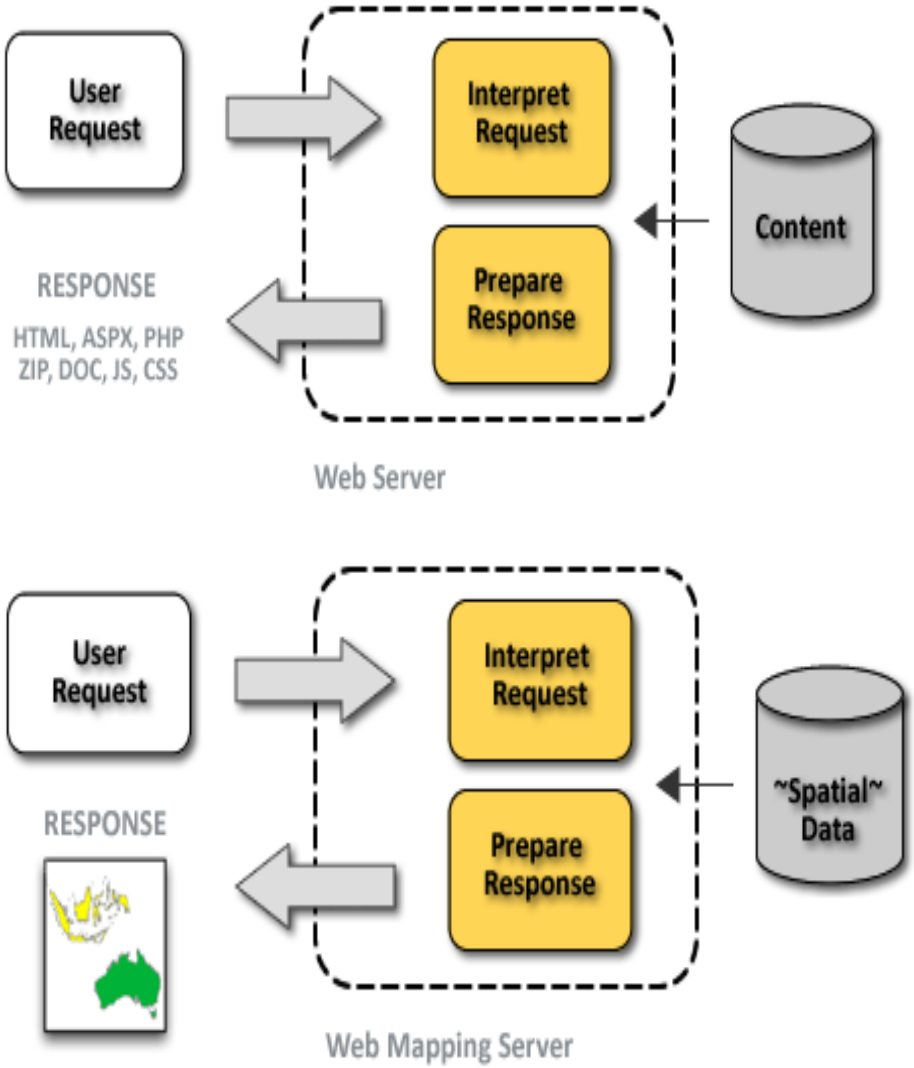
GeoServer fonctionne à partir de l'utilisation massive de **WebService** en tant que serveur web cartographique.

WEB SERVERS ET WEB MAPING SERVERS

Voyons ce qui distingue les deux.

Un serveur web est un programme qui renvoie des contenus (pages web, images, fichiers, données, etc.) en utilisant http (HyperText Transfer Protocol. Lorsqu'on utilise un navigateur pour se connecter à un site web, on utilise un serveur web.

Le serveur web prend la requête, l'interprète, et retourne une réponse à l'utilisateur. Celle-ci est renvoyée à l'écran via le navigateur.



Dans le cas d'un serveur web « Mapping » la réponse n'est pas de type document ou fichier mais des données géographiques.

QU'EST CE QU'UN WEBSERVICE ?

Un Service Web est un programme informatique permettant la communication et l'échange de données entre applications et **systèmes hétérogènes** dans des **environnements distribués**. Il s'agit donc d'un ensemble de fonctionnalités exposées sur Internet ou sur un Intranet, par et pour des applications ou machines, sans intervention humaine, et en temps réel.».

En géomatique un WebService propose un Service qui va permettre la prise en charge distante de données, soit pour l'affichage simple de carte (**WMS**), soit pour du stockage de données (WCS et **WFS**) soit pour du traitement distant pour éviter d'utiliser du temps d'utilisation du processeur et de la mémoire.

Du côté des services spatiaux, notre client envoi une requête pour connaitre les possibilités du serveur (GetCapabilities), il peut demander une description supplémentaire pour une couche particulière puis demande les données, le traitement, (GetMap ou GetFeatureInfo dans le cas d'un WMS). Après chaque requête le client reçoit une réponse sous forme de fichier XML ou image dans le cas des WebService de l'OGC.

En géomatique, un WebService est un service qui propose des cartes (WMS, WTS), des données brutes (WFS, WCS, GML, GeoRSS), des données sur les données ou métadonnées (CAT), des informations sur la sémiologie (SLD), sur les données d'une carte (WMC), etc.

INTEROPERABILITE

L'interopérabilité est la capacité que possède un produit ou un système dont les interfaces sont intégralement connues à fonctionner avec d'autres produits ou systèmes existants ou futurs.

Un exemple de systèmes interopérables est le téléphone. Toutes les interfaces sont des normes gérées par l'UIT-T. On peut ainsi téléphoner sans se soucier de la marque de téléphone de son correspondant ni des matériels utilisés par les différents opérateurs.»4)

L'interopérabilité permet ainsi une meilleure communication entre les différentes briques logiciels : client, serveur, à source ouverte ou propriétaire. Elle peut être considérée comme une langue universelle.

Revenons sur les définitions de WMS et WFS qui caractériseront les couches de carte que nous allons utiliser.

WMS (WEB MAP SERVICE)

Le WMS est une spécification internationale de diffusion et d'utilisation de cartes dynamiques sur le Web. WMS permet de produire des cartes de données georéférencées à partir de différents serveurs de données. Il définit un ensemble de serveurs cartographiques mis en

réseau pour construire des cartes interactives. Les données sont délivrées sous différents formats : tiff, gif, jpeg, bmp, png, svg.

Un serveur WMS satisfait aux requêtes GetCapabilities, GetMap et GetFeature. Une requête GetCapabilities retourne les métadonnées qui décrivent le contenu du service et les paramètres acceptés c'est-à-dire les capacités du serveur. La requête GetMap retourne une image de la carte demandée dont les paramètres geospatiaux et dimensionnels ont été définis. Le GetFeature retourne des informations sur un objet représenté sur la carte.

WFS (WEB FEATURE SERVICE)

Le WFS permet, au moyen d'une URL formatée, d'interroger des serveurs cartographiques afin de manipuler des objets géographiques (lignes, points, polygones,...), contrairement au WMS qui permet la production de cartes georéférencées à partir de serveurs géographiques. WFS propose des interfaces pour la description des manipulations de données sur des objets géographiques. Les opérations de manipulation de données permettent de :

- créer des nouveaux objets ;
- effacer des objets ;
- mettre à jour des objets ;
- prendre ou rechercher des objets sur la base de contraintes spatiales.

Un serveur WFS répond aux requêtes GetCapabilities, DescribeFeatureType, GetFeature, LockFeature, Transaction. La requête DescribeFeatureType permet de retourner la structure de chaque entité susceptible d'être fournie par le serveur. La requête Transaction permet de modifier un objet (création, mise à jour, suppression). Le LockFeature permet de bloquer des objets lors d'une transaction.

Passons à présent aux API telles qu'OpenLayers.

OpenLayers est une bibliothèque de code JavaScript à licence libre qui permet l'intégration et l'interaction avec des couches de données cartographiques en provenance de sources diverses. OpenLayers peut se connecter à des services tels que Google Maps, OpenStreet Maps, Bing Maps[16(*)] et aussi à des données locales fournies par des logiciels de cartographie Web supportant les normes OGC. A cet effet, il ne dépend d'aucun serveur cartographique. La bibliothèque est basée sur les technologies AJAX et permet de construire des images par tuiles en envoyant plusieurs requêtes au serveur. OpenLayers sépare les outils de la carte (l'interface cartographique) aux données cartographiques.

Etant une librairie de code JavaScript coté client, OpenLayers, pour l'utiliser, il n'est nullement nécessaire de télécharger quoi que ce soit !

OL étant open sources n'est lié à aucune technologie propriétaire d'une entreprise.

OL nous permet de construire une carte facilement en ayant la possibilité de customiser tous les aspects de la carte (les couches, les contrôles, les évènements, etc.). Nous pouvons utiliser différents serveurs de carte y compris ceux gérant les couches vecteur.

En parlant de librairie nous entendons qu'OL est une API (**Application Programmer Interface**). Nous disposons de nombreuses fonctionnalités développées par une communauté pour créer et customiser sa carte. Nous ne parons pas de zéro.

OL utilise la technologie **AJAX (**asynchronous JavaScript) pour interroger le serveur de cartes. Ainsi **OL envoie des requêtes au serveur de carte à chaque interaction avec la carte**. Puis toutes les pièces d'OL sont mises ensemble et des bouts de carte qui ressemblent à une grande carte au bout de l'échange avec le serveur de cartes.

.

Premiere carte :

```
1.<!DOCTYPE html>
2.<html lang='en'>
3.<head>
4. <meta charset='utf-8' />
5. <title>My OpenLayers Map</title>
6. <script type='text/javascript' src='OpenLayers.js'></script>
7. <script type='text/javascript'>
8.
9. var map;
10.
```

```
11. function init() {
12. map = new OpenLayers.Map('map_element', {});
13. var wms = new OpenLayers.Layer.WMS(
14. 'OpenLayers WMS',
15. 'http://vmap0.tiles.osgeo.org/wms/vmap0',
16. {layers: 'basic'},
17. {}
18. );
19.
20. map.addLayer(wms);
21. if(!map.getCenter()){
22. map.zoomToMaxExtent();
23. }
24. }
25.
26. </script>
27.</head>
28.
29.<body onload='init();'>
30. <div id='map_element' style='width: 500px; height: 500px;'>
31. </div>
32.</body>
33.</html>
```

COMPRENDRE LE CODE LIGNE PAR LIGNE :

Lignes [1] to [5]:

Positionnement du bon DOCTYPE.

Ligne [6]:

<script type='text/javascript' src='OpenLayers.js'></script>

Cela inclus la librairie l'OpenLayers. src='OpenLayers.js' indique le **chemin relatif** de la bibliothèque.

On peut aussi utiliser un **chemin absolu** : <script type='text/javascirpt' src='http://openlayers.org/api/

OpenLayers.js'></script>.

Ligne [7]:

On entame un block <script> . Nous insèrons tout notre code source ici. Comme la librairie OpenLayers est référencée en ligne 5, nous disposons de ces classes et fonctions.

Ligne [8]:

var map;

Variable globale map, accessible dans tout le code.

Ligne [11]:

Création d'une fonction nommée init. Celle-ci est appelée ligne 29 : onload='init();

Ligne [12]:

map = new OpenLayers.Map('map_element', { });

Rappelons-nous que nous avons déclaré une variable globale map. Nous la transformons ici en un objet de type map de la classe OpenLayers.Map.

La classe Map d'OpenLayers nécessite deux paramètres. Le premier map_element est l'ID de la page HTML ou la carte apparaitra. Le second paramètre {} est une liste d'options notées sous la forme d'une paire {key: value}

Ligne [13]:

var wms = new OpenLayers.Layer.WMS(

Nous créons ici une couche de la carte notée map. Nous utilisons la sous classe WMS de la classe Layer. WMS : **Web Map Service**, est une norme définie par l' **Open Geospatial Consortium** (**OGC**).

Ligne [14]:

'WMS Layer Title',

C'est le premier paramètre passé : le titre de la couche. Ce titre peut être utilisé dans des contrôles tels que « la liste des couches « que nous utiliserons.

Ligne [15]: 'http://vmap0.tiles.osgeo.org/wms/vmap0',

L'URL est le second paramètre que la couche WMS reçoit. Pour l'instant, nous récupérons une couche publique d'OSGEO. Plus tard nous positionnerons nos propres couches dans ce paramètre d'URL.

Ligne [16]:

{layers: 'basic'},

Le troisième paramètre est un objet anonyme qui contient les propriétés de la couche. Même notation qu'en ligne [12]. Il s'agit de la notation d'objet JavaScript de type {key1:value1, key2:value2}.

Ligne [17]: { }

Le 4e paramètre est optionnel, il reprend les options notées sous la forme type {key1:value1, key2:value2}. Par exemple l'opacité peut être notée : {opacity: .8} pour 80 pourcent d'opacité.

Ligne [18]:);

Finalisation de l'objet créé

Ligne [20]: map.addLayer(wms);

La couche wms est ajoutée à la carte (map).

En cas d'ajout de plusieurs couches, nous pouvons utiliser la méthode addLayer**s :** map.addLayers([layer1, layer2, ...]);

Ligne [21] - [23]:

```
if(!map.getCenter()){

map.zoomToMaxExtent();

}
```

On spécifie ici la zone où la carte sera visible : map.zoomToMaxExtent() qui zoom à l'extension maximale de la carte.

Ligne [24]: }

Fin de la fonction init().

Lignes [26], [27]:

Fin des tag script et head.

Ligne [29]: <body onload='init();

Appel de la function init ().

Ligne [30] and [31]:

Pour réaliser une carte OpenLayers, nous avons besoin d'un élément HTML où la carte sera affichée.

Lignes [32] and [33]: Finalisation de la page en refermant les tags restant.

OpenLayers est une API assez facile à programmer. J'ai pu produire des applications intéressantes dès le premier mois. Nous avons programmé avec la version 2 qui va rapidement devenir obsolète du fait de l'arrivée de la version 3 qui est en béta.

Cependant, elle manqué d'outils pour tout ce qui est GUI ; l'interface utilisateur. Cette partie est beaucoup mieux décrite par ExtJs.

EXT JS / FRAMEWORK JAVASCRIPT, UNE AUTRE API PUISSANTE

Disons quelques mots sur cette api.

Elle est orientée objet, comporte de nombreux contrôles prédéfinis, la documentation est plus compréhensible que celle d'OpenLayers, et enfin il est aisé d'adapter son code source à une version sur mobile (aspects non vus lors du stage).

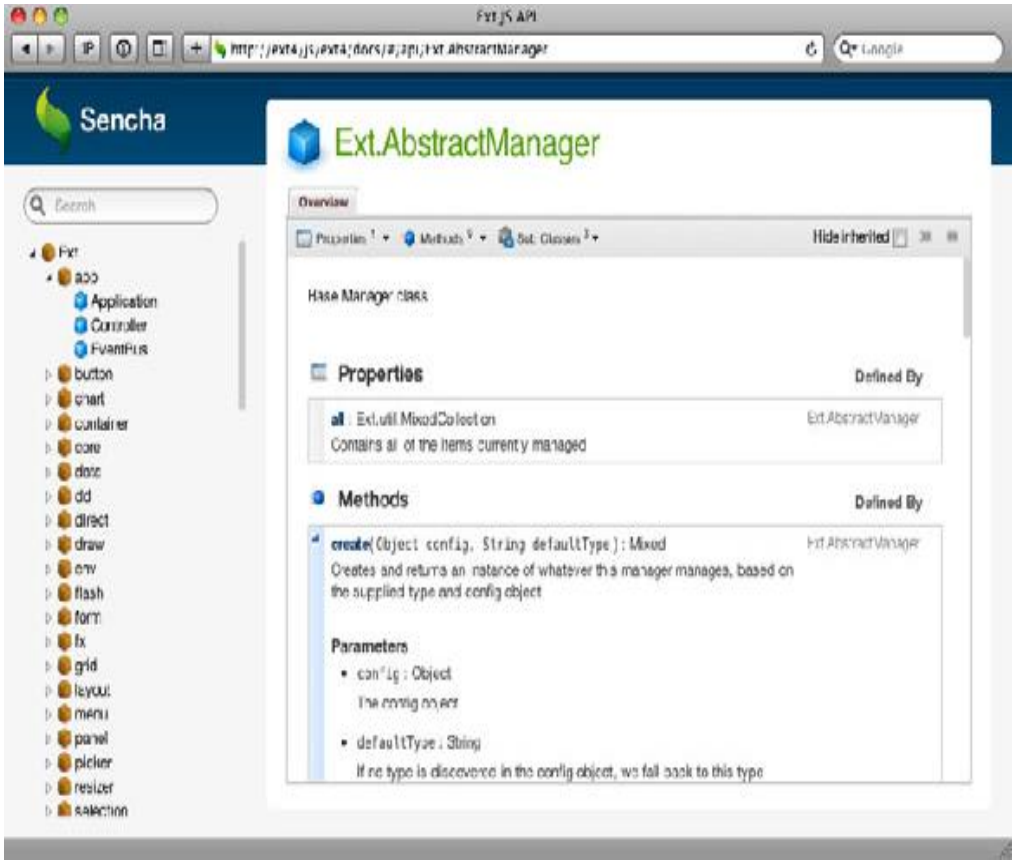
Je me suis auto-formé à cette API en 15 jours. C'est assez ardu mais j'ai réussi à faire fonctionner les nombreux exemples disponibles sur le net. J'utilisais, lors de mon autoformation les versions 3 et 4. Mais la version qui est utilisée dans l'OpenGeo Suite est la version 3.

Faire fonctionner des modules en « dur » ne m'a pas posé de problèmes. Par contre lorsqu'il a fallu puiser des données dans une BD pour remplir un formulaire, ça a été beaucoup plus compliqué.

Nous verrons plus loin que nous avons besoin de cette API car le SDK de la suite OpenGeo l'utilise.

Nous observons ci-dessous que la documentation présente sur le net est bien structurée.

Chaque classe est décrite avec ces propriétés, ces méthodes et son constructeur. On retrouve aussi aisément les classes dont elle hérite.



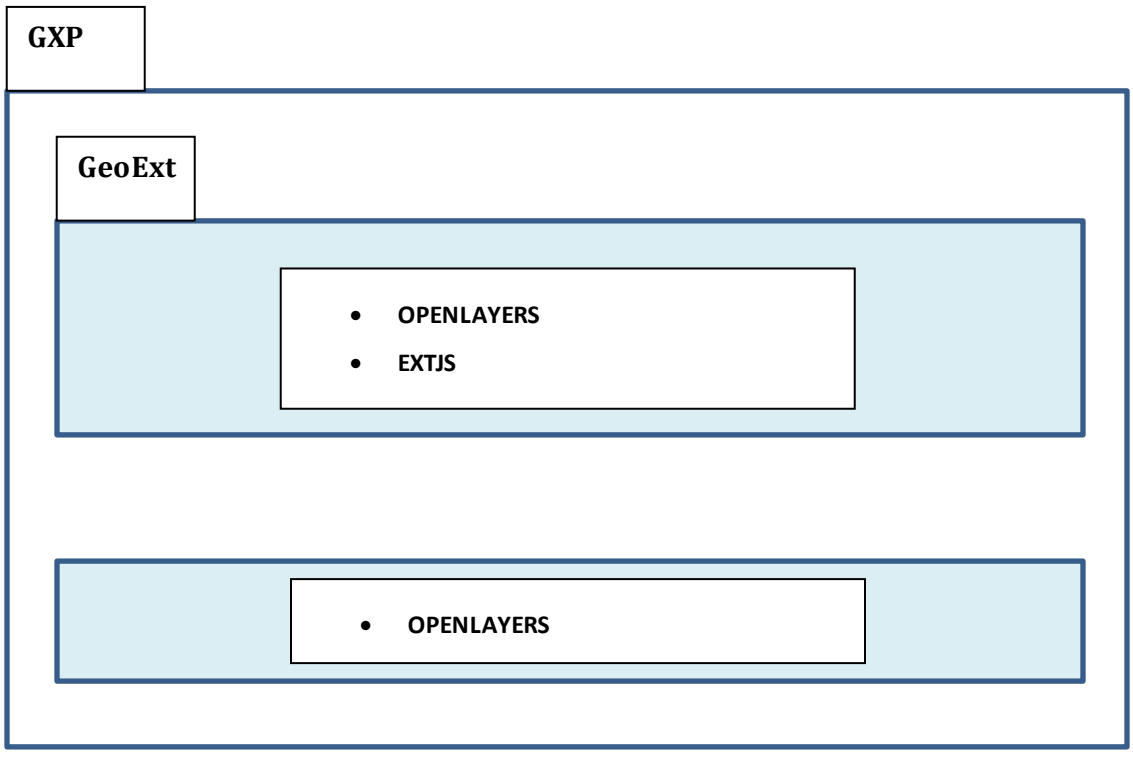
SDK DE L'OPENGEO SUITE

Le SDK de l'openGeo Suite est le moyen de programmer des applications sur cette suite. Il comporte plusieurs API.

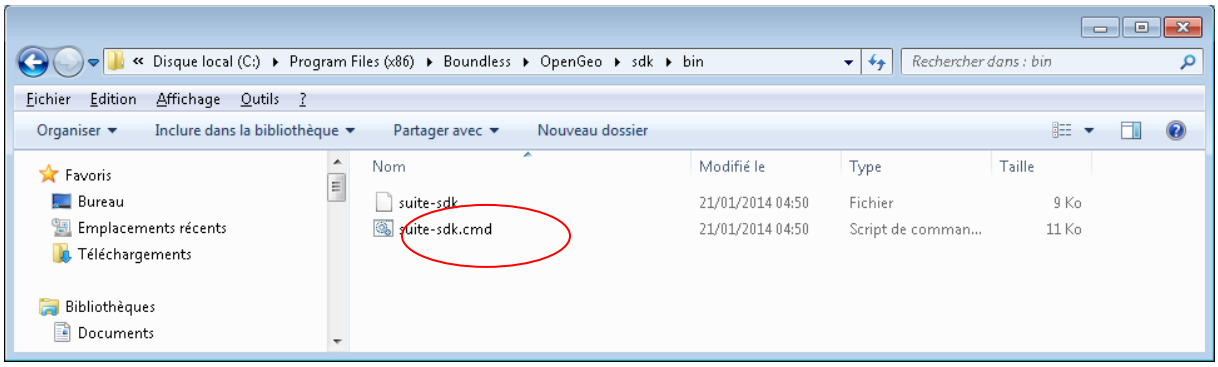
GXP est composé de GeoExt et d'OpenLayers. Ce qui rend l'architecture complexe. A chaque étape du développement, il est essentiel de savoir où se situe notre donnée, gxp GeoExt, ExtJs ou OpenLayers. Ces différentes documentations ne communiquent pas entre elles. De gxp, Nous serons amenés à lire la documentation de GeoExt, puis celle d'OpenLayers. Cette gymnastique demande un certain temps d'apprentissage. Nous y reviendrons plus bas.

GeoExt est une librairie JavaScript qui permet de créer des interfaces cartographiques riches. C'est la combinaison des librairies OpenLayers pour ses fonctionnalités géospatiales et ExtJs pour ses outils d'interface

Le schéma représentant l'emboitement de ces technologies est le suivant :



Ansi, une variable de gxp peut hériter à la fois des méthodes public de GeoExt et OpenLayers, donc en final d'OpenLayers, de GeoExt et de ExtJs. J'ai mis un cerain temps avant d'avoir en tête cet empilement de couches.

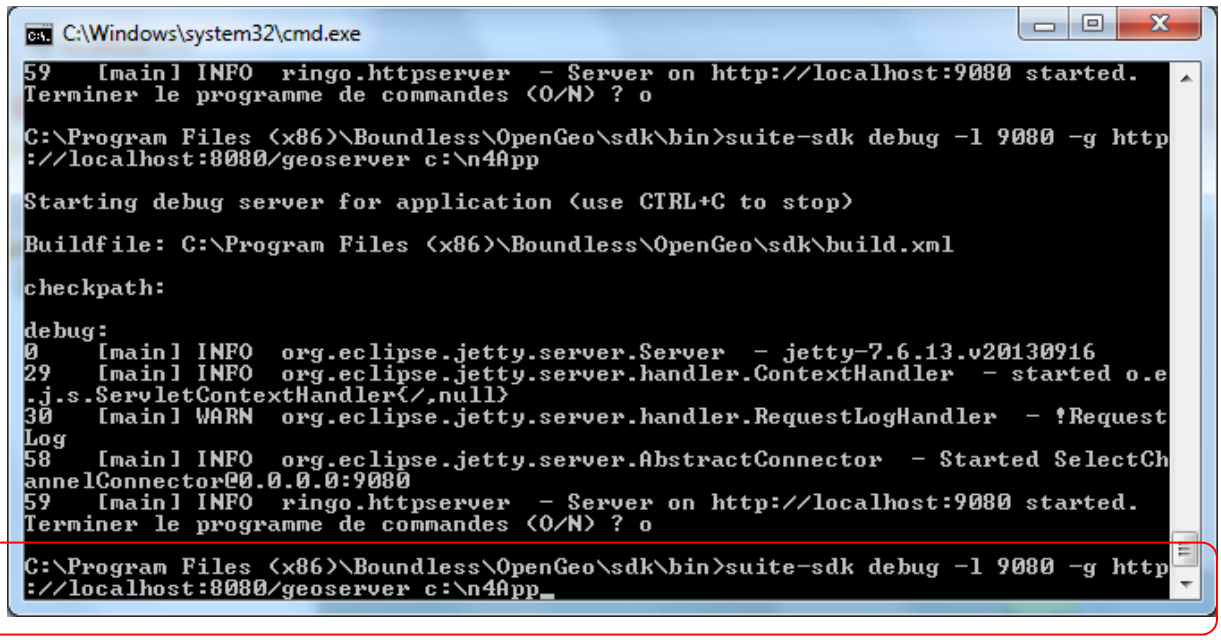


Pour débuguer les programmes, nous tapons les instructions suivantes sous la fenêtre de commande

Suite-sdk debug –l 9080 –g http://localhost:8080/geoserver c:\n4App

Ceci après avoir créé une application sous c:\n4App.

Pour lancer l'application sur le serveur, nous tapons **http://localhost:9080**



Après cette présentation de l'architecture, nous verrons comment je me suis approprié ces technologies au travers différents programmes. Puis nous mettrons l'accentsur les difficultés rencontrées et les moyens de les résoudre.

MES REALISATIONS EN TERMES DE DEVELOPPEMENT : LE « SNAP »

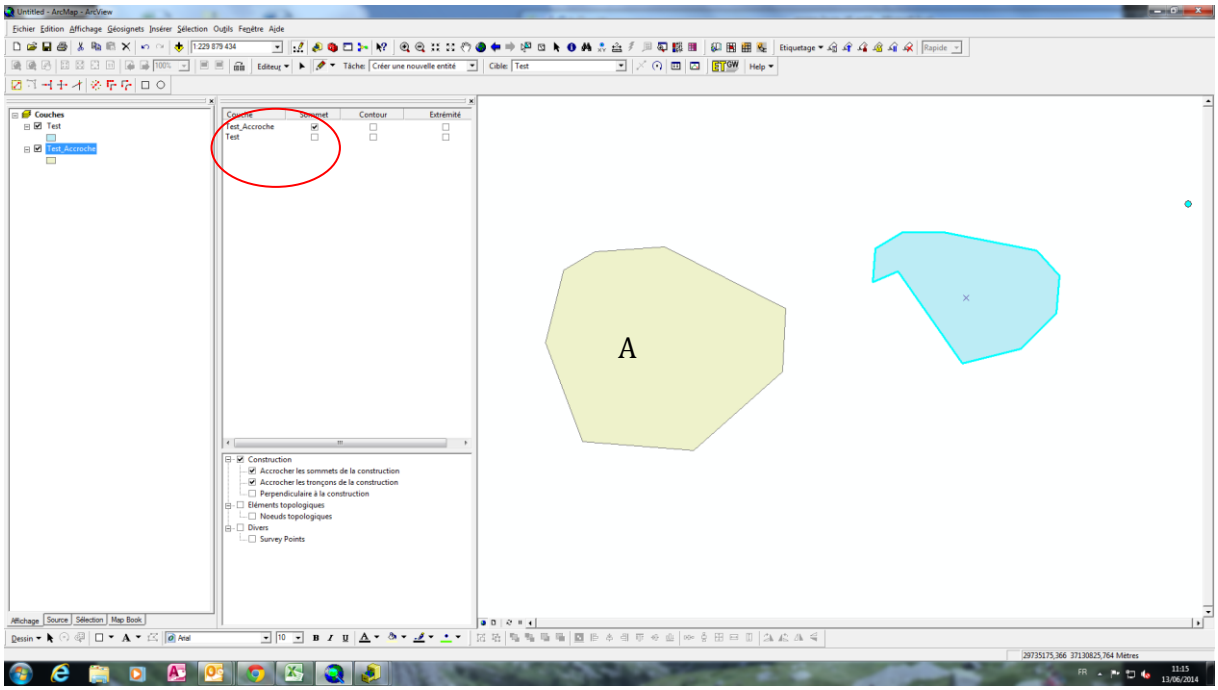
La majeure partie de mon temps a été consacré à la notion de Snap. Nous rappellerons en quoi ça consiste sous ArcGis. Puis nous décrirons nos programmes sous OpenLayers et sous le SDK OpenGeo.

Rappelons que l'action de snaper consiste à partir d'un point, d'une ligne ou plus souvent d'un polygone, à raccrocher d'autres surfaces de ce dernier.

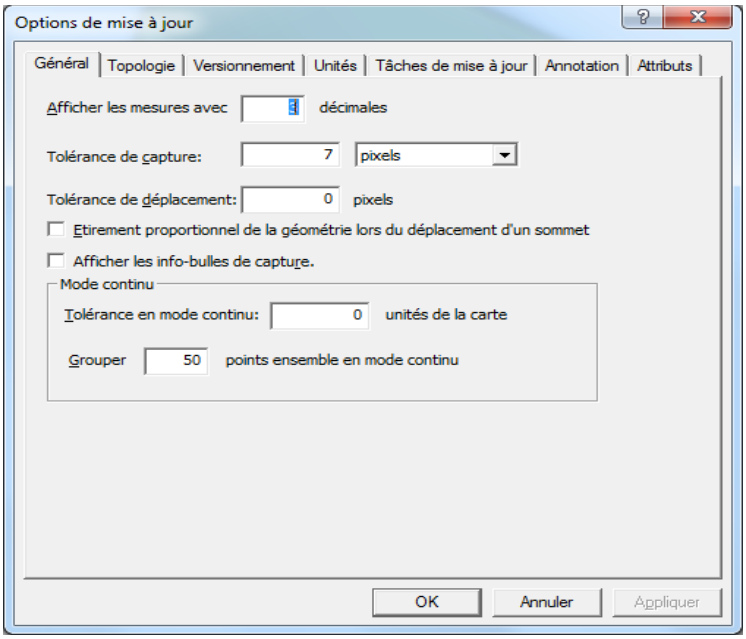
Cela est très utile pour délimiter des parcelles adjacentes sans avoir à relier tous les points un par un.

SOUS ARCGIS

Se mettre en mode Edition, puis ouvrir une session de mise à jour :

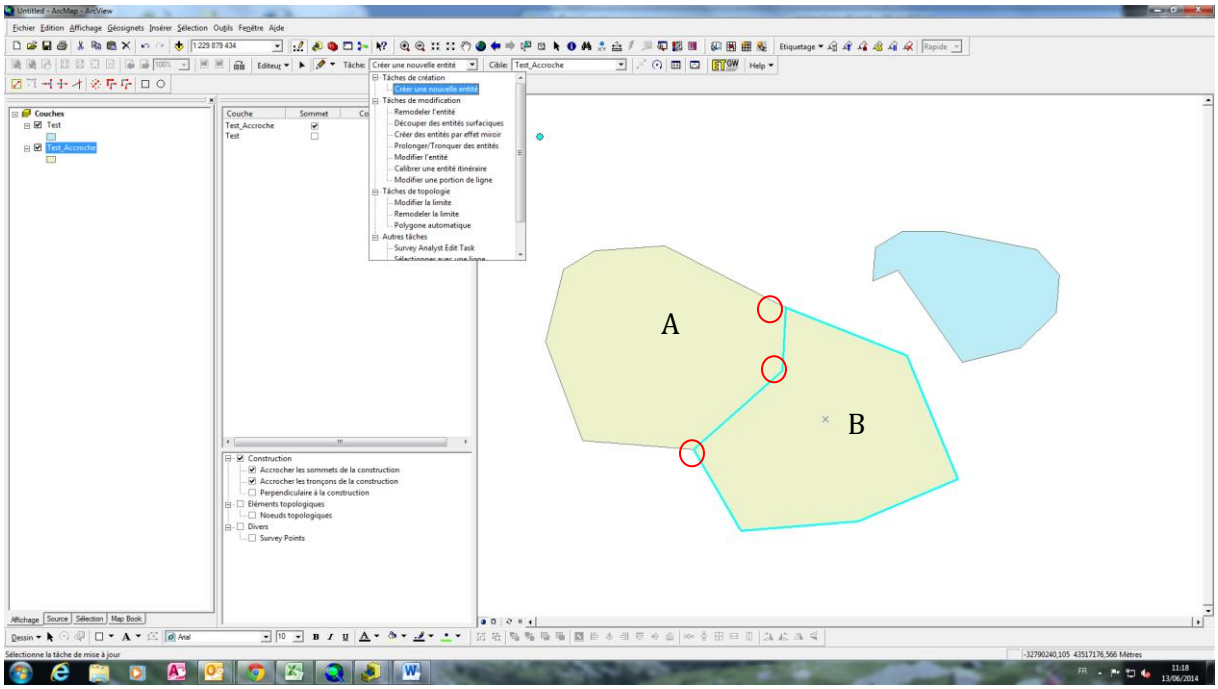


On peut se raccrocher au sommet, au contour ou à l'extrémité du polygone initial en gris nommé A. Ici on se raccroche aux sommets.



Dans les options on peut définir la tolérance. C'est-à-dire la distance à partir de laquelle se fera l'accroche, en pixels.

Ici, nous voyons le deuxième polygone gris entièrement collée aux sommets du polygone initial.



Les polygones A et B sont « collés » à partir des sommets de A entourés en rouge.

SOUS OPENLAYERS :

ON PART DE L'EXEMPLE FOURNI DS LA DOCUMENTATION SUR LE SNAPP

```
vectors = new OpenLayers.Layer.Vector("Lines",
    isBaseLayer: true,
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "data/roads.json",
        format: new OpenLayers.Format.GeoJSON()          }),
    styleMap: styles,
    maxExtent: new OpenLayers.Bounds(
        1549471.9221, 6403610.94, 1550001.32545, 6404015.8          ),
    renderers: renderer          });

map.addLayer(vectors);

// configure the snapping agent

snap = new OpenLayers.Control.Snapping({layer: vectors});
map.addControl(snap) ;
snap.activate();
```

COMMENTAIRE DU CODE SOURCE

On définit tout d'abord une variable **vector** qui est de type OpenLayers.Layer.Vector. Ce sera la couche qui sera affichée et que nous rendons snapable.

Une variable **map** de type Map d'OpenLayers a été déclarée plus haut (non visible ici). Elle contient la carte qui sera affichée avec sa couche vectors grâce à la ligne **map.addLayer(vectors).**

Puis on définit une variable **snap** qui est de type Control de l'API OpenLayers. Dans ce controle, ({**layer**: **vectors**}); définit la couche qui est rendue snapable par ce contrôle.

Enfin, nous ajoutons le contrôle snap à la carte snap par les instructions: **map.addControl**(**snap**).

Ainsi la carte map comporte un nouveau contrôle (snap) qui rend la couche vectors snapable.

Ces différentes instructions sont conformes à la documentation du contrôle Snaping. Cette documentation est accessible en ligne en tapant tout simplement sur Google : OpenLayers.Control.Snapping.

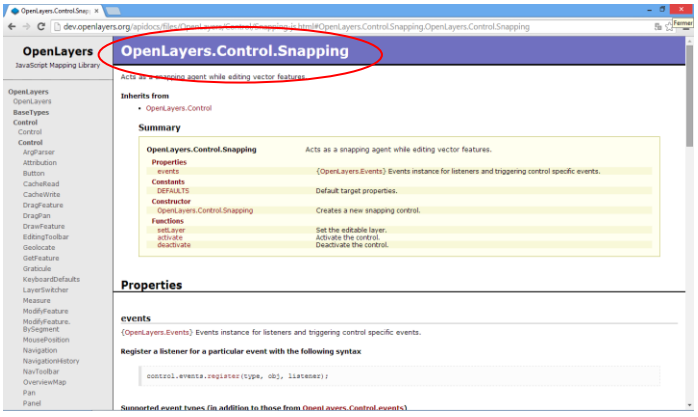
Nous affichons un extrait de la documentation sur le snaping contrôle en deux parties.

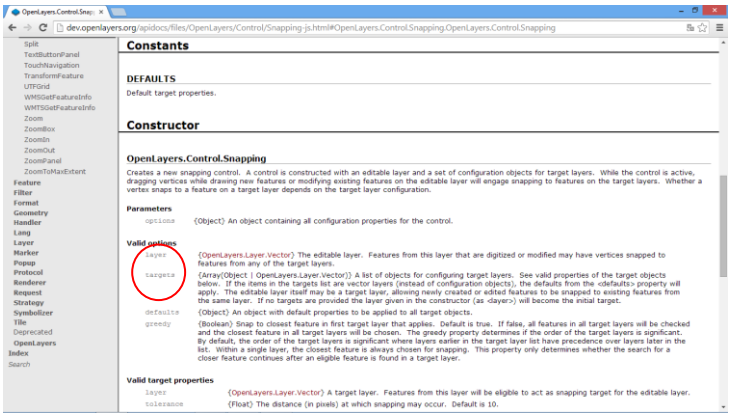
On observe sur la partie de gauche que la syntaxe de l'appel de ce contrôle est bien OpenLayers.Control.Snapping et qu'il existe une fonction **activité** () pour déclencher effectivement le contrôle sur la carte. On peut décider de désactiver le contrôle snaping le cas échéant par la fonction deactivate(). Nous ne servirons pas de cette dernière fonction.

Dans la partie droite, On peut lire ce que comporte le constructeur avec ces paramètres dans la partie "Valid options". On peut y lire qu'au moins deux paramètres sont attendus : **layer** et **targets**. Observons le type attendu ces ces deux paramètres. layer est de type OpenLayers.Layer.Vector . Dans notre exemple, on place la variable vector dans layer par l'instruction ({**layer**: **vectors**}. La variable est bien de type Vector, donc tout va bien.

Quant au paramètre **targets**, il est absent dans notre source car nous ne définissons qu'une seule couche à snaper. Ce paramètre est de type Array(Object | OpenLayers.Layer.Vector). Il s'agit donc d'un tableau de couches qui sont rendues snapable. Exemple de syntaxe : **targets**: [wfs, wfs2, layer3, couche_active_var].

Dans cet exemple 4 couches sont rendues snapable. Notons la notation [ ] qui est conforme à celle des tableaux.





Cette solution ne nous convient pas, tout est en « dur » et rien n'est dynamique, si on rajoute une couche à la volée, elle ne sera pas snapable. Ici seule la couche vectors est snapable. Pour cela on adapte le code source précédent.

L'idée de cette évolution est d'avancer pas à pas, en rajoutant une couche suite au click d'un bouton. On avance vers une solution entièrement dynamique où l'on pourrait ajouter une couche à la « volée ».

Nous affichons ici un extrait de bouts de code réellement utile à la compréhension.

```
function alert_on_changelayer (event) { alert('You change layer'); };


function  liste_layers (){
var mLayers = map.layers;
alert ('Nb de layer = ' + mLayers.length )
for(var a = 0; a < mLayers.length; a++ ){
alert("layer = " + mLayers[a].name );
};
};


map = new OpenLayers.Map('map_element',
 {eventListeners:{'changelayer': alert_on_changelayer}
, controls: [
new OpenLayers.Control.PanZoom(),
new OpenLayers.Control.Navigation()
]
});

map = new OpenLayers.Map('map_element',
{eventListeners:{'changelayer': alert_on_changelayer},
controls: [new OpenLayers.Control.PanZoom(),
new OpenLayers.Control.Navigation()]
});


var layer3 = new OpenLayers.Layer.Vector("WFS", {
 strategies: [new OpenLayers.Strategy.BBOX()],
 protocol: new OpenLayers.Protocol.WFS({
 url:  "http://demo.opengeo.org/geoserver/wfs",
 featureType: "tasmania_roads",
 featureNS: "http://www.openplans.org/topp"
    })
});

var custom_button_func = function() {
map.addLayers([layer3]);

var snap = new OpenLayers.Control.Snapping (
{layer: couche_active_var,
targets: [wfs, wfs2,  layer3, couche_active_var]}) ;

map.addControl(snap);
snap.activate();
snap_positionne = 1;
};
```

```
var my_button  = new OpenLayers.Control.Button({
displayClass: 'olControlCustomButton',
trigger: custom_button_func });

var control_panel = new OpenLayers.Control.Panel({});
control_panel.addControls([my_button]);
map.addControl(control_panel);
control_panel.moveTo(new OpenLayers.Pixel(250,0));
```

COMMENTAIRE DU SOURCE :

Pour l'instant nous n'utilisons pas l'évènement changement de couche. C'est ce qu'on réalise dans la suite ; Pour la suite on a besoin de ne sélectionner que la couche active comme étant éligible au snap.

Nous observons la présence de trois fonctions :

1. **liste_layers()**

Il s'agit d'une fonction qui affiche à l'écran l'ensemble des couches présentes dans la carte nommée map, à des fins de débogage.

On récupère tout d'abord les couches présentes dans la carte map.layers dans la variable mLayers. Celle-ci est un tableau de couches conformément au type OpenLayers.map.layers. Puis on boucle sur ce tableau, pour afficher le champ name de chaque couche à l'écran

2. **alert_on_changelayer** (event) :

Fonction utilisée plus loin lorsque nous lèverons l'évènement changement de couche qui affiche juste un message à l'écran (à des fins de débogage)

3. **custom_button_func**

Nous commençons par ajouter la couche **layer3** à la carte map : map.addLayers([**layer3**]);

Puis nous créons un contrôle une variable **snap** qui contient un contrôle de type snaping agent. Remarquons que la couche layer3 qui vient d'être crée est éligible aussi au snaping, tout comme les autres couches crées en « dur »

targets: [wfs, wfs2, **layer3**, couche_active_var]}) ;

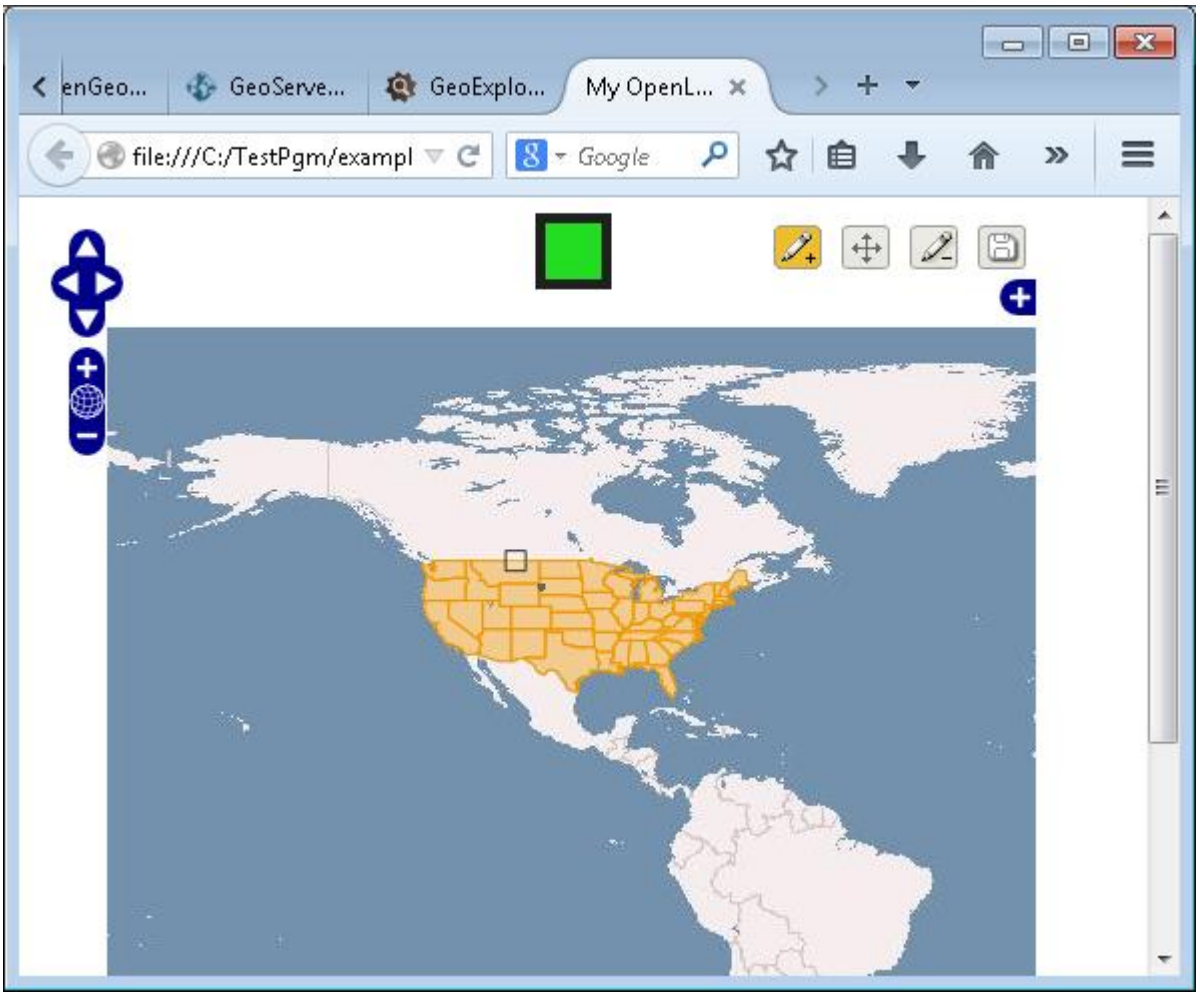
Puis on ajoute le contrôle de snaping snap à la carte map, et on active le snaping.

Code du bouton :

1. **my_button** contient un contrôle de type bouton. Lorsqu'on clique dessus il appelle la fonction **custom_button**_func grâce au mot clé trigger
2. **control_panel :** Le bouton qui précédé est inclus dans ce contrôle panel. On commence par ajouter le bouton au contrôle Panel. Puis on ajoute le contrôle Panel à la carte.

Nous progressons vers la solution mais c'est encore insuffisant. Certes, on ajoute une couche via un bouton en la rendant snapable. Mais on voudrait que cette couche soit ajoutée de manière entièrement dynamique. De plus la couche ainsi sélectionnée doit être la seule qui soit snapable.

Carte obtenue :



On observe ici que la couche states est snapable. Le bouton vert ajoute la couche Tasmanie et rend toutes les couches snapable. Le bouton activé en orange permet de saisir des polygones « draw feature »

ADAPTATION 2

```
function alert_on_changelayer (event){
var snap = new OpenLayers.Control.Snapping({layer: wfs, targets: [wfs, event.layer]}) ;
map.addControl(snap) ;
snap.activate();          };


map = new OpenLayers.Map('map_element', {eventListeners:
{'changelayer': alert_on_changelayer}
, controls: [
new OpenLayers.Control.PanZoom(),
new OpenLayers.Control.Navigation()
]
});
```

On lève toujours ici l'évènement **'changelayer'** dans la création de la carte map**.** On appelle toujours la fonction **alert_on_changelayer** mais son code a été entièrement revu.

Cette fois on ne construit le contrôle snaping que lors de l'évènement 'changelayer', et on positionne la couche actuelle dans l'option target par le code, targets: [wfs, **event**.layer].

Cette solution entièrement dynamique a nécessité un temps assez long avant d'être trouvé. C'est le type de l'objet event qui posait problème. Ce n'est qu'un objet qui est du type de l'évènement levé mais il fallait penser à le suffixer de .layer pour obtenir la couche active.

Nous obtenons une **solution très satisfaisante** mais un détail reste à régler. Ce détail a nécessité plus de trois semaines d'investigations.

PROBLEME DES COUCHES WFS LOCALES

A cette étape la snap dynamique fonctionne mais uniquement avec des couches distantes. Il ne s'agit pas de couches stockées en local sur GeoServer de ma machine.

Si on ajoute une couche comme celle des départements Français, stockée sur GeoServer, elle ne s'affiche pas.

Déclaration de la couche vecteur des départements Français :

```
dep_france = new OpenLayers.Layer.Vector                    (

'dep_france_l93',

{protocol: new OpenLayers.Protocol.WFS({
        url: "http://192.168.1.126:8080/geoserver/Couches/ows?service=wfs",
        featureType: 'dep_france_l93',
        version: "1.1.0",
         srsName: "EPSG:2154" }),
                                    });
    map.addLayers([dep_france]);
```

Après une analyse minutieuse des causes possibles de non affichage des couches WFS en local, nous avons établi un guideline pour contourner ce problème.

Au début nous pensions que l'URL était mal notée. Ce n'était pas le cas, d'autant que les couches WMS s'affichaient bien.

Nous avons finalement résolu le problème en près de trois semaines en consultant les différents forums qui traitent de la question. Chaque forum donnait une partie de la solution. En particulier, la FAQ ci-dessous nous a donné de bonnes indications.

Il s'agit là d'un des **succès d'étape**. Le problème était assez ardu. Beaucoup de monde s'interroge à ce sujet sur les forums du web.

Dans la solution, il est question de **proxy**. Nous expliciterons partiellement cette question en annexe.

A présent, recensons les différents détails à vérifier pour que l'ensemble fonctionne.

DETAILS A VERIFIER POUR AFFICHER UNE COUCHE WFS DE GEOSERVER AVEC OPENLAYERS

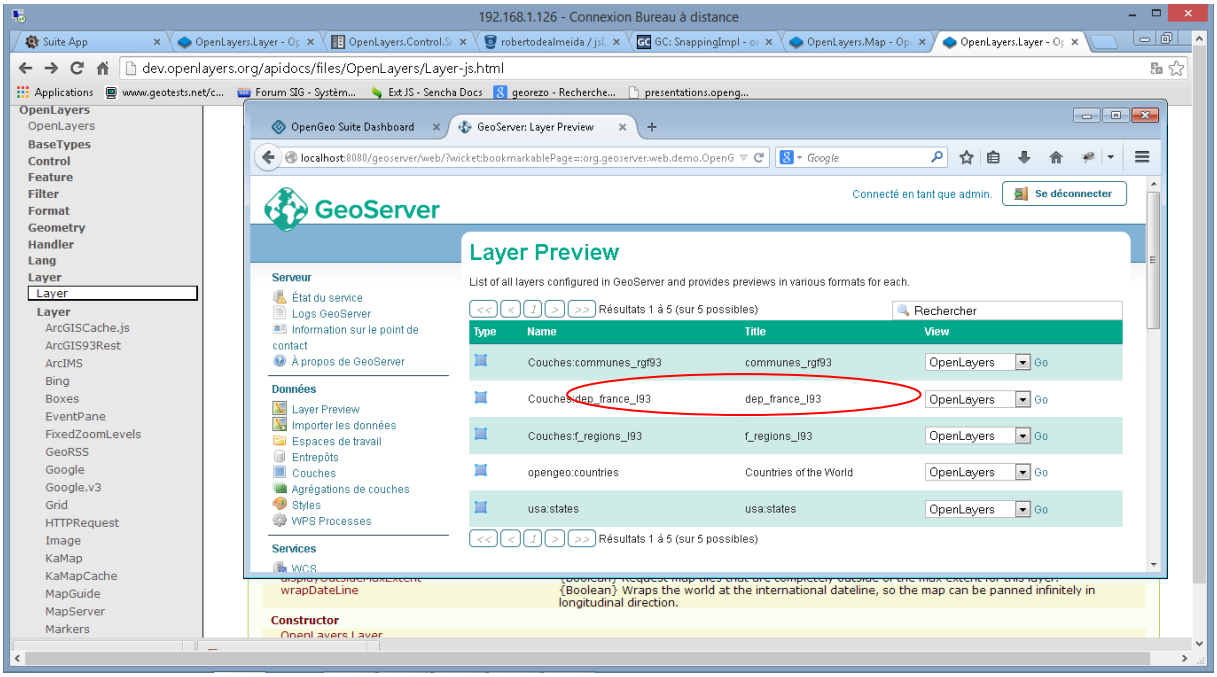
- Installer Apache
  Ce n'était pas le cas jusque présent.

- Mettre ma page html ainsi que mon code source JavaScript sur le répertoire **htdocs** du serveur apache

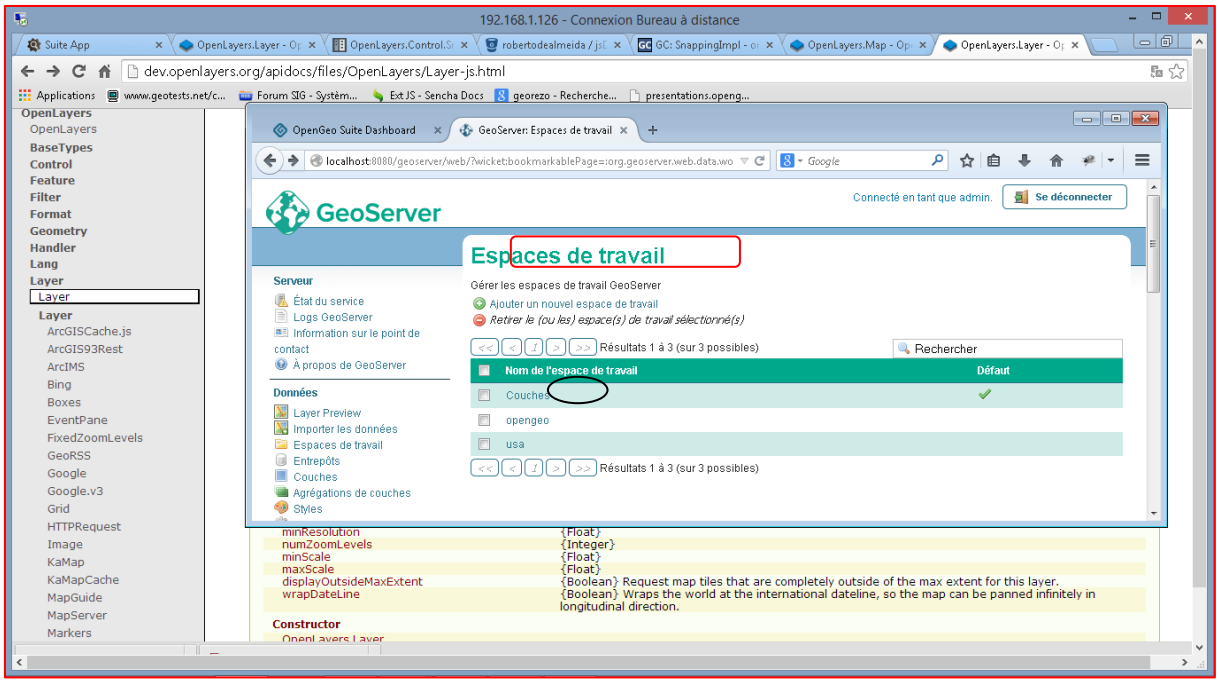
- Copier sur htdocs les répertoires de l'installation d'OpenLayers (en particulier /lib et /theme)

- Mettre un **featureNS** en concordance avec celui de GeoServer :

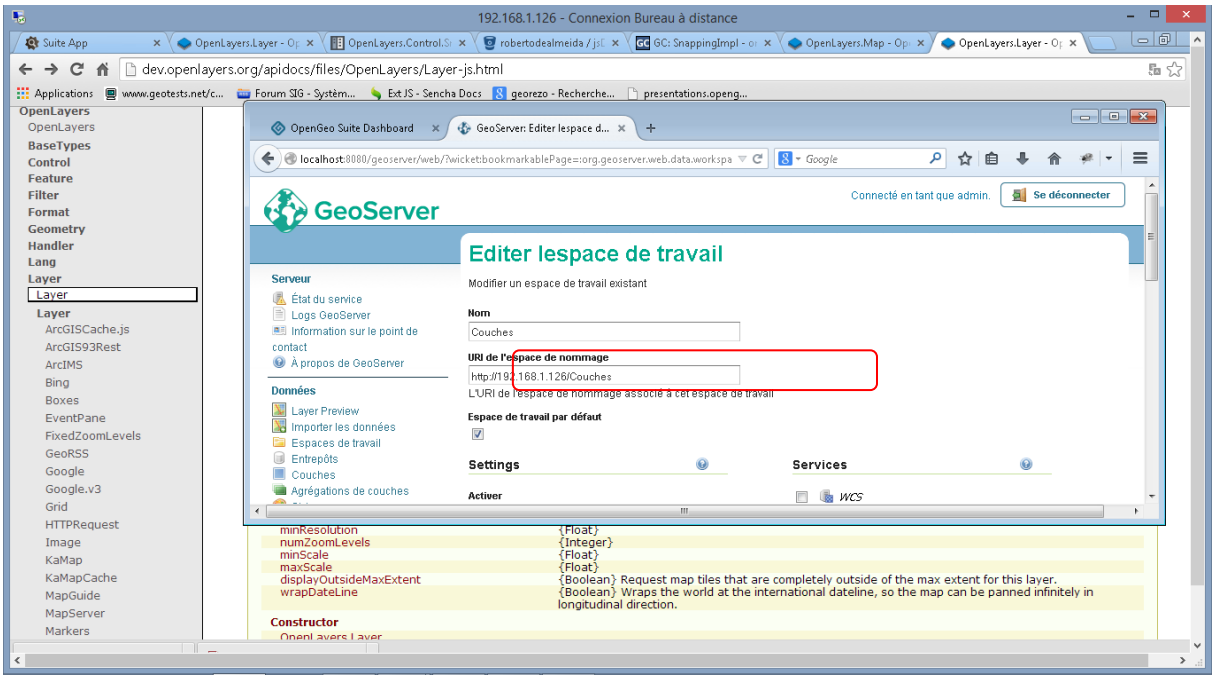
new OpenLayers.Layer.Vector("dep_france_l93", {
strategies: [new OpenLayers.Strategy.BBOX()],
protocol: new OpenLayers.Protocol.WFS({
version: "1.1.0",
url: "http://192.168.1.126:8080/geoserver/Couches/ows?service=WFS",
featureType: "dep_france_l93", *nom de la couches s GeoServer*
featureNS: http://192.168.1.126/Couches ***adresse ip de ma machine/NS ss GeoExplorer***



Voici l'ensemble des couches sur GeoServer avec leur titre différent du nom



Voici les différents espaces de travail de GeoServer, si on clique sur Couches :



On obtient le champ qu'il faut mettre dans ***featureNS*** *qui est ici dans* **URI de l'espace de nommage**

- Activer **proxy.cgi** dans le source JavaScript, voir la FAQ OpenLayers : http://trac.osgeo.org/openlayers/wiki/FrequentlyAskedQuestions#ProxyHost

OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";

PROXYHOST FAQ

Why do I need a ProxyHost?

Due to security restrictions in Javascript, it is not possible to retrieve information from remote domains via an XMLHttpRequest.

Classes like WFS and GeoRSS use XMLHTTPRequest to get their data. If they are querying a remote server (anything other than the machine hosting your page), you must install a proxy script somewhere web accessible on that machine. See below for how to set up your own ProxyHost.

If the OpenLayers.ProxyHost variable is not set to a valid proxy host, requests are sent directly to the remote servers. In most cases, the result will be a security exception, although this exception often occurs silently.

How do I set up a ProxyHost?

An example proxy host script is available here: trunk/openlayers/examples/proxy.cgi

For the standard Apache configuration, you would place proxy.cgi into your /usr/lib/cgi-bin/ directory.

Once a proxy host script has been installed, you must then edit the OpenLayers.ProxyHost variable to match that URL.

Given the above standard Apache configuration:

```
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
```

If you have done something like this, you should be able to visit:

http://YourDomain.example.com/cgi-bin/proxy.cgi

The resulting content at that page should be the openlayers.org website.

If you get a 404 error instead, either the proxy script is not in the right location, or your webserver is not configured correctly.

- Changer **allowedHosts** dans **proxy.cgi,** ajouter localhost avec les bons ports exemple **:**

  #!c:\Python27\python.exe –u

  ***allowedHosts** = ['192.168.1.126:80', 'localhost:80', 'localhost', 'localhost:8080','192.168.1.126:8080',* *'labs.metacarta.com', 'world.freemap.in', 'prototype.openmnnd.org', 'geo.openplans.org', 'www.openlayers.org', 'openlayers.org', 'sigma.openplans.org', 'demo.opengeo.org', 'www.openstreetmap.org', 'sample.avencia.com']*

- Vérifier que le fichier python.exe est bien présent sur le répertoire c:\Python27 Sous Windows : mettre dans anti slash pour le chemin d'accès du fichier

- mettre proxy.cgi modifié dans le répertoire **cgi-bin** d'Apache :

- Modifier **httpd.conf** tel que celui sur mon poste, voir le toto : http://vijaysambhe.wordpress.com/2012/10/12/openlayers-cross-domain-configure-proxyhost-on-windows/

Une fois **ce problème résolu**, il reste à adapter ce que nous avons réalisé en OpenLayers pour que ça tourne aisément sur l'OpenGeoSuite.

Je me suis ensuite auto-formé à une nouvelle API compatible avec l'OpenGeoSuite: Ext JS.

Présentons à présent la partie développée sur le SDK OpenGeo

SDK OPENGEO SUITE

1ERE VERSION : RENDRE SNAPABLE TOUTES LES COUCHES

Une application du SDK utilisés des bibliothèques de classes ou **plugin** qui sont à déclarer en entrée. Notre application utilise les plugins suivants :

```
/**
 * Add all your dependencies here.
 *
 * @require widgets/Viewer.js
 * @require plugins/LayerTree.js
 * @require plugins/OLSource.js
 * @require plugins/OSMSource.js
 * @require plugins/WMSCSource.js
 * @require plugins/ZoomToExtent.js
 * @require plugins/NavigationHistory.js
 * @require plugins/Zoom.js
 * @require plugins/AddLayers.js
 * @require plugins/RemoveLayer.js
 * @require RowExpander.js
 * @require plugins/FeatureManager.js
 * @require plugins/FeatureEditor.js
 * @require plugins/WMSGetFeatureInfo.js
 * @require plugins/WMSCSource.js
```

```
 * @require plugins/SnappingAgent.js
 * @require plugins/LayerProperties.js
 */
```

On remarquera la présence du plugin **SnappingAgent.js** que nous modifierons par la suite. Ces différents plugin apparaissent dans le source à l'intérieur du mot clé **tools**:

```
    // configuration of all tool plugins for this application

    tools: [{
        ptype: "gxp_layertree",
        outputConfig: {
            id: "tree",
            border: true,
            tbar: [] // we will add buttons to "tree.bbar" later
        },
        outputTarget: "westpanel"
    }, {
        ptype: "gxp_addlayers",
        actionTarget: "tree.tbar"
    }, {
        ptype: "gxp_removelayer",
        actionTarget: ["tree.tbar", "tree.contextMenu"]
    }, {
        ptype: "gxp_zoomtoextent",
        actionTarget: "map.tbar"
    }, {
        ptype: "gxp_zoom",
        actionTarget: "map.tbar"
    }, {
        ptype: "gxp_navigationhistory",
        actionTarget: "map.tbar"
    },
            {
        ptype: "gxp_layerproperties",
        actionTarget: "map.tbar"
    },
{
    ptype: "gxp_featuremanager",
    id: "states_manager",
    paging: false,
    layer: {
        source: "local",
        name: "usa:states"
    }
},
{
    ptype: "gxp_featureeditor",
    featureManager: "states_manager",
          autoActivate : true , //
    autoLoadFeature: true,
```

```
        //snappingAgent: "snapping-agent"
}
/* ,

{
   ptype: "gxp_snappingagent",
   id: "snapping-agent",
   targets: [{
      source: "local",
      name: "usa:states"
   }

        ]
} */
```

Puis nous définissons une variable app de type **gxp.Viewer** qui permettra d'afficher la carte.

var app = new gxp.**Viewer**({…..

Au début nous rendons la couche usa:states snapable en l'intégrant dans les layers du snaping agent :

```
{
   ptype: "gxp_snappingagent",
   id: "snapping-agent",
   targets: [{
      source: "local",
      name: "usa:states"    }
   ]
}
```

Puis nous rendons aussi de manière inconditionnelle la couche des départements Français snapable en faisant appel à la fonction **addSnappingTarget** du plugin SnappingAgent.

```
var record = "Couches:dep_france_l93";
var snapTarget = Ext.apply({}, snapTarget);
snapTarget.source = "local" ;
snapTarget.name = record ;


 app.tools['snapping-agent'].addSnappingTarget (snapTarget);
```

La syntaxe de la dernière ligne est un peu ardue.

App est l'application principale. Elle comporte une partie **tools** qui est en fait un tableau d'outils, ici nous nous concentrons sur l'outil 'snapping-agent' qui comporte la fonction addSnappingTarget. Celle-ci est décrite dans le code source du plugin **SnapingAgent**.

Nous pouvons, le cas échéant, modifier ce code source pour l'adapter à nos besoins. Rien n'est figé, les Classes définies dans les plugin peuvent être modifiés. Nous utiliserons ce procédé plus loin.

Notons que la variable snapTarget est un record avec ses champs source et name. Ici tout est codé "en dur".

Cette solution fonctionne mais rien n'est dynamique. Les deux couches snapables le sont de manière inconditionnelle. Si nous ajoutons une nouvelle couche à la carte elle ne sera pas snapable. Nous souhaitons rendre la couche active snapable.

Ce qui nous amène à une deuxième version, un peu plus dynamique.

2E VERSION : RENDRE SNAPABLE LA COUCHE ACTIVE

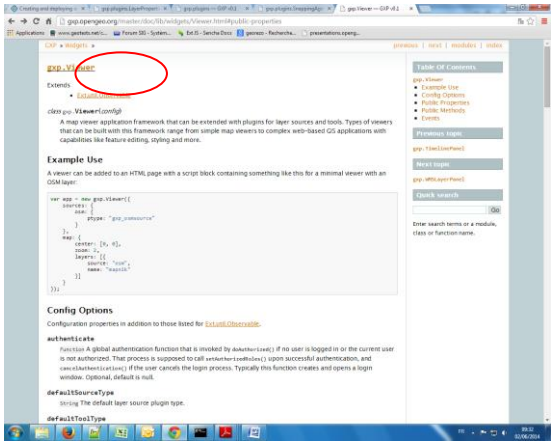
Nous allons définir l'action de snap à l'intérieur d'un bouton pour dériver d'une structure de code linéaire. A l'intérieur du code de la variable map nous décrivons le bouton avec toutes ces caractéristiques.

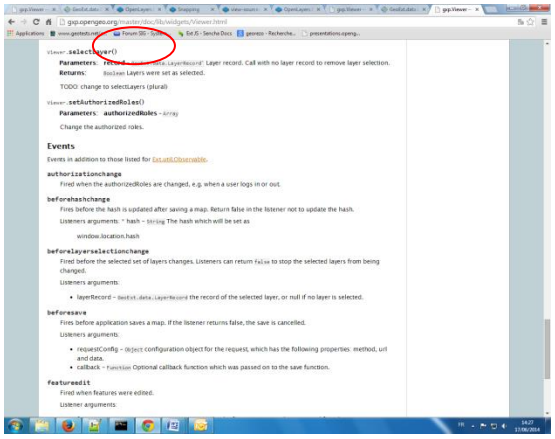
```
{
xtype:'button',
text: 'Couche active',
handler : function(){
        var snapTarget= Ext.apply({}, snapTarget);
    alert( 'getLayer().name = ' + app.selectedLayer.getLayer().name );
        app.tools['snapping-agent'].activate() ;
        alert ( app.tools['snapping-agent'].active );

        //var record = app.selectedLayer.getLayer();
        record = app.selectedLayer.get("name");
    alert ('record  = ' + record);

        snapTarget.source = "local" ;
        snapTarget.name = record ;
        snapTarget.projection = "EPSG:2154" ;
        //snapTarget = app.selectedLayer.getLayer();
        app.tools['snapping-agent'].addSnappingTarget  (snapTarget);
        alert ( app.tools['snapping-agent'].active );
                        },
    actionTarget: "map.tbar"

}
```

Nous avons besoin d'identifier la couche active afin de la rendre snapable. Cela se traduit par les instructions : **record = app.selectedLayer.get("name");**

Etudions en detail les types en presence.

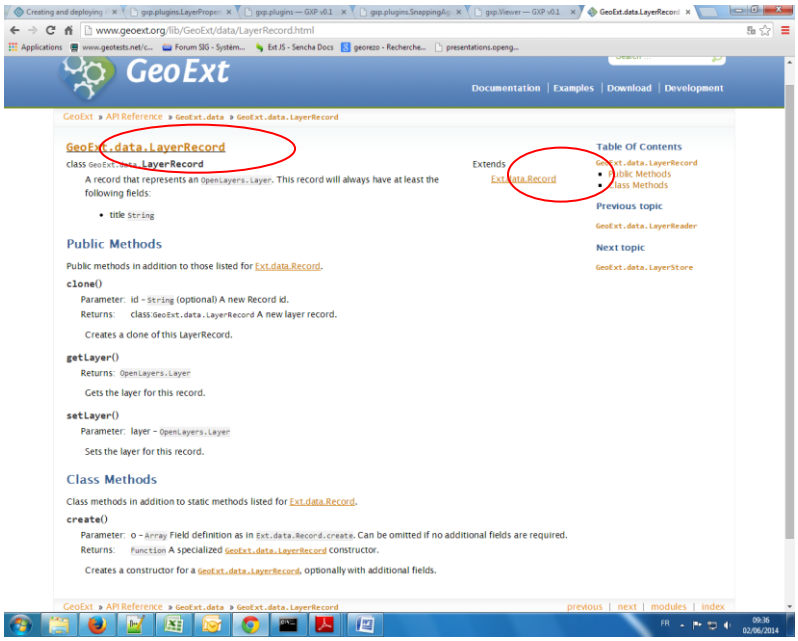
App est de type gxp.Viewer. Voyons que dit la documentation sur gxp.viewer





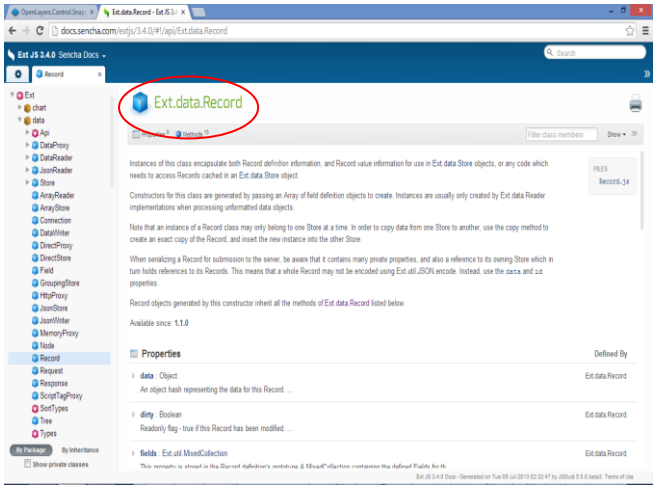
On trouve la méthode **selectedLayer** dans la rubrique Public Properties. Elle permet de ramener la couche active. Celle-ci amène une donnée de type : **GeoExt.data.LayerRecord**

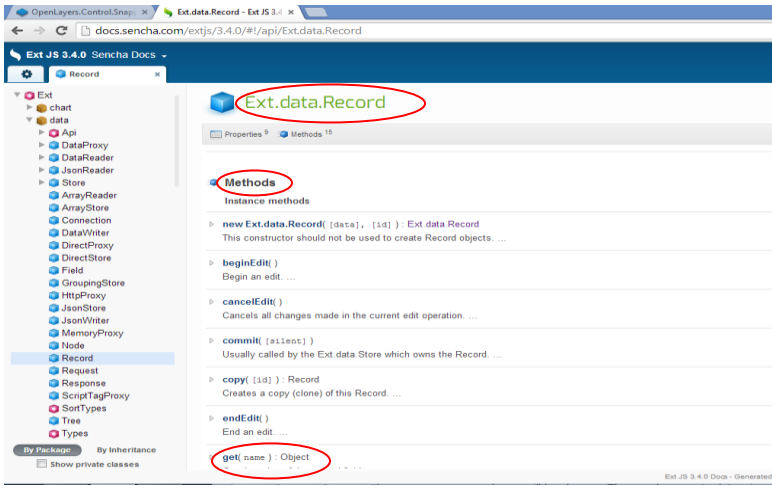
Ce qui nous amène sur la doc **GeoEXT**



On observe que **GeoExt.data.LayerRecord** hérite de **Ext.data.Record**

Allons à présent dans la documentation **d'ExtJs** :





Nous y trouvons la méthode get(name) qui permet de ramener le nom du record. Name étant de type String.

Donc au final, pour trouver la syntaxe de cette instruction :

record = app.selectedLayer.get("name");

Nous avons dû dérouler 3 documentations (gxp, GeoExt, ExtJs).

C'est cette gymnastique fondamentale dans le développement objet et dans cette architecture que j'ai mis un certain temps à maitriser.

Revenons au fil des versions de programme. Nous obtenons avec la 2e version un moyen de rendre toutes les couches snapable. Or nous souhaitons snaper uniquement sur la couche active. Ce qui nous amène à la troisième version.

3E VERSION : RENDRE UNIQUEMENT LA COUCHE ACTIVE SNAPABLE

L'idée à partir de la 2e version de l'application est d'enlever toutes les targets de couches snapable au début. Puis d'ajouter, comme nous l'avons fait, le snap pour la couche active à la fin. On est censé n'obtenir qu'une seule couche snapable : la couche active.

Etudions le code pour ce faire :

Au lieu d'utiliser un bouton pour sélectionner la couche active nous lèvons l'évènement '**layerselectionchange' :** app.on('layerselectionchange', **function**() {….

A l'intérieur de la fonction définit plus haut nous plaçons le code qui suit :

app.tools['snapping-agent'].**clearSnappingTargets** ();

Pour trouver la fonction **clearSnappingTargets** () , il est nécessaire de rentrer dans le code du script ou plugin **SnappingAgent**.js

```
/** public: method[clearSnappingTargets]
 *  Removes all existing snapping targets.  Snapping targets have references
 *  to vector layers that are created in :meth:`addSnappingTarget`.  This
 *  method destroys those layers.
 */

clearSnappingTargets: function() {
    var target;
        for (var i=0, ii=this.snappingTargets.length; i<ii; ++i) {
       target = this.snappingTargets[i];
       if (target.layer) {
          target.layer.destroy();
```

```
        }
    }
    this.snappingTargets.length = 0;//
},
```

Nous remarquons tout d'abord le mot clé **public** qui indique que cette méthode est accessible. Ce n'était pas le cas auparavant. Cette méthode était **private**. Nous sommes intervenus dans le code source du plugin SnappingAgent.js pour modifier cet état.

Le problème est que la méthode **clearSnappingTargets** marche trop bien! Plus aucune couche n'est snapable après.

D'où l'idée de réécrire cette fonction en la remplaçant par une autre **enlever_targets** qui sera de type public.

4E VERSION

```
/** public : method[enlever_targets]
 *  Removes all existing snapping targets.  sauf celle qui est sélectionnée
      */

    enlever_targets :function (record_name){
    var target;
    jj = this.snappingTargets.length ; //nb de target initial
    alert ('nb de target initial  = ' + jj);
    for (var i=0, ii=this.snappingTargets.length; i<ii; ++i) {
     target = this.snappingTargets[i];
     if (target.layer) {
          alert ('                    target.layer.name = ' + target.layer.name) ;
          alert  ('                    record_name = ' + record_name);
               if (record_name != target.name) {
               jj--;
               alert ('   target.layer.name    DESTROYED= ' + target.layer.name) ;
               target.layer.destroy();
                                        }
    }
}}
```

Cette fonction prend en paramètre le nom de la couche active (record_name) et supprime toutes les couches du target de snap autres que la couche active.

Ainsi dans notre application app nous retrouvons :

```
var record = app.selectedLayer.get("name");
app.tools['snapping-agent'].enlever_targets (record);

var snapTarget = Ext.apply({}, snapTarget);
init (snapTarget);
snapTarget.source = "local" ;
snapTarget.name = record ;
snapTarget.projection= "EPSG:2154";
app.tools['snapping-agent'].addSnappingTarget (snapTarget)
```

Cette solution ne fonctionne pas non plus. Plus aucune couche n'est snapable.

Nous en revenons à une solution qui utilise OpenLayers

5E VERSION : OPENLAYERS LE RETOUR

Nous avons vu qu'une application de type gxp dispose des bibliothèques de classes de d'ExtJs mais aussi d'OpenLayers. L'idée est de capitaliser sur ce que nous avons réalisé dans la section précédente sous OpenLayers avec l'adaptation 2.

On modifie donc la levée d'évènement de changement de couche

```
app.on('layerselectionchange', function() {

var record = app.selectedLayer.get("name");
var le_nom_taget_select = app.selectedLayer.get("name");

var la_target  = app.selectedLayer.getLayer();
//alert(' nom de target = ' + la_target.name);
```

```
//alert(' ID de target = ' + la_target.id);

var snap = new OpenLayers.Control.Snapping({layer: la_target, targets: [la_target]}) ;
app.mapPanel.map.addControl (snap);


snap.activate();

        } ,

this);
```

app.mapPanel.map.addControl (snap) est la ligne principale

// MapPanel est rendu public dans le source du viewer !!

// app.map est de type GeoExt.MapPanel , on le voit dans la doc de GXP.viwer

// il contient un objet map de type OpenLayers.Map, on peut donc appliquer sur cet objet la fonction addControl()

Pour des raisons qu'on ignore aucune couche n'est snapable. Nous sommes à la recherché d'une solution alternative.

Nous en sommes arrivés à ce point d'arrêt avant de consacrer plus de temps au présent rapport de stage.

Les pricnipales difficultés rencontrés lors de ces phases de développement informatique ont été :

Une difficulté inhérente au langage Javascript qui n'est pas typé. Il est dès lors difficile de repérer le type d'une variable. L'autocomplétion n'est pas activée dans notre éditeur de sources Notepad++.

Des difficultés à saisir la théorie et la pratique du développement objet. J'ai mis un certain temps à comprendre les documentations en ligne sur les classes des différentes bibliothèques.

Un manque de connaissances sur des connaissances de base en architecture web : notion de proxy…

Nous aborderons deux technologies en annexe qui ont été importantes durant le stage, ainsi que la lettre envoyée aux utilisateurs leur demandant un retour sur l'application existante.

CONCLUSION

Ce stage constitue une expérience riche et stimulante pour moi. L'ambiance à la fois sérieuse et détendue a permis mon intégration rapide en sein de l'équipe d'Ecosphère.

J'ai néanmoins été confronté à plusieurs difficultés: la première est mon manque de préparation pour ce type de travail (un manque d'expérience en développement Orienté Objet, un manque de connaissance en architecture web TCP/IP (proxy, localhost, adresses IP…). En ce domaine, le décalage est important entre les cours dispensés à l'université et le niveau requis pour le stage, très exigeant. La deuxième est liée à l'encadrement au sein de la société, celui-ci est en effet assuré par un seul informaticien. La documentation, rare, souvent lapidaire et exclusivement en anglais a rendu mon auto-formation difficile. La troisième a trait à la mission qui m'a été confiée. Elle était sans doute trop vaste pour que j'en prenne toute la mesure compte tenu de mon expérience en développement dans des langages orientés objet.

Cependant ces quatre mois de stage m'ont permis d'acquérir une solide connaissance des logiciels et d'élargir mes compétences dans le domaine du WebMapping. De ce point de vue, ce rapport de stage marque pour moi une étape. Je me sens en effet nettement plus autonome dans mon travail et me sent désormais capable de produire plus rapidement pour l'entreprise.

Ce point positif au plan technique peut être nuancé par une absence de ma part d'une vision globale et transversale du projet. Cela pourrait s'expliquer par le fait que je suis accaparé à des taches techniques très prenantes. Et par le mode de management. Celui-ci réclame une extrême autonomie notamment dans les PME. Je suis habitué aux entreprises de taille plus importantes où la hiérachie est plus présente.

Mais il reste encore deux mois de stage qui me permettront certainement d'acquérir cette vision d'ensemble. Je me sens en tous cas capable aujourd'hui de réaliser ce type de mission avec succès fort de l'expérience acquise.

BIBLIOGRAPHIE ET SITOGRAPHIE

OpenLayers 2.10 Beginner's Guide Copyright © 2011 Packt Publishing
De Erik Hazzard

Ext JS 3.0 Cookbook Copyright © 2009 Packt Publishing
De Jorge Ramon

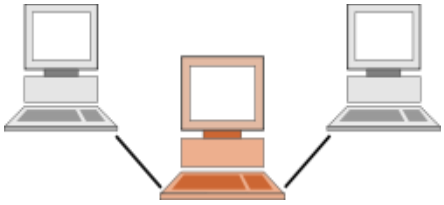
Pour les **schémas** : différentes sources internet

ANNEXES

PROXY

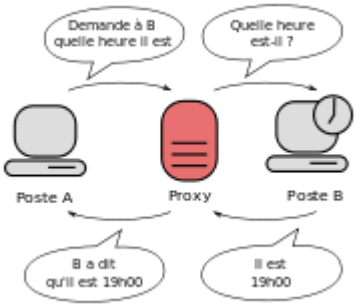
Un serveur proxy (traduction française de «proxy server», appelé aussi «serveur mandataire») est à l'origine une machine faisant fonction d'intermédiaire entre les ordinateurs d'un réseau local (utilisant parfois des protocoles autres que leprotocole TCP/IP) et internet.

La plupart du temps le serveur proxy est utilisé pour le web, il s'agit alors d'un proxy HTTP. Toutefois il peut exister des serveurs proxy pour chaque protocole applicatif (FTP, ...).



FONCTIONNEMENT D'UN PROXY

Le principe de fonctionnement basique d'un serveur proxy est assez simple : il s'agit d'un serveur "mandaté" par une application pour effectuer une requête sur Internet à sa place. Ainsi, lorsqu'un utilisateur se connecte à internet à l'aide d'une application cliente configurée pour utiliser un serveur proxy, celle-ci va se connecter en premier lieu au serveur proxy et lui donner sa requête. Le serveur proxy va alors se connecter au serveur que l'application cliente cherche à joindre et lui transmettre la requête.
Le serveur va ensuite donner sa réponse au proxy, qui va à son tour la transmettre à l'application cliente.



FONCTIONNALITES D'UN SERVEUR PROXY

Désormais, avec l'utilisation de TCP/IP au sein des réseaux locaux, le rôle de relais du serveur proxy est directement assuré par les passerelles et les routeurs. Pour autant, les

serveurs proxy sont toujours d'actualité grâce à un certain nombre d'autres fonctionnalités.

PROXY-CACHE

La plupart des proxys assurent ainsi une fonction de cache (en anglais caching), c'est-à-dire la capacité à garder en mémoire (en "cache") les pages les plus souvent visitées par les utilisateurs du réseau local afin de pouvoir les leur fournir le plus rapidement possible. En effet, en informatique, le terme de "cache" désigne un espace
de stockage temporaire de données (le terme de "tampon" est également parfois utilisé).

Un serveur proxy ayant la possibilité de cacher (néologisme signifiant "mettre en mémoire cache") les informations est généralement appelé "serveur proxy-cache".

Cette fonctionnalité implémentée dans certains serveurs proxy permet d'une part de réduire l'utilisation de la bande passante vers internet ainsi que de réduire le temps d'accès aux documents pour les utilisateurs.

Toutefois, pour mener à bien cette mission, il est nécessaire que le proxy compare régulièrement les données qu'il stocke en mémoire cache avec les données distantes afin de s'assurer que les données en cache sont toujours valides.

FILTRAGE

D'autre part, grâce à l'utilisation d'un proxy, il est possible d'assurer un suivi des connexions (en anglais logging ou tracking) via la constitution de journaux d'activité (logs)
en enregistrant systématiquement les requêtes des utilisateurs lors de leurs demandes de connexion à Internet.

Il est ainsi possible de filtrer les connexions à internet en analysant d'une part les requêtes des clients, d'autre part les réponses des serveurs. Lorsque le filtrage est réalisé en comparant la requête du client à une liste de requêtes autorisées, on parle de liste blanche, lorsqu'il s'agit d'une liste de sites interdits on parle de liste noire. Enfin l'analyse des réponses des serveurs conformément à une liste de critères (mots-clés, ...) est
appelé filtrage de contenu.

AUTHENTIFICATION

Dans la mesure où le proxy est l'intermédiaire indispensable des utilisateurs du réseau interne pour accéder à des ressources externes, il est parfois possible de l'utiliser pour authentifier les utilisateurs, c'est-à-dire de leur demander de s'identifier à l'aide d'un nom d'utilisateur et d'un mot de passe par exemple. Il est ainsi aisé de donner l'accès aux ressources externes aux seules personnes autorisées à le faire et de pouvoir enregistrer dans les fichiers journaux des accès identifiés.

Ce type de mécanisme lorsqu'il est mis en œuvre pose bien évidemment de nombreux problèmes relatifs aux libertés individuelles et aux droits des personnes...

LOCALHOST

En informatique, on travaille souvent en mode client-serveur : une ou plusieurs machines envoient des requêtes à un serveur central1 qui envoie les réponses appropriées. C'est par exemple le cas pour un serveur web ou bien un serveur de bases de données.

Lors du développement, il n'est pas pour autant nécessaire de disposer de plusieurs machines physiques ni même virtuelles : la même machine physique peut parfaitement héberger le serveur et un ou plusieurs clients, exactement dans les mêmes conditions: en communiquant par des ports.

Dans les domaines des réseaux informatiques, localhost (l'hôte local en français) est un nom habituel pour se référer à une interface logique de l'ordinateur local.

Le ou les clients hébergés sur une machine2 utilisent le protocole IP pour communiquer. Il importe peu de savoir où se trouvent physiquement les programmes, les couches basses du protocole se chargeant justement d'en masquer les détails. Le nom localhost est associé à l'adresse IPv6 ::1 et à la plage d'adresses adresse IPv4 127.0.0.0/8 (toutes les adresses IPv4 comprises entre 127.0.0.1 et 127.255.255.255 dont la plus utilisée est 127.0.0.1).

L'interface réseau virtuelle utilisée dans cette situation se nomme l'interface de loopback (abrégée par lo sous Unix).

DANS LA PARATIQUE

Toute machine disposant d'une pile TCP/IP fonctionnelle permet de s'adresser à localhost, même si cette machine n'est reliée à aucun réseau physique.

Il faut bien entendu que soit démarré au préalable le serveur approprié (http, base de données...) sur un port préalablement convenu (souvent 80 ou 8080 pour http, souvent 3306 pour MySQL, etc.), ping3 l'étant en standard s'il n'a pas été préalablement inhibé pour des raisons de sécurité4.

On vérifie typiquement que la pile TCP/IP d'une machine est bien en place en exécutant la commande ping sur localhost.

FICHE ENTRETIEN AVEC UN UTILISATEUR DE L'APPLICATION WEBMAPPING EXISTANTE

Bonjour,

Je suis Thierry, nouveau stagiaire Ecosphère pour 6 mois dans l'équipe Géomatique.

Le but de mon stage est d'améliorer l'utilisation des outils de WebMapping Geoexplorer et GeoNode que vous utilisez depuis un certain temps.

Cette enquête vise à déterminer votre degré de satisfaction à ces outils.

Depuis quand utilisez-vous cet outil ? Faites-vous de la création de carte ou uniquement de la consultation ?

Nous avons un certain nombre de questions sur lesquelles nous avons besoin de votre retour pour avancer au mieux.

Architecture, temps de réponse :
Est-ce que les cartes ou les couches s'affichent rapidement en mode création ou visualisation ? Les temps de réponses sont-ils acceptables ? Lorsque vous sauvegardez votre travail et le réaffichez ultérieurement, avez-vous un affichage fluide ?

Nombre de couches proposées :
Le nombre de couches proposé est-il suffisant ? Avez-vous des suggestions sur les couches à ajouter, de type par exemple « espèces protégées » ou autres ?

Outil de saisie:
Comment faites-vous pour saisir plusieurs Espèces pour un même Enregistrement. La solution proposée vous semble-t-elle satisfaisante ? Le cas échéant que souhaiteriez-vous à la place. Par exemple, un menu déroulant des espèces vous semble-t-il plus ergonomique ?

Ergonomie :
Arrivez-vous à retrouver vos cartes aisément ? Le fait de préfixer la carte par le nom de l'étude vous semble-t-il judicieux ? Qu'est-ce que devrait comporter un outil de recherche des cartes ? Par mot clé, par date de saisie ? Par Etude ? Est-ce qu'une liste déroulante ou répertoire qui classerait les cartes par Etudes ou par Auteur vous semble-t-il opportun ?

Bouton Mesure :

- Il comporte actuellement les lignes et les polygones. Est-ce suffisant ? Souhaitez-vous ajouter d'autres formes géométriques de type Ellipse ou autre ?

- Est vous gêné en mode saisie par l'étiquette qui s'affiche à l'endroit où la souris se déplace ? Cette étiquette est-elle source d'informations ou inutile ?
- En mode modification, on peut créer d'autres points sur un polygone mais pas en supprimer avec un clic droit par exemple. Cette fonctionnalité vous serait-elle utile ?
- En cas de saisie d'une ligne ou d'un polygone complexe, Auriez-vous besoin de revenir en arrière au point antérieur plutôt que d'avoir à tout refaire et revenir par défaut au pont initial ? Un peu à la manière d'un Contrôle Z.

Ajout de nouvelle couche :
Actuellement dans ce cas de figure, la couche est positionnée par défaut au centre du monde. Est-ce gênant ? Ou faudrait-il placer la souris ailleurs le cas échéant ?

Bouton d'impression :
Est-il conforme à vos attentes ? Peut-on passer d'un format à un autre aisément ?

Bouton Interrogation :
L'utilisez-vous ? Est-ce que l'ergonomie et les fonctionnalités vous semblent correctes ?
Utilisez-vous le prédicat LIKE ?

N'hésitez pas à nous manifester toute demande d'amélioration souhaitée.

Thomas et moi-même sommes à votre disposition pour toute réclamation constructive en vue d'améliorer l'outil.

Cordialement,

Thierry Koskas