Etude exploratoire XML/SVG

- Centre d'Études Sur Les Réseaux, Les Transports, L'Urbanisme Et Les Constructions

Étude exploratoire
XML/SVG

Certu

centre d'Études sur les réseaux,
les transports, l'urbanisme
et les constructions publiques
9, rue Juliette Récamier
69456 Lyon Cedex 06
téléphone : 04 72 74 58 00
télécopie : 04 72 74 59 00
www.certu.fr

Avis aux lecteurs

La collection Rapports d'étude du Certu se compose de publications proposant des informations inédites, analysant et explorant de nouveaux champs d'investigation. Cependant l'évolution des idées est susceptible de remettre en cause le contenu de ces rapports.

Le Certu publie aussi les collections :

Dossiers : Ouvrages faisant le point sur un sujet précis assez limité, correspondant soit à une technique nouvelle, soit à un problème nouveau non traité dans la littérature courante. Le sujet de l'ouvrage s'adresse plutôt aux professionnels confirmés. Le Certu s'engage sur le contenu mais la nouveauté ou la difficulté des sujets concernés implique un certain droit à l'erreur.

Références : Cette collection comporte les guides techniques, les ouvrages méthodologiques et les autres ouvrages qui, sur un champ donné assez vaste, présentent de manière pédagogique ce que le professionnel courant doit savoir. Le Certu s'engage sur le contenu.

Débats : Publications recueillant des contributions d'experts d'origines diverses, autour d'un thème spécifique. Les contributions présentées n'engagent que leurs auteurs.

Catalogue des publications disponible sur http://www.certu.fr

NOTICE ANALYTIQUE

Organisme commanditaire :
Certu (département TEC/Pôle géomatique) sur demande de la DRAST

Titre :
Etude exploratoire XML/SVG

Sous-titre :	**Date d'achèvement :** mars 2002	**Langue :** français
Organisme auteur : Société SWORD	**Rédacteurs ou coordonnateurs :** Société SWORD et Certu	**Relecteur assurance qualité :** CHABRIER Denis BALME Jacques

Résumé :
Il s'agissait de mener une étude exploratoire sur les potentialités et limites des technologies XML pour la consultation distante de document composite (textes et cartes).
Une maquette a été réalisée en prenant comme exemple le POS : règlement et document cartographique de zonage. Deux solutions techniques ont été développées : consultation en client léger avec serveur TOMCAT et consultation en client lourd autonome.
Les différents points étudiés sont :

- Une DTD et un schémaXML du règlement,
- La transformation fichier Word du règlement en fichier XML,
- La transformation fichier Mapinfo en fichier SVG,
- La présentation du document au moyen de XSLT pour le règlement et CSS pour le document cartographique,
- L'interactivité de lecture du règlement, de navigation dans la carte et de relation entre les deux composants au moyen de Xlink, Xpath et SVG viewer d'Adobe,
- Les différentes architectures envisageables,
- L'édition papier du document ou d'extrait,
- Les évolutions annoncées : GML, services web (SOAP, WSDL, UDDI),
- Les problèmes de volumétrie que posent les fichiers géographiques,
- Des éléments d'impacts sur les métiers liés à la production et la mise à disposition de données.

Cette étude devrait enrichir les choix du ministère concernant des recommandations d'architecture technique cible pour diffusion et consultation de documents ou données géographiques.

Remarques complémentaires éventuelles (rubrique facultative) :

- En annexe sont livrés les sources des développements.
- Cette étude a été suivie par un groupe projet constitué de représentant du Certu, Cetes, DIREN, DDE, SEGAT et communiquée à DPSM, DGUHC, SETRA.
- La maquette est visible (mais pas en permanence) sur le site du pôle géomatique.

Mots clés : XML, SVG, interactivité, consultation distante, diffusion de données, techniques web, schémaXML		**Diffusion :** Non restrictive	
Nombre de pages :	82 annexe : 52	**Confidentialité :** aucune	**Bibliographie :** Sites web traitant du sujet

Etude exploratoire XML / SVG

IDL_CERTU1/ETU_001 / 1.1

Client	: CERTU		
Entité	: SWORD/IDL		
Projet	: ETUDE EXPLOIRATOIRE XML-SVG		
Id Projet	: IDL_CERTU1		
Date (1)	: 25/03/02		
Etat	: ☒ **à valider**	☐ **validé**	
Diffusion	: ☐ **interne**	☒ **contrôlée**	☐ **libre**

(1) Date d'approbation (cf circuit de validation interne).

CIRCUIT DE VALIDATION

Version	Rédaction			Vérification			Approbation		
	NOM	DATE	VISA	NOM	DATE	VISA	NOM	DATE	VISA
1.0	Renaud BARATE	17/12/01		Yann HERVOUËT	22/02/02		Eric BOUVET	22/02/02	
1.1	Renaud BARATE	25/03/02		Y. HERVOUËT	25/03/02		Eric BOUVET	25/03/02	

HISTORIQUE DES EVOLUTIONS

Version	Objet de la version (citer les fiches de réception de document prises en compte)
1.0	Initialisation du document
1.1	Impact sur les métiers

LISTE DE DIFFUSION

Destinataire	Fonction	Nombre d'exemplaires	Support
Jacques BALME	CERTU – pôle géomatique	1	Electronique
Martine CHATAIN	CERTU – pôle géomatique	1	Electronique
Denis CHABRIER	CERTU – pôle géomatique	1	Electronique

DOCUMENTS REFERENCES

Origine	N°	Titre	Référence	Usage (*)
SWORD	[1]	Annexe de l'étude	IDL_CERTU01/SPF_002/1.0	Document complémentaire
MELT	[2]	Administrer les données localisées, une exigence pour les services		référence

(*) : indiquer le contexte de citation du document : à lire au préalable, documents de référence, documents complémentaires, ...

SOMMAIRE

1. INTRODUCTION 1

1.1 CONTEXTE 1
1.2 PERIMETRE 1
1.3 ATTENTE 2
1.4 ORGANISATION DE L'ETUDE ET DU DOCUMENT 2

2. DESCRIPTION FONCTIONNELLE D'UNE APPLICATION DE CONSULTATION DU P.O.S. 3

2.1 PRESENTATION DE LA MAQUETTE 3
2.2 CAS D'UTILISATION 10
2.2.1 Cas n°1 : Navigation géographique dans la carte du P.O.S. 11
2.2.2 Cas n°2 : Consultation du règlement 12
2.2.3 Cas n°3 : Recherche d'une parcelle 13
2.2.4 Cas n°4 : Génération d'une fiche parcellaire 13
2.2.5 Cas n°5 : Génération du règlement en PDF 14
2.2.6 Cas n°6 : Génération de la carte en PDF 14
2.2.7 Cas n°7 : Consultation du document original du règlement 15
2.3 FONCTIONNALITES DE LA MAQUETTE 15
2.4 LES DONNEES XML ET SVG 16
2.4.1 Document XML : Le règlement du P.O.S. 16
2.4.2 Document SVG : Zones et secteurs géographiques 23

3. TECHNOLOGIES UTILISEES 27

3.1 ARCHITECTURES 27
3.1.1 Application locale 27
3.1.2 Client léger 28
3.1.3 Comparaison et préconisations 28
3.2 REPRESENTATION ET DECOMPOSITION DES FICHIERS XML : LES APIS DOM ET SAX 29
3.2.1 Présentation de DOM 29
3.2.2 Présentation de SAX 29
3.2.3 Comparaison de DOM et SAX 30
3.2.4 JAXP et les différentes implémentations de parseurs 30
3.3 TRANSFORMATION DE DOCUMENTS XML AVEC XSLT ET XPATH 32
3.3.1 Présentation de XSLT 32
3.3.2 Quelques exemples d'utilisation de XSLT 32
3.3.3 Présentation de XPath 34
3.3.4 Sélection et mise en forme d'un article du règlement avec XSLT et XPath 35
3.4 AUTRES LANGAGES DE REQUETE SUR DES DOCUMENTS XML 38
3.4.1 Historique 38
3.4.2 XQL 38
3.4.3 XQuery 39
3.4.4 Avantages et inconvénients 40
3.5 FEUILLES DE STYLE CSS POUR LA REPRESENTATION GRAPHIQUE DES ZONES ET SECTEURS 41
3.5.1 Présentation de CSS 41
3.5.2 Légende automatique avec CSS 41
3.5.3 Comparaison entre XSL et CSS 43
3.6 EDITION AVEC XSL-FO 44
3.6.1 Présentation de XSL-FO 44
3.6.2 Utilisation de XSL-FO pour éditer une fiche parcellaire en format PDF 45
3.7 INTERFACE GRAPHIQUE AVEC SVG ET JAVASCRIPT 49
3.7.1 Généralités 49
3.7.2 Surlignage des contours et effets de transparence sur les zones 49
3.7.3 Réalisation d'un double trait pour le contour des secteurs 50
3.7.4 Gestion d'une carte miniature 51
3.7.5 Zoom sur une parcelle 52
3.8 LIENS AVEC XLINK 54

4. OUTILS DE MANIPULATION DE DONNEES XML ET SVG 55

4.1 EDITION DE DOCUMENTS XML 55
4.1.1 Microsoft XML Notepad 55
4.1.2 eXcelon Stylus Studio 56
4.1.3 XMetal de SoftQuad 56
4.1.4 Arbortext Epic 57
4.1.5 Comparatif et préconisations 58
4.2 VISUALISATION DE DOCUMENTS SVG 60
4.2.1 Adobe SVG Viewer 60
4.2.2 CSIRO SVG Toolkit 60
4.2.3 IBM SVGView 61
4.2.4 Apache Batik SVG Toolkit 61
4.2.5 Préconisations 62
4.3 GENERATION DE SVG 63
4.3.1 Jasc WebDraw 63
4.3.2 W3C Amaya 63
4.3.3 Conversion MapInfo → SVG 64
4.3.4 Conversion DXF → SVG 67

5. ECHANGE DE DONNEES A DISTANCE : SVG OU GML ? 69

5.1 LIMITES DU FORMAT SVG 69
5.2 GML 69
5.3 PROPOSITION D'ARCHITECTURE 70
5.4 OUTILS 71

6. AUTRES LANGAGES ET TECHNOLOGIES DE L'UNIVERS XML 72

6.1 SERVICES WEB (SOAP, WSDL ET UDDI) 72
6.2 AUTRES LANGAGES NORMALISES 72
6.2.1 RDF 72
6.2.2 SMIL 73
6.2.3 MathML 73

7. VOLUMETRIE ET SOLUTIONS 74

7.1 VOLUMETRIE DES DONNEES 74
7.1.1 Généralités 74
7.1.2 Tests de volumétrie 74
7.2 MISE EN ŒUVRE D'UN SERVEUR SPATIAL 75
7.2.1 Mode client / serveur 75
7.2.2 Mode Intranet par imagette 75
7.2.3 Mode Intranet par applet java 75
7.2.4 Mise en œuvre avec SVG 76

8. IMPACTS SUR LES METIERS 78

8.1 SCENARIO FONCTIONNEL RETENU 78
8.2 PRODUCTION DE DONNEES 78
8.2.1 Définition des règles de production d'un lot de donnée 79
8.2.2 Réception d'un lot de donnés 79
8.2.3 Sélection des données nouvelles à intégrer 79
8.2.4 Mise à jour du catalogue de données 80
8.3 MISE A DISPOSITION DES DONNEES 80
8.3.1 Echange de données 80
8.3.2 Consultation de données 81

- o O o -

1. INTRODUCTION

1.1 CONTEXTE

Le Centre d'Etudes sur les Réseaux, les Transports, l'Urbanisme et les Constructions Publiques (CERTU) conduit des études pour les services de l'Etat et les Collectivités Locales.

Au sein du département « Technologies », le pôle géomatique a une activité nationale pour la mise en œuvre des mesures d'accompagnement nécessaires au déploiement réussi des Systèmes d'Information Géographique dans les services du Ministère.

Dans le cadre du Programme d'Action Gouvernemental pour la Société de l'Information (PAGSI) visant à la modernisation des services administratifs et l'amélioration des données publiques, le CERTU souhaite étudier les possibilités offertes par XML / SVG pour structurer et échanger l'information.

Afin d'illustrer les résultats de l'étude, les Plans d'Occupation des Sols constitués de document de présentation, de documents cartographiques et de règlement sont retenus comme exemple.

Cette étude s'inscrit dans la prolongation de la démarche NTIC ADS engagée par la Direction Générale de l'Urbanisme, de l'Habitat et de la Construction.

1.2 PERIMETRE

Cette étude, a pour objectif de déterminer l'apport des technologies XML (eXtensible Markup Language) et SVG (Scalable Vector Graphics) pour la consultation et l'échange de données géographiques et textuelles. C'est une étude exploratoire, qui couvre un grand nombre de sujets et présente un panorama des standards et des logiciels existants autour de l'univers de XML.

Pour illustrer concrètement les possibilités offertes par XML et SVG, nous nous basons sur le document de Plan d'Occupation des Sols (POS) d'une commune. Une maquette a été réalisée pour rendre ce document consultable par l'intermédiaire du World Wide Web (WWW).

Les interrogations du CERTU sont multiples :

- Les technologies XML / SVG sont-elles adaptées pour structurer de l'information et des documents de type POS / PLU ?
- Ces technologies sont-elles suffisantes ou bien faut-il utiliser des normalisations complémentaires tel que GML ?
- Quels sont les apports de ces technologies pour les échanges de données et pour leur exploitation ?
- Quels sont les mécanismes existants pour manipuler l'information ainsi structurée ? Quelles sont les possibilités de navigation, de représentation, d'impression, de requête, de tri, … ?
- Comment constituer les informations / les documents aux formats XML / SVG à partir des sources de données existantes ?
- Quelles sont les contraintes et les limites d'utilisation de ces technologies ?
- Quels sont les impacts sur les processus et les méthodes de création, de mise à jour et de diffusion de l'information ?

1.3 ATTENTE

Les attentes du CERTU pour cette étude sont les suivantes :

- Vérifier l'aptitude de XML / SVG à répondre aux attentes suivantes :
  - capacité à lier les différents types d'informations (alphanumériques, cartographiques et textuelles),
  - capacité à structurer l'information et à constituer une structure partageable entre les différents acteurs,
  - capacité à échanger des documents sans contrainte informatique,
  - capacité à faciliter la mise à jour et l'utilisation des documents,
  - capacité à mettre à disposition les documents sous différentes formes,
- Mesurer les apports et les limites de ces technologies,
- Identifier les impacts sur les processus de fabrication et de diffusion des documents.

1.4 ORGANISATION DE L'ETUDE ET DU DOCUMENT

Pour atteindre les objectifs du CERTU, SWORD propose d'adopter sa démarche d'étude.

Cette démarche est organisée en 4 phases. Elle respecte le découpage de l'étude décrit dans le cahier des charges :

- Phase 1 : Initialisation de la mission,
- Phase2 : Conception, réalisation, tests d'utilisation d'XML / SVG,
- Phase 3 : Qualification des technologies et bilan technique,
- Phase 4 : Etude d'impacts sur les processus de création, de modification et de diffusion de l'information.

Le document s'organise en séparant les aspects fonctionnels des points plus techniques. Il se découpe ainsi :

- Présentation fonctionnelle de la maquette et des données
- Description et argumentation sur les technologies utilisées
- Outils existants pour éditer et convertir ces données dans d'autres formats
- Comparaison entre SVG et GML pour l'échange de données entre SIG
- Présentation des technologies qui gravitent autour du standard XML et qui n'ont pas été illustrées par la maquette.
- Conclusions

2. DESCRIPTION FONCTIONNELLE D'UNE APPLICATION DE CONSULTATION DU P.O.S.

2.1 PRESENTATION DE LA MAQUETTE

Ce chapitre présente la maquette réalisée pour illustrer l'utilisation des technologies XML et SVG. Celle-ci se base sur un exemple de Plan d'Occupation des Sols (P.O.S.). Le but principal est d'afficher la carte de ce P.O.S. grâce à un document SVG et de lier ces informations géographiques aux informations règlementaires. Celles-ci concernent les différentes autorisations et restrictions qui s'appliquent dans chaque zone et sont contenues dans un fichier XML. On souhaite également afficher les parcelles pour pouvoir déterminer dans quelle(s) zone(s) se trouve une parcelle en particulier.

Quelques captures d'écran devraient permettre de mieux appréhender le fonctionnement de cette maquette.

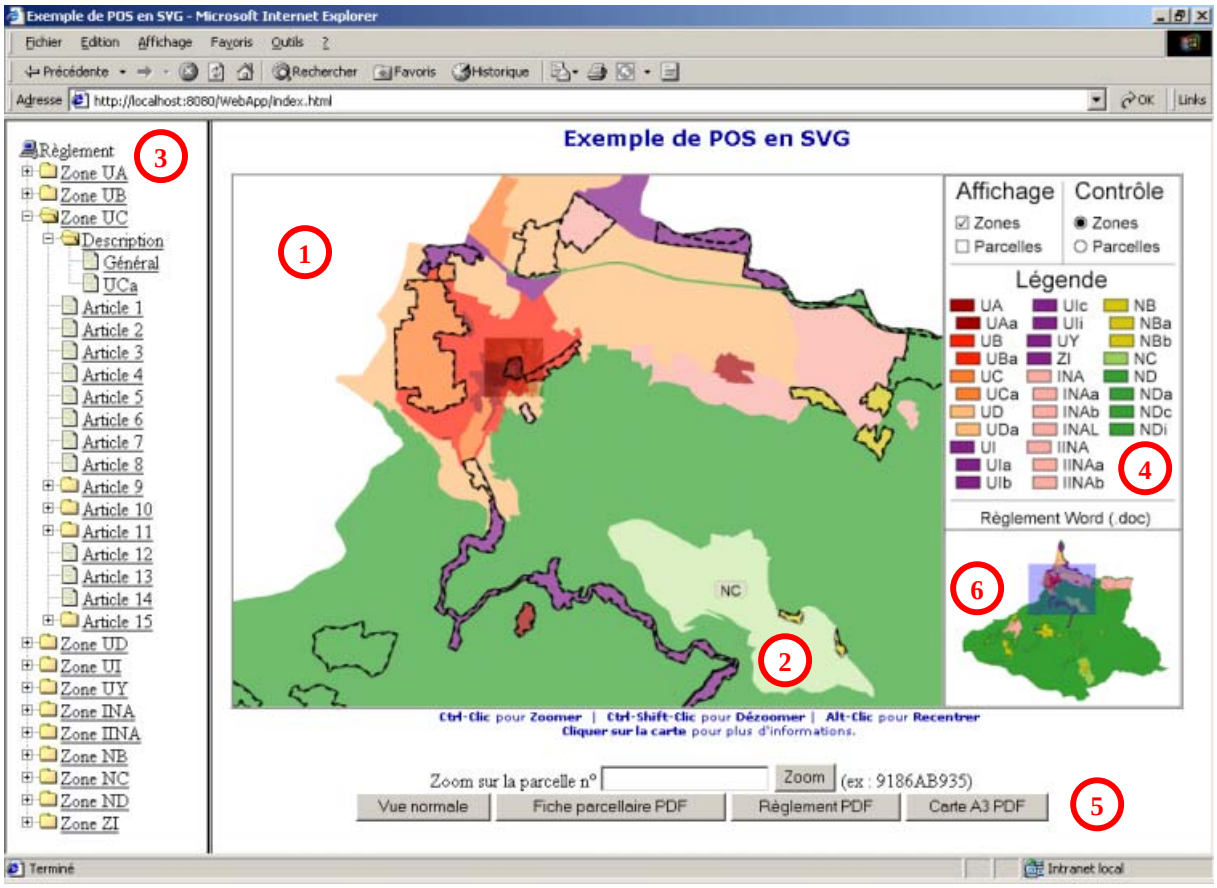


Figure 1 : Fenêtre principale de la maquette

Cette fenêtre principale se décompose ainsi :

1. Carte du P.O.S.,
2. Zone sélectionnée : Elle apparaît dans une couleur plus pâle et un message indiquant le type de zone s'affiche (ici NC),
3. Arborescence pour la navigation dans le règlement,
4. Légende, qui permet également d'afficher les zones et/ou les parcelles, de déterminer sur quelle couche s'effectuent les contrôles (clics de souris), d'afficher tous les secteurs ou toutes les zones d'un certain type ou d'ouvrir une fenêtre contenant le règlement entier au format Word (dans le seul but d'illustrer l'utilisation de XLink),

5. Sélection d'une parcelle. On peut localiser une parcelle en indiquant son numéro. Les autres boutons servent respectivement à revenir à la vue originale, à générer un document PDF contenant une vue d'une parcelle et un extrait du règlement correspondant à une zone et un secteur donnés, à générer un document PDF contenant le règlement entier et à générer un PDF au format A3 avec la carte du P.O.S. et la légende,
6. Carte miniature pour la navigation dans la carte

La localisation de parcelle fonctionne ainsi : après avoir saisi le numéro de la parcelle, l'utilisateur clique sur le bouton « Zoom ». La vue est alors centrée sur la parcelle sélectionnée. Elle est entourée de bleu pour être repérée plus facilement. Pour revenir à la vue générale, l'utilisateur doit simplement cliquer sur le bouton « Vue normale ».

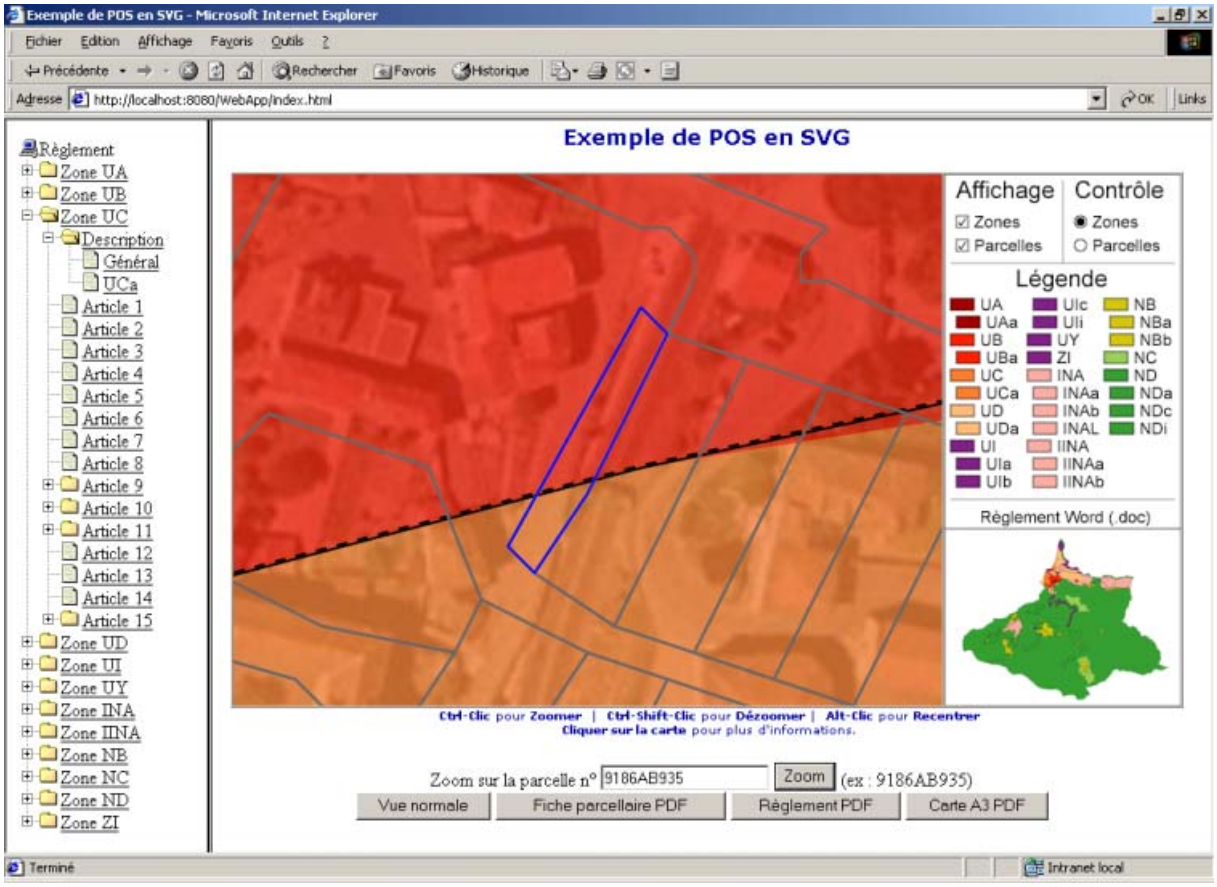


Figure 2 : Localisation d'une parcelle

Lorsque l'on clique sur un lien dans l'arborescence ou sur une zone de la carte, l'application ouvre une nouvelle fenêtre pour présenter les informations sur ce type de zone (ou de secteur). Cette fenêtre se présente ainsi :

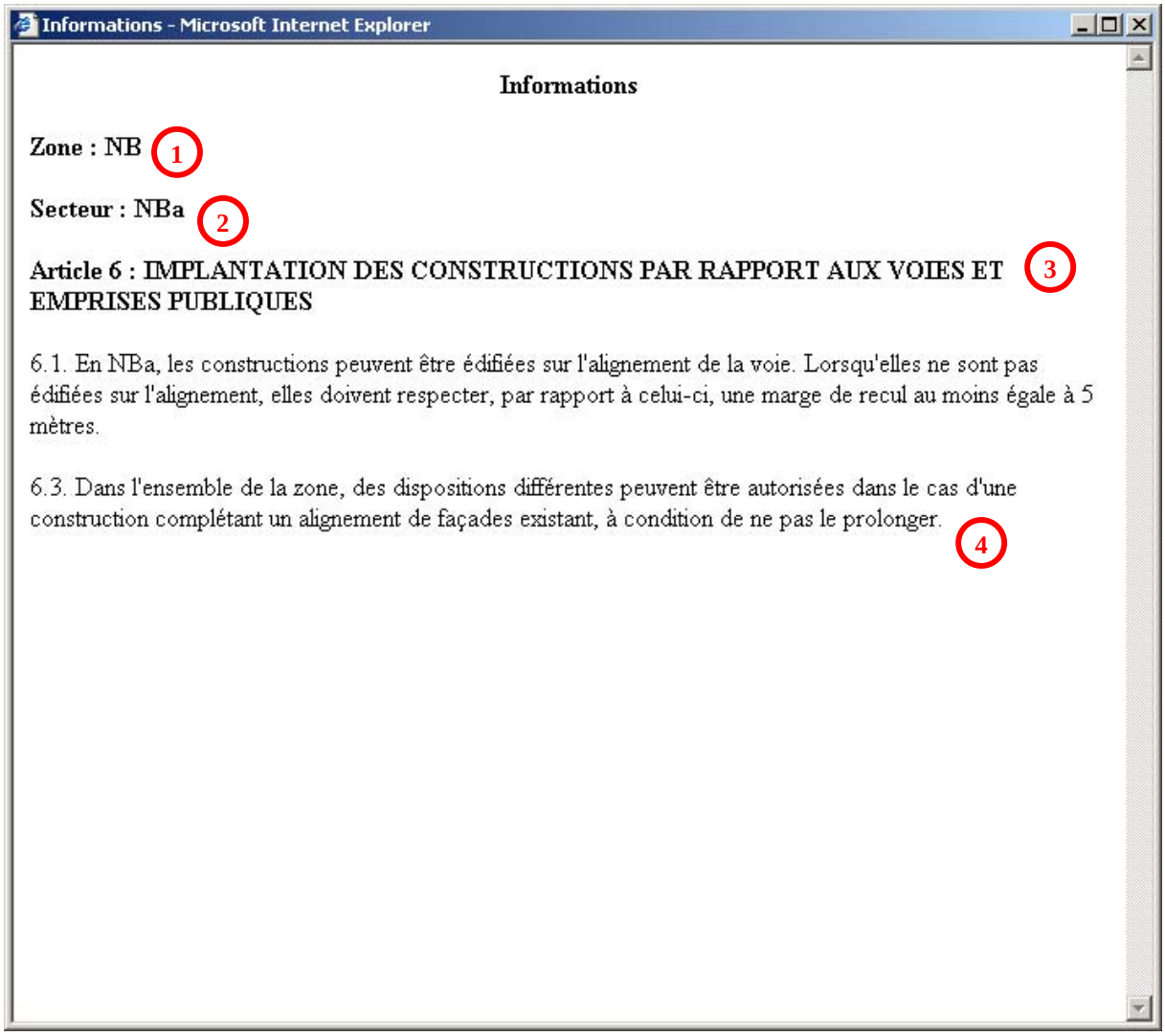


Figure 3 : La fenêtre d'information

Les informations présentées dans cette fenêtre sont les suivantes :

1. Type de zone,
2. Type de secteur,
3. Numéro et nom de l'article. Lorsque l'utilisateur demande des informations en cliquant sur la carte, c'est toujours la description générale de la zone ou du secteur qui est affichée,
4. Contenu de l'article correspondant à la zone et au secteur.

La génération d'une fiche parcellaire au format PDF se fait en cliquant sur le bouton « Fiche parcellaire PDF ». Une fenêtre s'ouvre alors pour choisir la parcelle, la zone et le secteur pour lesquels on souhaite éditer la fiche.

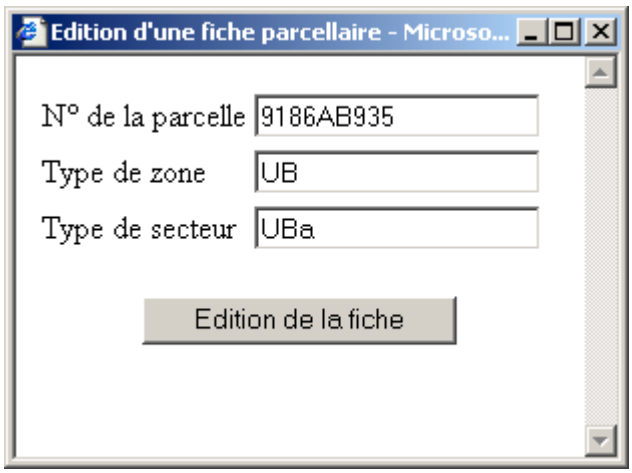


Figure 4 : Fenêtre de sélection de la parcelle, de la zone et du secteur

Il suffit de cliquer ensuite sur le bouton « Edition de la fiche » pour l'obtenir. En voici un exemple :

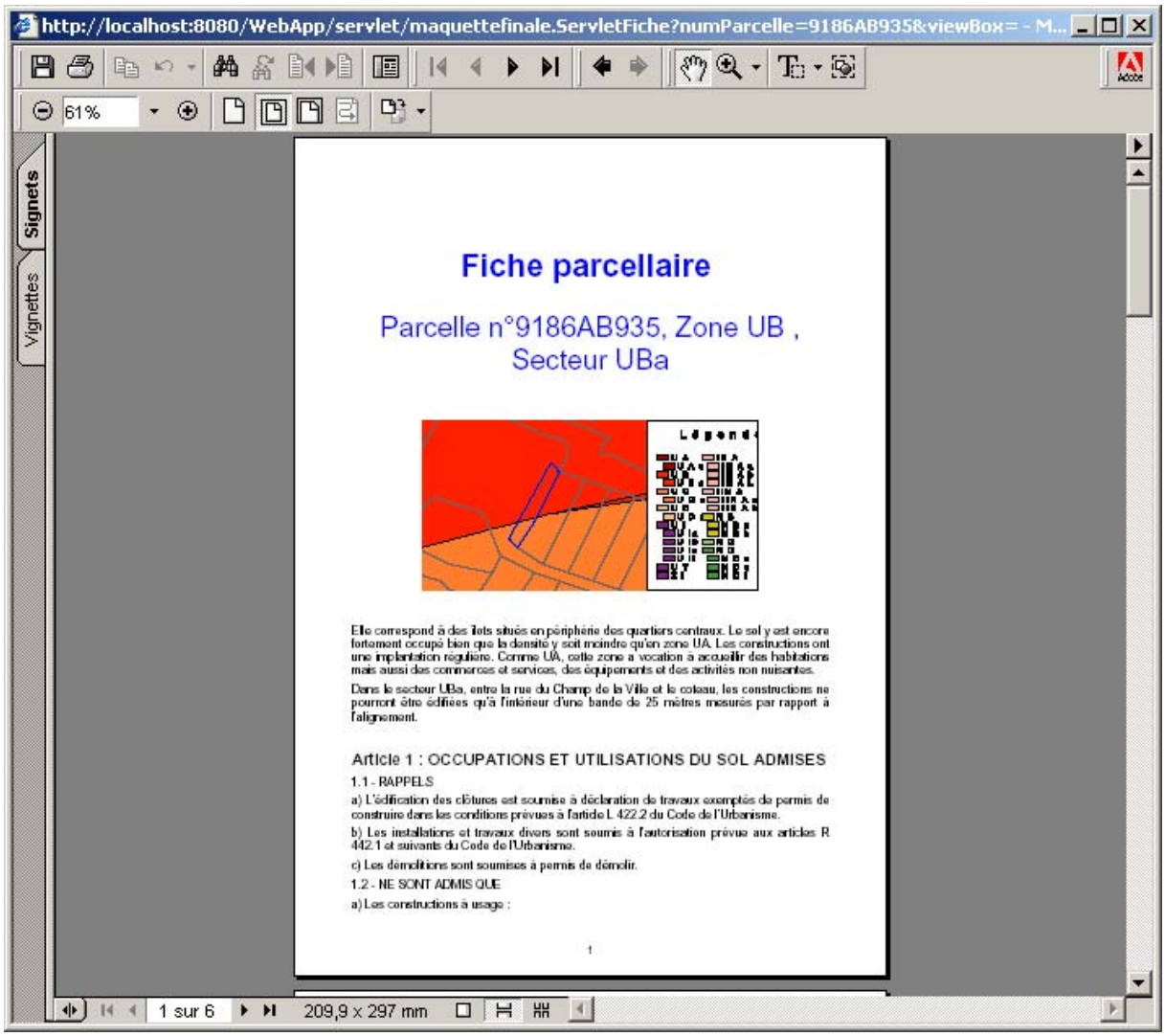


Figure 5 : Première page d'une fiche parcellaire

Pour obtenir le règlement entier au format PDF, il suffit de cliquer sur le bouton « Règlement PDF ».

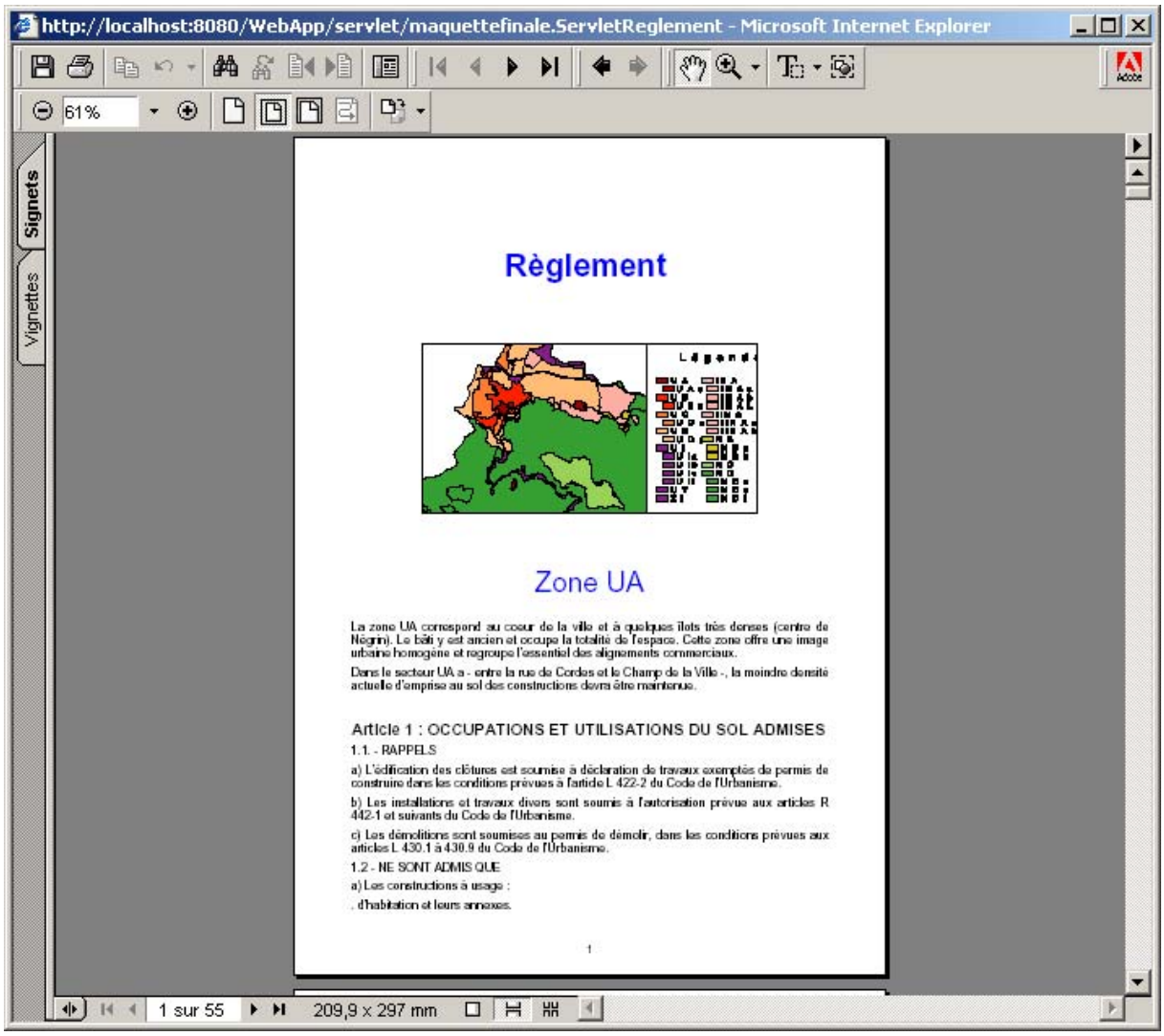


Figure 6 : Première page du règlement en PDF

Pour obtenir une carte au format A3 du P.O.S. en PDF, il faut cliquer sur le bouton « Carte A3 PDF ».

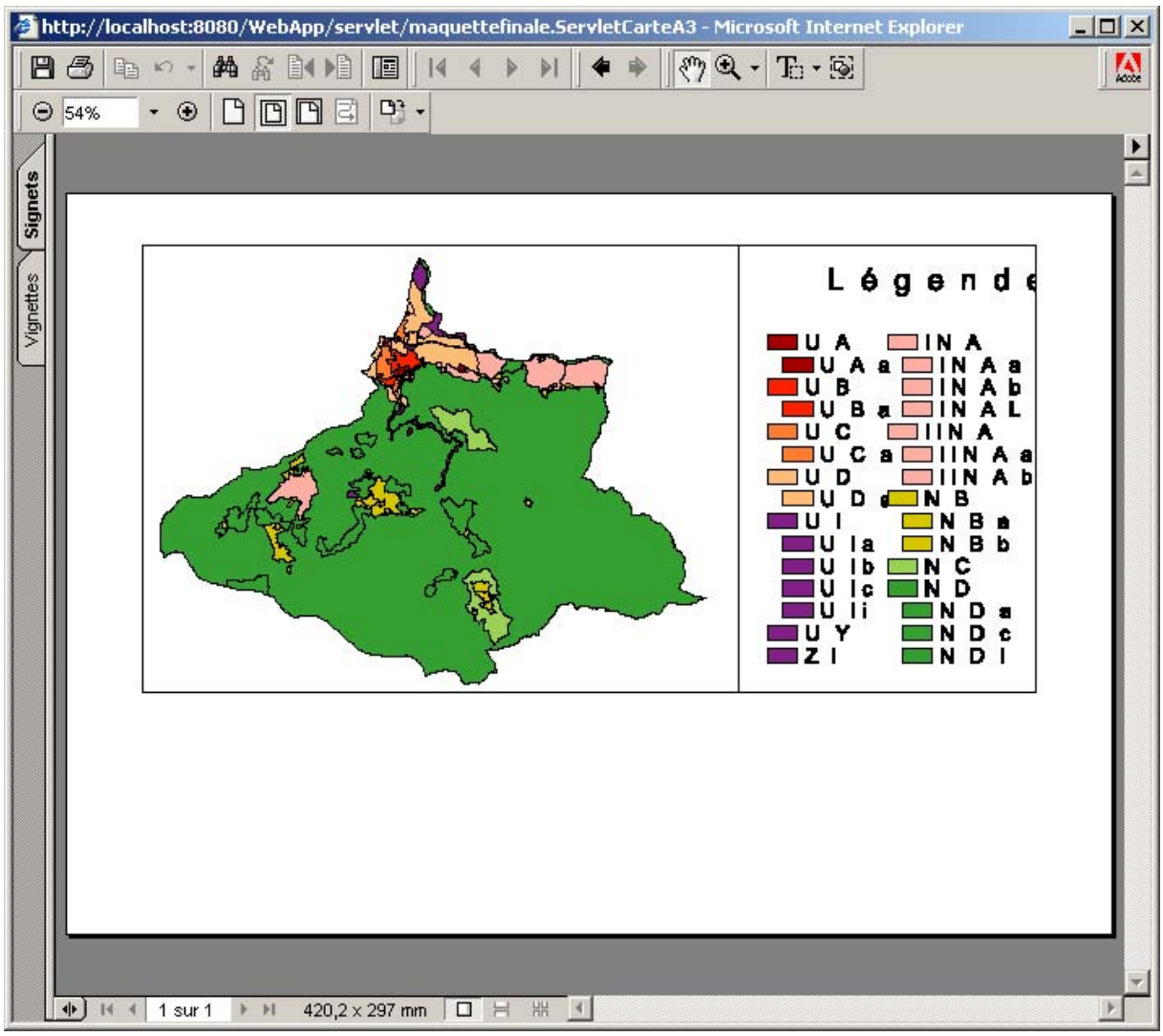


Figure 7 : Carte du P.O.S. en PDF au format A3

Enfin, on peut ouvrir le document Word du règlement en cliquant sur le lien « Règlement Word (.doc) » dans la légende.

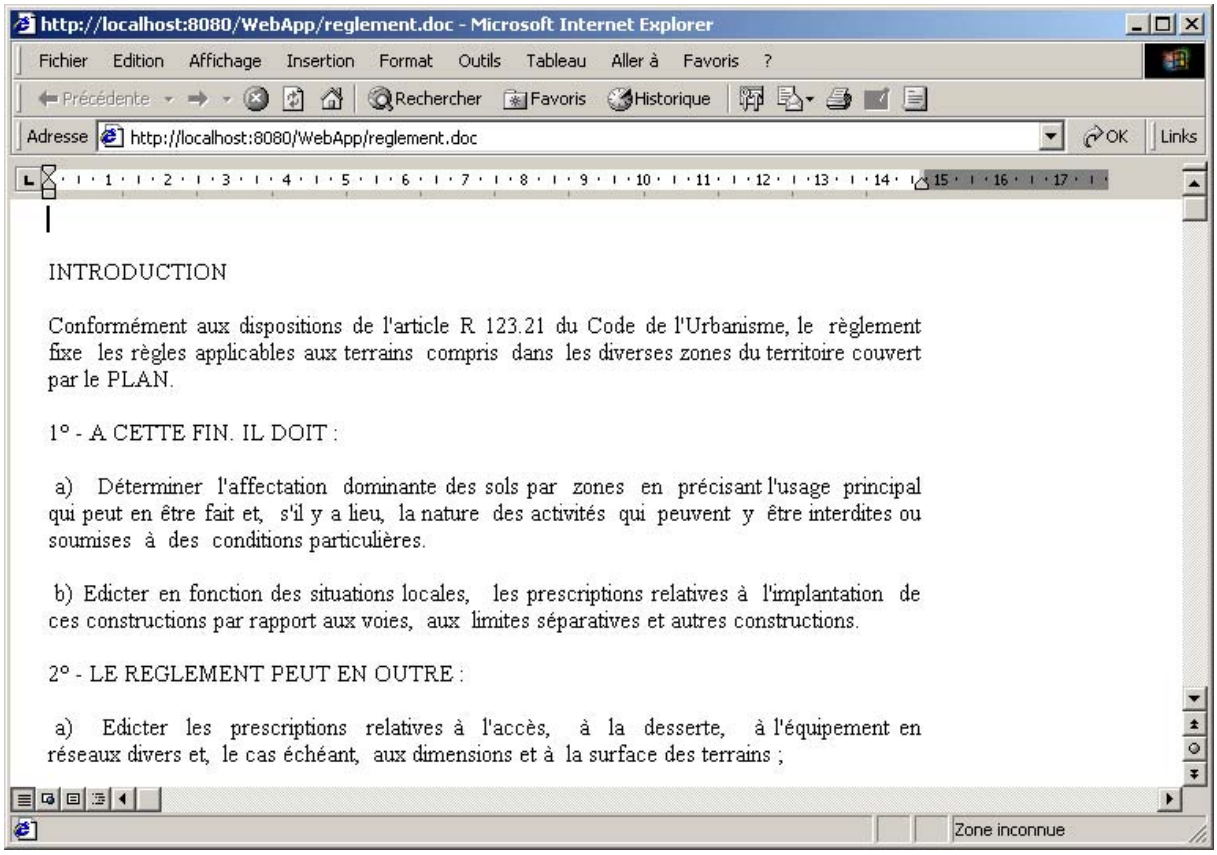


Figure 8 : Document Word original du règlement

2.2 CAS D'UTILISATION

Cette section décrit les différents cas dans lesquels on souhaite consulter le P.O.S. On a décidé de ne pas faire de distinction entre les types d'utilisateurs, car on ne sait pas à l'heure actuelle si l'utilisation qui sera faite du P.O.S. sera réellement différente pour un élu, un particulier ou un professionnel.

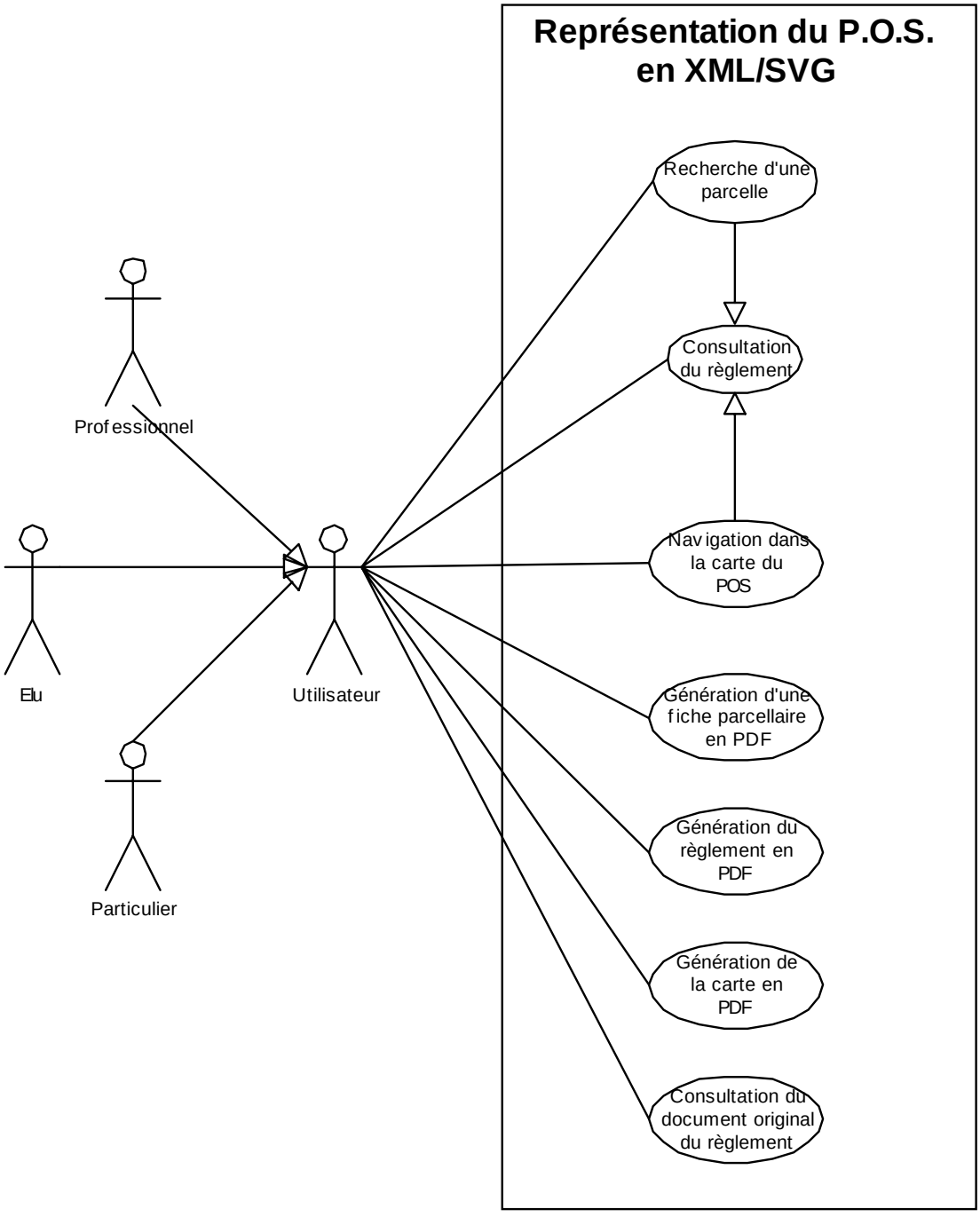


Figure 9 : Les différents cas d'utilisation

2.2.1 Cas n°1 : Navigation géographique dans la carte du P.O.S.

Dans ce cas, l'utilisateur souhaite simplement consulter la carte du P.O.S. pour identifier les différentes zones et leurs secteurs. Il veut également obtenir des informations sur ceux-ci. Le but est de s'informer sur le plan établi par la commune.

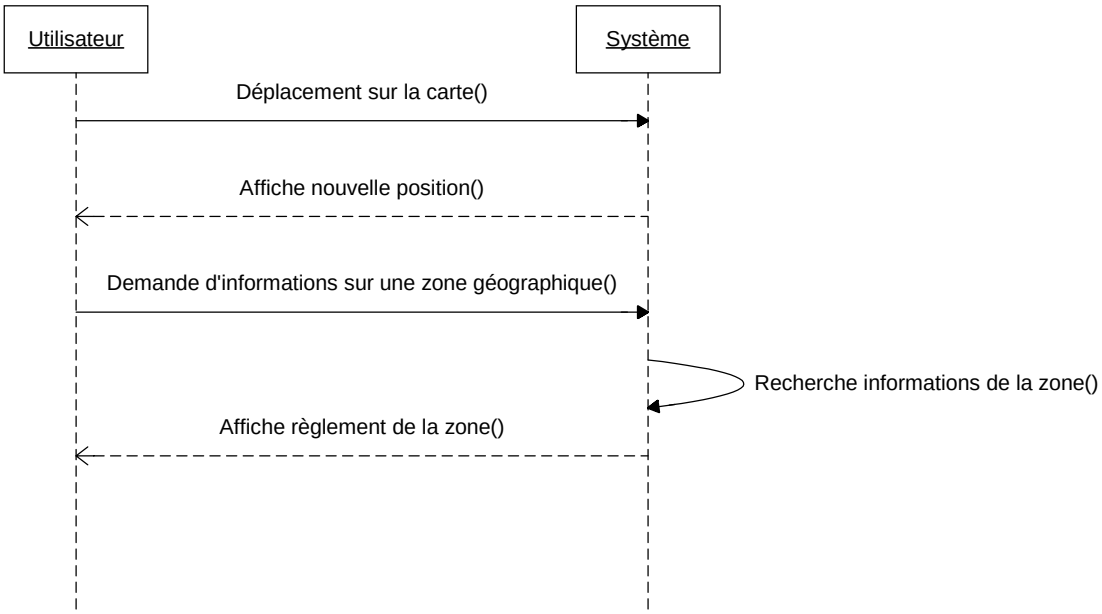


Figure 10 : Séquence d'opérations correspondant à la localisation d'une zone géographique et l'affichage de son règlement

2.2.2 Cas n°2 : Consultation du règlement

L'utilisateur cherche à connaître le règlement d'un type de zone ou de secteur. La navigation ne se fait plus géographiquement mais par type de zone et type de secteur.

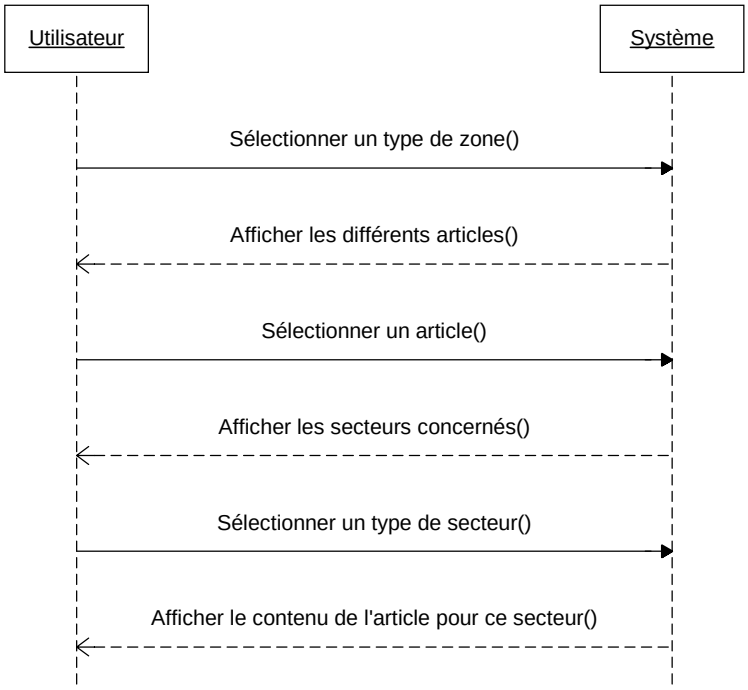


Figure 11 : Séquence d'opérations pour visualiser le règlement pour un type de zone et/ou un type de secteur

2.2.3 Cas n°3 : Recherche d'une parcelle

Dans ce cas, l'utilisateur désire localiser une parcelle dont il connaît le numéro, et éventuellement consulter le règlement de la zone sur laquelle elle se situe.

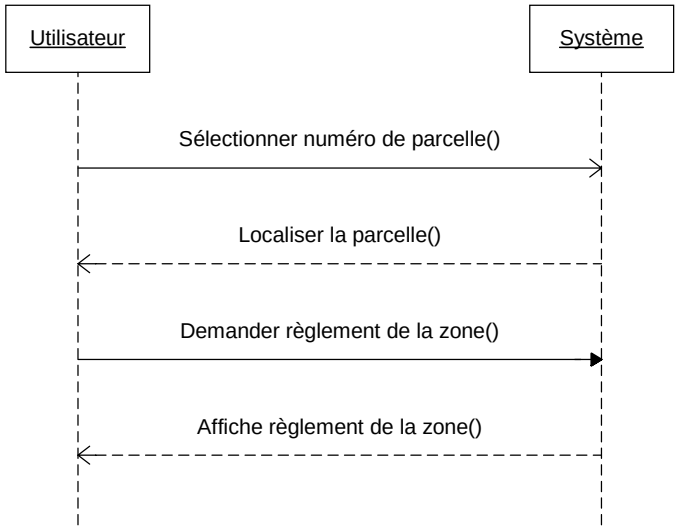


Figure 12 : Séquence d'opérations correspondant à la localisation d'une parcelle et à l'affichage du règlement de la zone correspondante

2.2.4 Cas n°4 : Génération d'une fiche parcellaire

Dans ce cas, l'utilisateur souhaite générer une fiche parcellaire au format PDF pour une parcelle, une zone et éventuellement un secteur donnés.

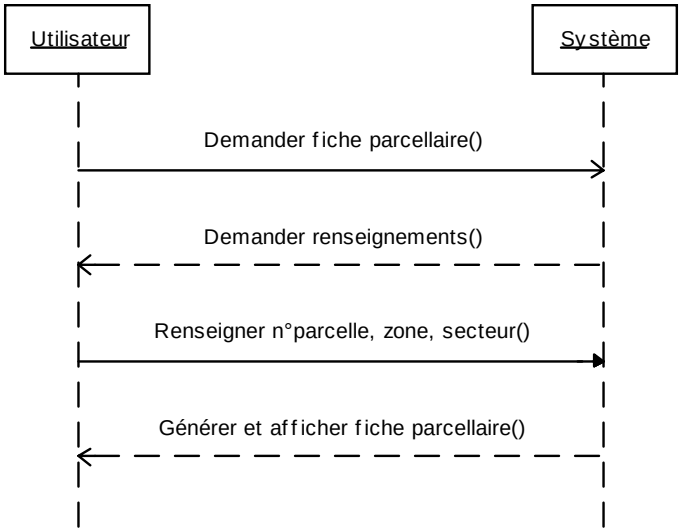


Figure 13 : Séquence d'opérations correspondant à la génération d'une fiche parcellaire

2.2.5 Cas n°5 : Génération du règlement en PDF

Dans ce cas, l'utilisateur désire générer l'ensemble du règlement au format PDF.

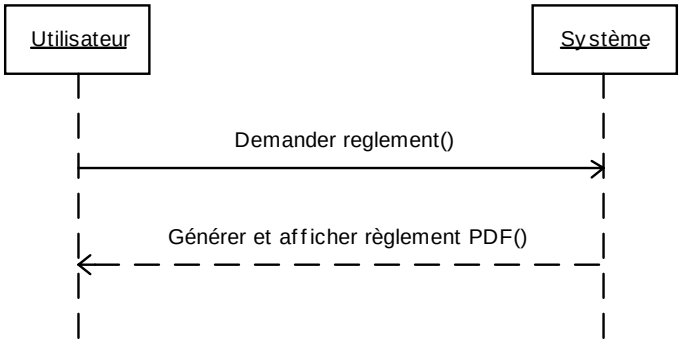


Figure 14 : Séquence d'opération correspondant à la génération du règlement au format PDF

2.2.6 Cas n°6 : Génération de la carte en PDF

Dans ce cas, l'utilisateur désire générer une carte du P.O.S. au format A3 en PDF.

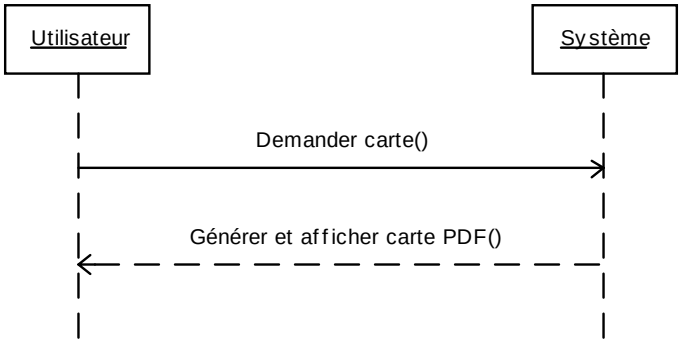


Figure 15 : Séquence d'opérations correspondant à la génération de la carte du P.O.S. au format A3

2.2.7 Cas n°7 : Consultation du document original du règlement

Dans ce cas, l'utilisateur désire consulter le document original du règlement au format Word (.doc).

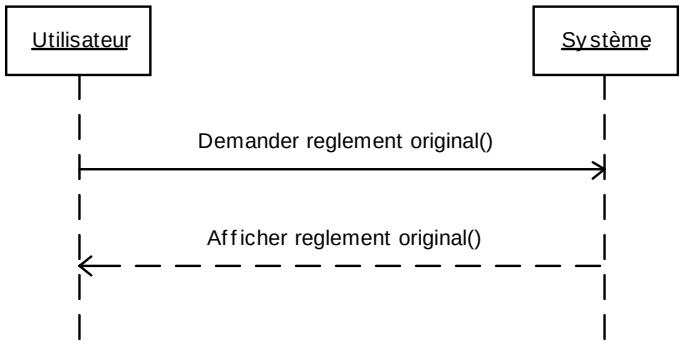


Figure 16 : Séquence d'opérations permettant de consulter le document original du règlement

2.3 FONCTIONNALITES DE LA MAQUETTE

Le rôle de cette maquette est d'illustrer les possibilités d'utilisation de XML, SVG et des technologies associées pour consulter sur le Web le P.O.S. d'une commune. En particulier, elle doit répondre aux attentes des utilisateurs pour les aspects suivants :

- **Carte du P.O.S.**
  - Navigation et zoom sur l'ensemble de la carte du P.O.S.,
  - Possibilité d'afficher les parcelles et/ou les zones et secteurs,
  - Message d'information (« Tooltip ») lors du survol d'une zone ou d'un secteur par le pointeur de souris,
  - Surlignage des zones ou secteurs survolés par le pointeur de souris,
  - Affichage d'une légende permettant de sélectionner un type de zone ou de secteur,
  - Affichage d'une carte miniature pour naviguer plus rapidement,
  - Affichage du règlement de la zone ou du secteur correspondant lors d'un clic de souris,
  - Localisation d'une parcelle par son numéro.
- **Règlement**
  - Arborescence pour naviguer dans le règlement (Zones / Articles / Secteurs),
  - Affichage du règlement par article correspondant à la zone et au secteur sélectionnés,
  - Consultation du document original.
- **Edition**
  - Edition d'une fiche parcellaire avec une vue de la carte et un extrait du règlement,
  - Edition du règlement entier,
  - Edition d'une carte au format A3.

Toutes ces fonctionnalités devront être réalisées en utilisant autant que possible les technologies associées à XML pour les illustrer (XSLT, XLink, XPath, XSL-FO, …).

2.4 LES DONNEES XML ET SVG

Les données concernant le Plan d'Occupation des Sols sont regroupées dans deux documents :

- Un document au format SVG qui contient les données géographiques (zones, secteurs et parcelles).
- Un document au format XML qui contient les informations du règlement.

Chaque partie du règlement correspond à un type de zone ou de secteur. On peut ainsi recouper facilement les informations géographiques et les informations règlementaires. Examinons tout d'abord le document XML contenant le règlement.

2.4.1 Document XML : Le règlement du P.O.S.

2.4.1.1 Présentation de XML

XML (eXtensible Markup Language) a été créé et standardisé par le W3C (World Wide Web Consortium) pour représenter tout type de document de manière structurée. Il est issu de SGML (Standard Generalized Markup Language), complet mais trop complexe pour que son utilisation puisse être généralisée sur le Web.

XML est un méta-langage, c'est à dire qu'il permet de décrire un langage spécifique à chaque application. Même si sa première raison d'être est la consultation et l'échange de documents par le Web, son utilisation est appropriée dans tous les cas qui nécessitent des documents structurés sous forme électronique. Son principal atout est sa simplicité. Pour s'en convaincre, voici un extrait de document XML décrivant une bibliothèque :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bibliothèque>
      <livre titre = 'Le rouge et le noir'>
            <auteur>Stendhal</auteur>
            <année>1830</année>
      </livre>
      <livre titre = 'Les misérables'>
            <auteur>Victor Hugo</auteur>
            <année>1862</année>
      </livre>
      <livre titre = 'La nouvelle Héloïse'>
            <auteur>Jean-Jacques Rousseau</auteur>
            <année>1761</année>
      </livre>
</bibliothèque>
```

La première ligne est la déclaration XML, qui indique simplement qu'il s'agit d'un document XML 1.0, qui utilise le jeu de caractères « ISO-8859-1 » (qui regroupe les caractères des langues latines). Elle est facultative mais il est fortement conseillé de l'utiliser. On la retrouve donc dans la plupart des documents XML.

Le document est ensuite structuré grâce à des balises (encadrées par les caractères '<' et '>'). Celles-ci définissent les éléments du document (par exemple un livre), et leur hiérarchie. Ainsi, toutes les informations contenues entre la balise ouvrante '<livre>' et la balise fermante '</livre>' correspondent au même livre. Chaque élément peut contenir d'autres éléments, et être décrit par des attributs (le titre du livre dans l'exemple précédent).

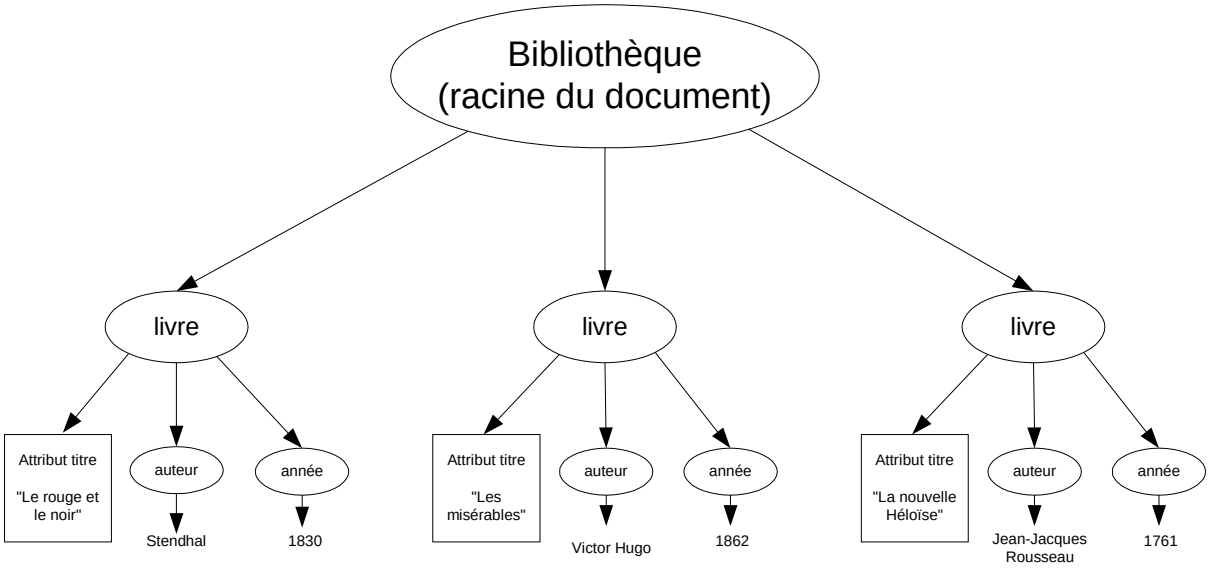


Figure 17 : Hiérarchie d'un document XML

On peut ainsi définir grâce à XML un langage de description de bibliothèques, et la grammaire correspondante. Celle-ci indiquera par exemple qu'une bibliothèque contient plusieurs livres, que chacun a un titre, une année de publication et un ou plusieurs auteurs. Cette grammaire peut être formalisée par une DTD ou un schéma XML. Ces deux termes seront détaillés plus tard avec la structure du règlement du P.O.S.

2.4.1.2 Les recommandations du W3C

Le W3C est au centre de toutes les technologies se rapprochant de XML. Il est composé de différents groupes de travail qui normalisent les spécifications des différents langages (XML, XSLT, SVG, etc.). Le cycle de normalisation d'une technologie se déroule en suivant les étapes suivantes :

- **Working Draft** : C'est le premier élément produit par un groupe de travail (Working Group).
- **Last Call Working Draft** : C'est un rapport du groupe de travail qui est publié lorsque celui-ci pense que les spécifications sont satisfaisantes. Le but est d'obtenir des critiques d'autres groupes ou membres du W3C ou du public.
- **Candidate Recommendation** : Ce document est publié pour pouvoir prendre en compte les premières expériences d'implémentation des spécifications.
- **Proposed Recommendation** : A ce stade, les spécifications sont estimées satisfaisantes au vu des besoins et des implémentations réalisées.
- **W3C Recommendation** : C'est le stade final du document, issu d'un consensus entre tous les acteurs impliqués dans la technologie. Cela signifie que les spécifications sont stables et peuvent être utilisées sans risque de modification ultérieure.

Les retours à l'étape précédente sont souvent nombreux avant qu'une technologie ne parvienne au stade de recommandation du W3C. L'avantage est de pouvoir garantir que la recommandation finale est stable est qu'elle sera normalement respectée par toutes les implémentations. Nous préciserons donc dans ce document les technologies qui ont atteint le stade de la recommandation du W3C et l'adresse où l'on peut trouver ce document. Pour le langage XML lui-même, la recommandation est disponible à l'adresse suivante :

http://www.w3.org/TR/REC-xml : Recommandation pour XML 1.0 (6 Octobre 2000)

2.4.1.3 Présentation du règlement

Le règlement du P.O.S. est un document très structuré, ce qui rend l'utilisation de XML particulièrement adaptée pour le représenter. Il est découpé en plusieurs parties, chaque partie correspondant à un type de zone présent sur le territoire de la commune (par exemple « UA » pour les zones d'habitation dense). Une partie contient une description générale de ce type de zone, et quinze articles. Chaque article correspond à un thème particulier (par exemple l'article 4 traite toujours de la desserte par les réseaux). Enfin, un article est découpé en paragraphes, qui peuvent concerner l'ensemble de la zone ou seulement un ou plusieurs secteur(s) particulier(s).

Grâce à XML, on peut découper ce document de manière hiérarchique et isoler chaque élément sémantique. Voici un extrait de règlement, décrit grâce au langage XML que nous avons formalisé pour cela :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<reglement>
   <zone type="ND">
      <description>
         <paragraphe>
            <texte>La zone ND couvre une large partie du territoire communal, puisqu'elle
            protège les espaces naturels de qualité dont toute urbanisation doit être écartée.
            Cette protection ne peut être supprimée que par une révision du P.O.S.</texte>
         </paragraphe>
         <paragraphe>
            <secteur type = "NDa"/>
            <texte>Le secteur NDa peut recevoir des constructions liées à l'activité agricole
            (y compris logements).</texte>
         </paragraphe>
         ...
      </description>

      <article numero="1" titre="OCCUPATIONS ET UTILISATIONS DU SOL ADMISES">
         ...
      </article>
      <article numero="2" titre="OCCUPATIONS ET UTILISATIONS DU SOL INTERDITES">
         <paragraphe>
            <texte>a) Les lotissements de toute nature.</texte>
         </paragraphe>
         ...
         <paragraphe>
            <secteur type="ND"/>
            <secteur type="NDa"/>
            <secteur type="NDi"/>
            <texte>e) Les terrains de caravanes et de camping sauf dans le secteur ND c.
            </texte>
         </paragraphe>
         <paragraphe>
            <texte>f) Les abris fixes d'une surface supérieure à 15 m2.</texte>
         </paragraphe>
         <paragraphe>
            <texte>g) L'ouverture et l'exploitation des carrières.</texte>
         </paragraphe>
         <paragraphe>
            <secteur type="NDi"/>
            <texte>h) toutes constructions en secteur NDi.</texte>
         </paragraphe>
      </article>
      ...
   </zone>
   ...
</reglement>
```

Il est possible de décrire ce document de manière hiérarchique, de la même manière que dans l'exemple précédent. Cela permet de mieux se représenter la structure du règlement.

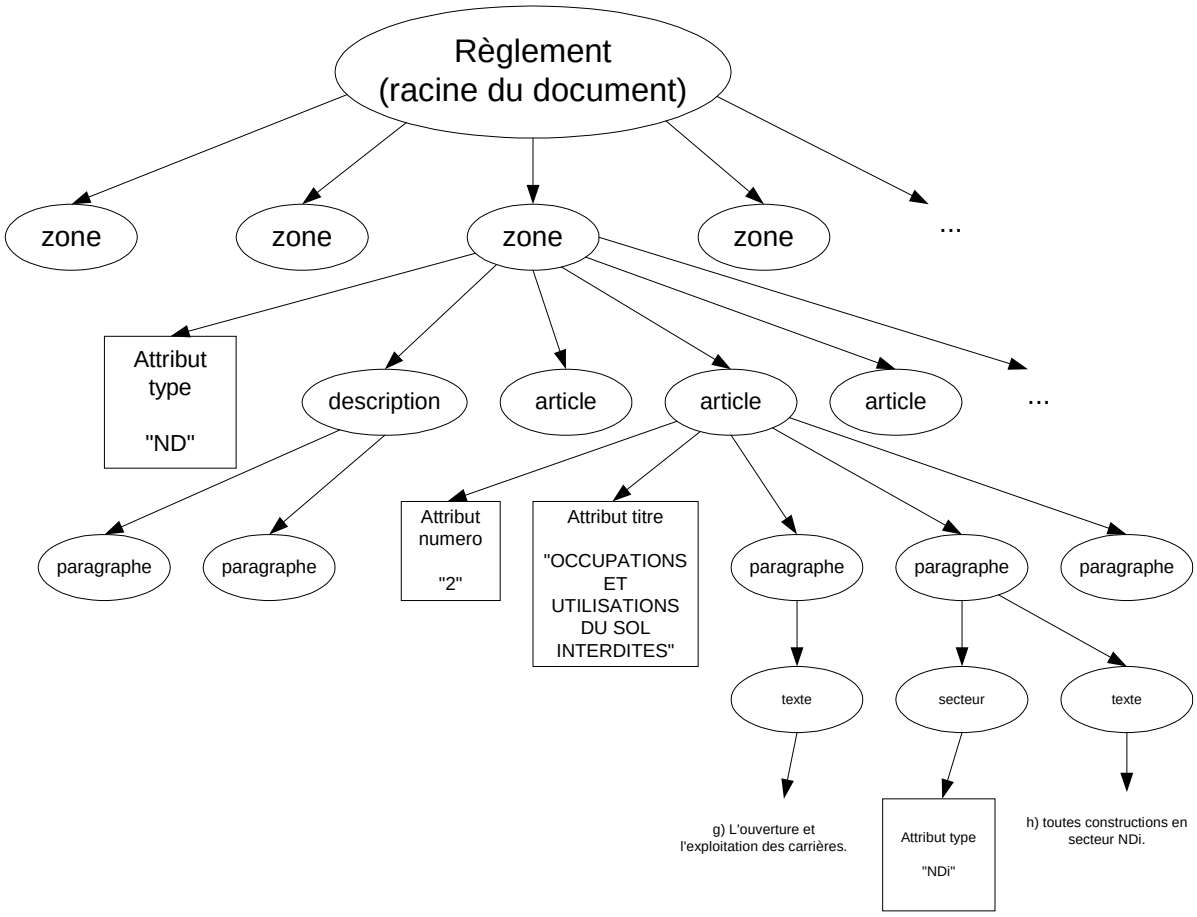


Figure 18 : Hiérarchie du document du règlement

Il faut noter que pour chaque paragraphe, on a un élément « texte » qui correspond au texte du paragraphe, et éventuellement un ou plusieurs éléments « secteur » qui correspondent aux secteurs pour lesquels le texte de ce paragraphe s'applique. S'il n'y a pas d'élément « secteur », le texte s'applique à tous les secteurs de la zone. Par exemple, le point h) de l'extrait ci-dessus ne s'applique qu'au secteur NDi. Certains paragraphes du règlement s'appliquent partout dans la zone sauf dans un secteur donné. On aurait pu alors définir des exclusions. Pour simplifier, on a préféré dans ce cas lister tous les secteurs où le paragraphe s'applique. Par exemple, pour le point e) qui s'applique partout sauf dans le secteur NDc, on a la liste de tous les autres secteurs, plus le « pseudo-secteur » ND qui correspond ici à toutes les parties de la zone ND qui ne font pas partie d'un secteur. On peut ainsi définir précisément pour chaque paragraphe à quel(s) type(s) de secteur il s'applique.

Il est possible de définir formellement une grammaire décrivant le langage utilisé pour représenter le règlement. Celle-ci définit les différents éléments du document XML, leurs attributs et les autres éléments qu'ils peuvent contenir. Elle peut être définie grâce à une DTD ou avec un schéma XML.

2.4.1.4 DTD du langage décrivant le règlement

L'intérêt de créer une DTD (Document Type Definition) pour définir le langage de représentation du règlement est multiple. Tout d'abord, cette DTD est réutilisable pour créer d'autres documents du même type. Si on considère qu'il y a un P.O.S. par commune en France, on aurait environ 36000 règlements différents (en réalité, il en existe beaucoup moins). En utilisant la même DTD, on pourra garantir que tous ces documents présentent la même structure.

Ensuite, une DTD permet de valider un document en utilisant un outil approprié. Cela permet de vérifier que le document est **bien formé** (il respecte la syntaxe de XML), et **valide** (il est conforme à la DTD). En produisant uniquement des documents valides, on s'assure qu'ils pourront être mis en forme ou utilisés indifféremment par une application. Par exemple, notre maquette permettant la consultation par le Web du P.O.S de Mazamet pourrait aussi bien servir à consulter le P.O.S d'une autre commune si son règlement est valide.

Voici donc la DTD correspondant à un document de règlement :

```
<?xml version="1.0" encoding="UCS-2"?>
<!ELEMENT article (paragraphe)+>
<!ATTLIST article
        numero CDATA #REQUIRED
        titre CDATA #REQUIRED>

<!ELEMENT secteur EMPTY>
<!ATTLIST secteur type CDATA #REQUIRED>

<!ELEMENT texte (#PCDATA)>
<!ELEMENT paragraphe ((secteur)*,texte)>
<!ELEMENT description (paragraphe)+>
<!ELEMENT zone (description,(article)+)>
<!ATTLIST zone type CDATA #REQUIRED>

<!ELEMENT reglement (zone)+>
```

Cette DTD est relativement simple. Elle contient uniquement des déclarations d'éléments et de listes d'attributs. D'autres DTD peuvent contenir des déclarations d'entités ou de notations. Nous ne détaillerons pas ces autres types de déclarations ici, car notre exemple est suffisant pour présenter les avantages et les inconvénients des DTD.

Les déclarations d'éléments (balise <!ELEMENT>) permettent de définir la structure des éléments du document. Par exemple, un élément zone est composé d'une description et d'un ou plusieurs articles. Les caractères spéciaux utilisés pour définir la cardinalité d'un élément sont les suivants :

- * : 0 ou plus
- + : 1 ou plus
- ? : 0 ou 1
- pas de caractère : 1 élément

Le mot-clé #PCDATA (parsed character data) indique que l'élément contient du texte.

Les déclarations de listes d'attributs (balise <!ATTLIST>) définissent les attributs correspondant à un élément. Le mot-clé CDATA indique que la valeur de l'attribut est une chaîne de caractères, et le mot-clé #REQUIRED indique qu'il est requis pour cet élément.

Une DTD permet ainsi de définir très simplement la **structure** d'un document. Par contre, elle ne donne presque aucune indication sur la forme du **contenu** du document. De plus, elle n'est pas exprimée en langage XML.

2.4.1.5 Schéma XML

Le rôle d'un schéma XML est très proche de celui de la DTD. Il décrit également la structure du document XML, mais il permet de détailler le contenu des éléments et de typer les attributs. Cela peut être indispensable lorsqu'on utilise des fichiers XML dans certaines applications (bases de données, …). Dans notre exemple, un schéma XML permet notamment de vérifier que le document comporte bien 15 articles pour chaque type de zone, et que les numéros des articles sont des entiers compris entre 1 et 15. Voici le schéma XML utilisé pour décrire un règlement :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

	<xsd:element name="secteur">
		<xsd:complexType>
			<xsd:attribute name="type" type="xsd:string" use="required"/>
		</xsd:complexType>
	</xsd:element>

	<xsd:element name="description">
		<xsd:complexType>
			<xsd:sequence maxOccurs="unbounded" minOccurs="1">
				<xsd:element ref="paragraphe"/>
			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>

	<xsd:element name="texte" type="xsd:string">
	</xsd:element>

	<xsd:element name="paragraphe">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:sequence minOccurs="0" maxOccurs="unbounded">
					<xsd:element ref="secteur"/>
				</xsd:sequence>
				<xsd:element ref="texte"/>
			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>

	<xsd:element name="article">
		<xsd:complexType>
			<xsd:sequence maxOccurs="unbounded" minOccurs="1">
				<xsd:element ref="paragraphe"/>
			</xsd:sequence>
			<xsd:attribute name="numero" type="NumArticle" use="required"/>
			<xsd:attribute name="titre" type="xsd:string" use="required"/>
		</xsd:complexType>
	</xsd:element>

	<xsd:element name="zone">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element ref="description"/>
				<xsd:sequence maxOccurs="15" minOccurs="15">
					<xsd:element ref="article"/>
				</xsd:sequence>
			</xsd:sequence>
			<xsd:attribute name="type" type="xsd:string" use="required"/>
		</xsd:complexType>
	</xsd:element>

	<xsd:element name="reglement">
		<xsd:complexType>
			<xsd:sequence maxOccurs="unbounded" minOccurs="1">
				<xsd:element ref="zone"/>
			</xsd:sequence>
		</xsd:complexType>
```

```
        </xsd:element>

        <xsd:simpleType name="NumArticle">
                <xsd:restriction base="xsd:positiveInteger">
                        <xsd:minInclusive value="1"/>
                        <xsd:maxInclusive value="15"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

On peut tout d'abord noter qu'un schéma xml est exprimé en syntaxe XML. Cela permet d'avoir une meilleure cohérence entre les données et leur description, mais surtout cela permet de valider un schéma XML avec les mêmes outils qu'on utilise pour valider un document. Cela procure un avantage non négligeable par rapport aux DTD.

Le second avantage réside dans les possibilités offertes par les schémas XML pour décrire des types de données et les appliquer aux éléments et aux attributs. Prenons comme exemple le type « NumArticle » :

```
        <xsd:simpleType name="NumArticle">
                <xsd:restriction base="xsd:positiveInteger">
                        <xsd:minInclusive value="1"/>
                        <xsd:maxInclusive value="15"/>
                </xsd:restriction>
        </xsd:simpleType>
```

L'élément <xsd:simpleType> indique que l'on décrit ici un type non composé, c'est à dire qui ne contient pas d'attributs ni d'éléments. <xsd:restriction> indique qu'il s'agit d'une restriction d'un type prédéfini, ici un entier positif. La restriction qui s'applique est que sa valeur minimale est 1 et sa valeur maximale est 15.

Les déclarations d'éléments se font de la même façon, en déclarant l'élément par la balise <xsd:element>, puis en déclarant son type.

```
        <xsd:element name="paragraphe">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                                        <xsd:element ref="secteur"/>
                                </xsd:sequence>
                                <xsd:element ref="texte"/>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
```

Cet élément est de type complexe car il contient d'autres éléments, ici un ensemble d'éléments « secteur » et un élément « texte ». On peut définir précisément le nombre d'occurrences de chaque élément grâce aux attributs « minOccurs » et « maxOccurs ».

Cet exemple montre bien l'intérêt des schémas XML par rapport aux DTD pour définir la structure et les types d'un document XML. Il ne s'agit bien sûr pas d'une description exhaustive. Les liens suivants permettent de trouver des informations plus complètes sur les schémas XML, qui font l'objet d'une recommandation du W3C depuis le 2 Mai 2001.

http://www.w3.org/XML/Schema : Spécifications des schémas XML

http://xmlfr.org/documentations/tutoriels/001218-0001 : Un tutoriel en français sur les schémas XML

2.4.1.6 Comparaison entre DTD et schéma XML et préconisations

	Avantages	**Inconvénients**
DTD	+ Simplicité + Compacité + Technique déjà éprouvée par SGML	- Typage faible - Syntaxe non XML
Schéma XML	+ Syntaxe XML + Nombreux types prédéfinis + Types adaptables par restriction ou extension	- Syntaxe relativement lourde - Technique encore jeune

Les DTD sont tout à fait adaptées pour décrire la structure de **documents**. Les schémas XML se positionnent plutôt sur l'aspect description de **données**. C'est le principal point dont il faut tenir compte lors du choix de l'utilisation de l'une ou l'autre des technologies. Si on prévoit d'utiliser XML pour structurer des documents afin de les présenter sur différents médias, alors les DTD sont mieux adaptées. Si on utilise XML pour stocker ou transporter des données sur lesquelles on souhaite effectuer des traitements, alors nous préconisons l'utilisation de schémas XML.

Une fois que l'on a pris en compte ces aspects fondamentaux, il faut savoir que les schémas XML permettent de décrire tout ce qui peut être décrit par une DTD. De plus, il semble que la tendance générale s'oriente vers les schémas XML plutôt que vers les DTD. Dans le doute, mieux vaut donc utiliser des schémas XML.

Les langages qui font l'objet d'une recommandation du W3C disposent pour la plupart d'une DTD. Par exemple, la DTD du langage SVG que nous allons étudier maintenant est disponible à l'adresse suivante :

http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd : DTD du langage SVG 1.0

2.4.2 Document SVG : Zones et secteurs géographiques

2.4.2.1 Présentation de SVG

Le format SVG (Scalable Vector Graphics) est un langage XML destiné à décrire des graphiques vectoriels. En utilisant un visualisateur adapté, on peut afficher ces graphiques, zoomer ou se déplacer dans ceux-ci, et les lier avec des informations textuelles ou d'un autre type. SVG permet également d'intégrer et d'afficher des images raster et du texte.

Voici un résumé des possibilités offertes par SVG :

- Dessin de formes vectorielles simples ou complexes (lignes, ellipses, polygones, courbes, …),
- Dessin de texte avec différentes polices et différents styles,
- Styles de traits et de remplissages personnalisables (motifs, gradients de couleurs, …),
- Intégration de feuilles de style pour appliquer le même style à tous les objets d'une certaine classe,
- Intégration d'images raster,
- Animations,
- Filtres,

- …

Ce langage est parvenu au statut de recommandation du W3C le 4 Septembre 2001 pour sa version 1.0. Il est donc stable et peut servir de base à des développements durables. Ses spécifications sont consultables à l'adresse suivante :

http://www.w3.org/TR/2001/REC-SVG-20010904/ : recommandation du W3C pour SVG 1.0

Il existe plusieurs outils permettant de visualiser un document SVG. Ceux-ci seront détaillés dans le chapitre 4.2, mais il est important de savoir que le plus utilisé, Adobe SVG Viewer, est disponible sous forme de plug-in et permet ainsi d'intégrer des documents SVG dans une page Web.

2.4.2.2 Le document SVG

Un document SVG est un document XML ayant pour racine la balise <svg>, et contenant des éléments graphiques décrits en langage SVG. Cet élément <svg> peut être inclus dans un autre document XML, on parle alors de fragment de document SVG.

L'élément <svg> lui-même permet de définir la taille du graphique et quelques autres attributs généraux. Il contient les déclarations des éléments graphiques eux-mêmes. Voici une liste de quelques-uns de ces éléments :

- <g> : définit un groupe d'éléments qui partagent les mêmes propriétés (style, …)
- <texte> : élément permettant de dessiner un texte
- <image> : élément permettant d'inclure une image
- <rect> : dessine un rectangle
- <circle> : dessine un cercle
- <line> : dessine une ligne
- <polygon> : dessine un polygone
- <path> : définit un chemin. Un chemin est une structure géométrique un peu plus complexe qu'un simple polygone, qui permet d'inclure des courbes et d'utiliser plusieurs polygones.
- <animate> : permet de définir une animation

La plupart de ces éléments ont comme attributs des coordonnées, un style (qui permet de définir la couleur de remplissage, le type de trait, etc.) et d'autres qui permettent d'exécuter des scripts lors de certains événements (clic de souris, etc.)

2.4.2.3 Carte du P.O.S.

Une carte de P.O.S. est en fait simplement un ensemble de zones et de secteurs auxquels on associe un type. Ces zones et secteurs peuvent être représentés comme des polygones. Le document contient simplement une liste de chemins qui sont groupés en types de zones et en types de secteurs. Voici un extrait de SVG d'un P.O.S. :

```
<g id="ZonesUA" class="UA">
<path id="Zone5" d="M4266 1667 L4266 1668 4252 1663 4223 1661 4221 1664 4225 1671 4235 1675
4248 1683 4251 1689 4260 1699 4265 1705 4227 1707 4225 1703 4206 1704 4203 1704 4195 1703 4177
1701 4167 1699 4170 1690 4157 1688 4144 1682 4147 1672 4124 1667 4117 1662 4117 1661 4118 1661
4123 1663 4128 1662 4135 1638 4130 1633 4152 1601 4164 1611 4179 1599 4189 1603 4189 1607 4208
1613 4206 1621 4220 1626 4221 1625 4234 1629 4262 1635 4263 1634 4263 1635 4265 1665 4265 1666
4265 1667 4266 1667
 Z" />
<path id="Zone33" d="M3415 1776 L3412 1781 3411 1782 3405 1779 3404 1780 3401 1780 3402 1777
3393 1775 3387 1799 3385 1807 3374 1824 3360 1844 3353 1857 3351 1866 3338 1864 3364 1820 3367
1816 3373 1805 3374 1799 3375 1798 3378 1787 3372 1785 3373 1780 3374 1772 3364 1771 3367 1753
```

```
3362 1752 3360 1754 3357 1755 3354 1754 3353 1752 3343 1747 3337 1759 3343 1760 3342 1770 3346
1770 3340 1806 3317 1800 3320 1789 3313 1786 3316 1771 3311 1770 3303 1771 3302 1766 3300 1766
3300 1764 3296 1765 3295 1759 3306 1755 3301 1745 3307 1742 3307 1745 3311 1745 3310 1739 3313
1738 3313 1739 3318 1738 3318 1739 3325 1737 3324 1734 3334 1730 3335 1728 3333 1723 3332 1722
3329 1707 3326 1708 3325 1702 3321 1702 3319 1686 3310 1686 3297 1686 3297 1679 3297 1673 3300
1666 3306 1658 3311 1663 3328 1664 3338 1663 3347 1661 3346 1654 3336 1626 3336 1624 3331 1598
3352 1603 3351 1615 3351 1619 3347 1627 3382 1633 3388 1634 3389 1634 3410 1634 3430 1634 3432
1625 3443 1622 3471 1616 3474 1629 3476 1635 3477 1657 3478 1681 3472 1685 3469 1686 3472 1693
3469 1694 3462 1696 3464 1701 3463 1702 3464 1705 3465 1704 3466 1707 3458 1709 3457 1705 3454
1691 3448 1692 3440 1691 3439 1691 3430 1691 3427 1698 3426 1702 3423 1708 3420 1713 3418 1713
3415 1722 3412 1729 3406 1727 3405 1740 3402 1758 3408 1760 3410 1761 3409 1765 3408 1765 3407
1773 3415 1776
 Z" />
<path id="Zone38" d="M3236 1895 L3226 1897 3224 1897 3207 1897 3207 1898 3207 1884 3242 1884
3242 1894 3236 1895
 Z" />
<path id="Zone47" d="M3313 1497 L3314 1512 3317 1527 3317 1528 3318 1531 3314 1534 3308 1542
3303 1549 3289 1542 3282 1542 3282 1533 3297 1526 3301 1525 3300 1524 3300 1521 3294 1521 3294
1511 3288 1503 3282 1508 3276 1508 3276 1501 3276 1452 3307 1490 3308 1490 3313 1497
 Z" />
<path id="Zone48" d="M3457 2537 L3459 2537 3461 2535 3462 2534 3463 2532 3464 2533 3465 2533
3463 2526 3464 2514 3463 2509 3461 2504 3458 2501 3464 2491 3466 2491 3470 2488 3472 2486 3473
2479 3474 2476 3476 2477 3480 2467 3484 2464 3485 2456 3485 2450 3493 2450 3497 2448 3499 2448
3500 2448 3500 2451 3501 2451 3506 2451 3505 2455 3505 2456 3506 2455 3506 2456 3510 2454 3514
2464 3515 2464 3517 2463 3517 2476 3519 2479 3521 2484 3520 2485 3519 2493 3516 2507 3515 2515
3506 2501 3505 2501 3504 2502 3511 2526 3514 2530 3508 2534 3499 2539 3492 2541 3483 2542 3486
2550 3474 2559 3468 2563 3467 2558 3458 2548 3454 2546 3457 2544 3457 2541 3455 2541 3457 2537
 Z" />
<g id="SecteursUAa" class="UAa" style="fill:none">
<path id="Secteur47" d="M3431 1634 L3432 1625 3443 1622 3471 1616 3476 1635 3478 1681 3467
1688 3454 1691 3452 1681 3449 1681 3437 1681 3427 1680 3414 1677 3421 1663 3428 1643 3431 1634
 Z" />
<path id="Secteur48" d="M3474 2559 L3468 2563 3467 2558 3457 2548 3454 2546 3456 2544 3457
2541 3455 2541 3457 2537 3459 2537 3460 2535 3461 2534 3463 2532 3464 2532 3463 2526 3463 2514
3462 2509 3460 2504 3458 2501 3464 2491 3466 2491 3465 2491 3470 2488 3471 2486 3472 2479 3473
2476 3476 2477 3480 2467 3483 2464 3485 2456 3485 2450 3493 2450 3497 2448 3498 2448 3500 2448
3500 2451 3506 2451 3505 2456 3510 2454 3514 2464 3517 2463 3517 2476 3519 2479 3521 2484 3520
2485 3519 2493 3516 2507 3515 2515 3514 2515 3505 2501 3504 2502 3511 2526 3514 2530 3508 2534
3499 2539 3492 2541 3483 2542 3486 2550 3474 2559
 Z" />
</g>
</g>
```

Chaque groupe (balise <g>) correspond à un type de zone ou de secteur. Sa classe permet entre autres de déterminer quel style doit être appliqué aux chemins correspondants (couleur de remplissage, …). Les chemins (balises <path>) sont décrits par leurs coordonnées (attribut « d »). La lettre « M » et les deux chiffres qui suivent correspondent aux coordonnées du début du chemin. La lettre « L » et les paires de chiffres qui suivent correspondent aux segments qui décrivent ce chemin. La lettre « Z » indique la fin du chemin.

On peut ensuite visualiser ce document SVG en utilisant un outil adapté (Adobe SVG Viewer par exemple). Voici ce que donne le document SVG du P.O.S. après visualisation :

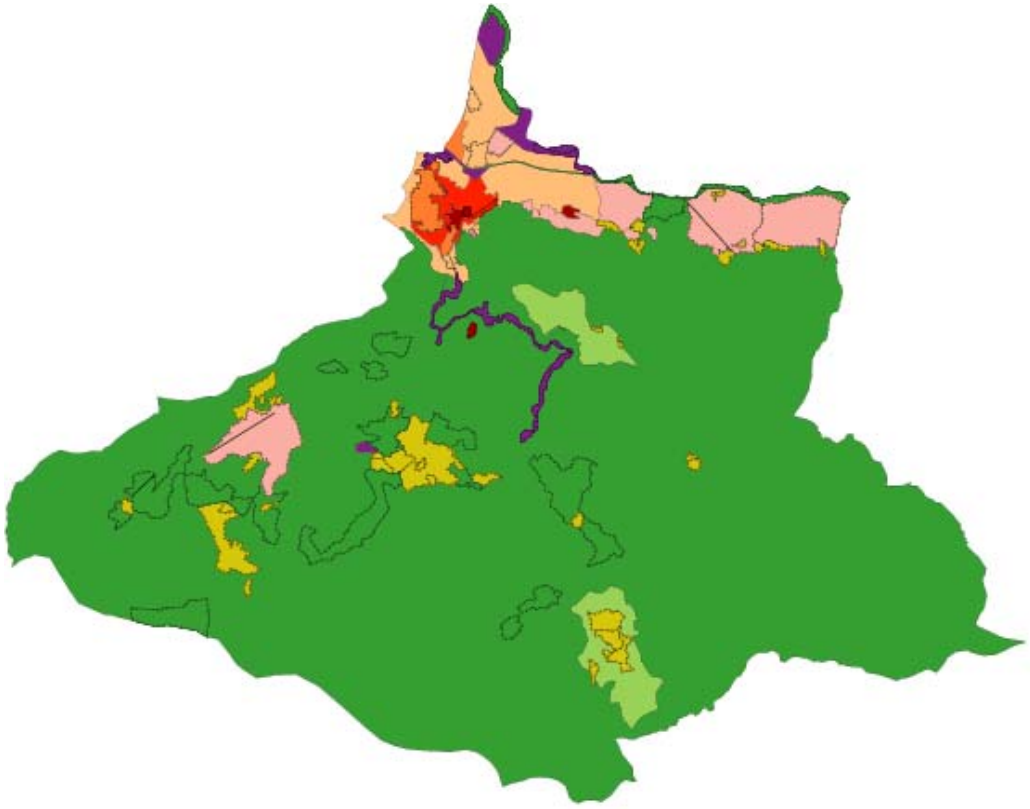


Figure 19 : Carte du P.O.S. en SVG

Le langage SVG est uniquement un langage graphique. Les coordonnées des points n'ont donc pas de signification propre. Ceux-ci sont définis dans un « espace utilisateur » dans lequel les coordonnées sont des nombres réels. L'attribut « viewBox » de l'élément SVG définit les coordonnées x et y, la largeur et la hauteur de la partie représentée de cet espace utilisateur. On peut ensuite zoomer et se déplacer dans celui-ci à volonté (ou au moins dans les limites autorisées par le visualisateur).

Dans notre cas, on souhaite représenter des données géographiques, donc les unités utilisées pour les coordonnées sont importantes. Les outils avec lesquels nous avons converti les données ne permettent malheureusement pas de contrôler la manière dont s'effectue cette conversion. Dans l'extrait ci-dessus, les coordonnées n'ont donc aucune signification d'un point de vue géographique. Il serait nécessaire d'adapter ces outils ou d'en développer un nouveau permettant de contrôler la manière dont s'effectue la projection et la transcription des coordonnées. On pourrait alors facilement indiquer l'échelle de la carte ou éditer une carte à une échelle donnée.

3. TECHNOLOGIES UTILISEES

3.1 ARCHITECTURES

Dans le cas de la consultation à distance de documents XML et XML / SVG, on peut distinguer deux types d'architecture.

- L'architecture « application locale » : Dans ce cas, le client télécharge toutes les informations relatives à la carte et au règlement. Il n'y a plus d'échange avec le serveur par la suite. Les fichiers peuvent également être gravés sur un CD-ROM pour une utilisation nomade.

- L'architecture « client léger » : Dans ce cas, le client ne télécharge que le fichier SVG et les fichiers HTML et Javascript nécessaires à son exploitation. Toutes les données du règlement du P.O.S. restent sur le serveur et ne sont accessibles que sur demande.

3.1.1 Application locale

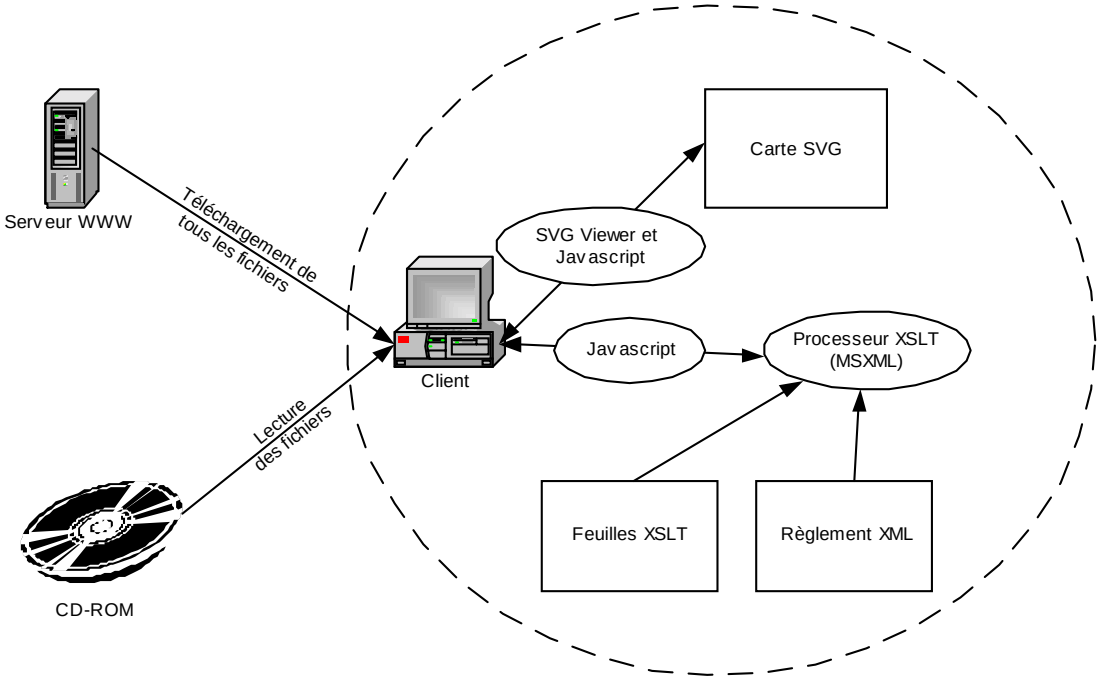


Figure 20 : Architecture "application locale"

Dans ce type d'architecture, tous les traitements XSLT sont effectués sur le poste client par le processeur MSXML, intégré à Microsoft Internet Explorer. On peut donc déjà noter que cette architecture ne fonctionne pas lorsqu'on utilise d'autres navigateurs, comme par exemple Netscape Navigator. Le chargement des fichiers XML et l'interface avec MSXML sont effectués par du code en Javascript intégré dans la page Web. Tous les traitements étant effectués en local sur le client, on peut facilement mettre ces données sur CD-Rom pour une utilisation nomade. C'est là le principal avantage de cette architecture.

3.1.2 Client léger

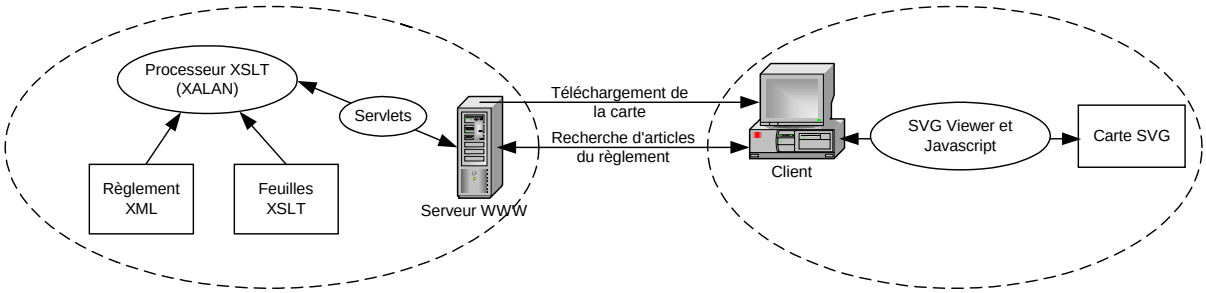


Figure 21 : Architecture "client léger "

Dans ce cas, tous les traitements XSLT sont effectués sur le serveur par le processeur XALAN d'Apache, qui utilise Xerces pour le parsing des documents XML. Il est un peu moins performant que MSXML mais il présente l'avantage d'être entièrement écrit en Java. Il est donc facile de l'utiliser avec une servlet. Les servlets servent simplement à recevoir la requête, à invoquer le processeur Xalan pour la transformation et à renvoyer la réponse. Dans cette architecture, les données reçues par le client sont uniquement en HTML ou en SVG. Elles sont donc utilisables par n'importe quel navigateur équipé du plug-in Adobe SVG Viewer. Il faut toutefois noter qu'on ne peut pas se dispenser d'un peu de code Javascript pour les interactions avec la carte.

3.1.3 Comparaison et préconisations

	Avantages	**Inconvénients**
Application locale	+ Possibilité de consulter les informations « hors ligne » ou sur CD-ROM + Aucun traitement sur le serveur → Moins de charge	- Téléchargement des fichiers plus long - Nécessité d'un système client plus performant - Ne fonctionne aujourd'hui qu'avec Microsoft Internet Explorer - Certaines fonctions ne sont pas réalisables
Client léger	+ Relativement peu de données à charger sur le client → Accès plus rapide au démarrage + Fonctionne théoriquement avec tous les navigateurs	- Nombreux échanges client-serveur → Plus grande charge sur le serveur

Le choix de l'une ou l'autre architecture sera effectué en fonction de l'importance que l'on accorde aux différents points cités ci-dessus et des orientations prises pour la diffusion des données. Toutefois, nous préconisons l'architecture client léger, qui se base sur des technologies Java qui garantissent une meilleure portabilité. Cette architecture permet également d'offrir plus de fonctionnalités comme par exemple la génération de PDF (Cela serait possible dans l'application locale uniquement en développant une applet très volumineuse car elle utiliserait des librairies spécifique. On peut toutefois envisager cette possibilité dans le cadre du développement d'une application sur CD-ROM).

3.2 REPRESENTATION ET DECOMPOSITION DES FICHIERS XML : LES APIS DOM ET SAX

Pour qu'une application puisse utiliser un document XML, il est nécessaire qu'elle puisse le décomposer et en construire une représentation interne. Ces opérations sont réalisées grâce à des « parseurs » XML. On en distingue deux types, qui diffèrent par leur mode de fonctionnement : les parseurs basés sur l'API DOM et ceux qui sont basés sur l'API SAX.

3.2.1 Présentation de DOM

Le modèle DOM (Document Object Model) est utilisé pour représenter un document XML de manière hiérarchique, sous forme d'arbre (voir section 2.4.1). L'application peut alors accéder arbitrairement à n'importe quel élément en parcourant cet arbre. Ce modèle DOM a déjà fait l'objet de plusieurs recommandations du W3C. Celle du modèle DOM niveau 2 date du 13 Novembre 2000 et peut être consultée à l'adresse suivante :

http://www.w3.org/TR/DOM-Level-2-Core/ : Recommandation du W3C pour DOM niveau 2

http://www.w3.org/TR/DOM-Level-3-Core/ : Working Draft pour DOM niveau 3 (14 Janvier 2002)

Les parseurs DOM (Document Object Model) traitent l'ensemble du document XML, et en construisent une représentation sous forme d'arbre. Ces parseurs sont adaptés lorsque l'on doit accéder à des éléments de l'arbre de manière aléatoire (navigation dans le document). De plus, le modèle DOM est implémenté par les navigateurs Web et permet ainsi de parcourir les documents pour les applications Web. Il devient par contre inutilisable lorsque le nombre d'éléments devient trop important. Il est implémenté par quasiment toutes les applications qui manipulent des données XML. Par exemple, Adobe SVG Viewer construit une représentation du document SVG en utilisant le modèle DOM.

3.2.2 Présentation de SAX

SAX (Simple API for XML) est une API utilisée également pour décomposer des documents XML. A l'origine, elle était conçue uniquement pour Java, mais d'autres implémentations ont fait leur apparition depuis. La version actuelle est SAX 2.0, mais il faut noter que cette API ne fait pas l'objet d'une recommandation du W3C. Son développement est donc un peu plus anarchique, mais on trouve tout de même plusieurs implémentations stables de cette API. La page officielle est la suivante :

http://www.saxproject.org/ : Page officielle de SAX (Simple API for XML)

Les parseurs qui implémentent l'API SAX décomposent le document XML au fur et à mesure de la lecture, sans construire de représentation de l'ensemble du document. La programmation se fait de manière événementielle, c'est à dire que l'application traite des événements qui lui sont envoyés par le parseur (par exemple lorsqu'il rencontre une certaine balise). Chaque type d'événement reconnu sera traité par une fonction spécifique de l'application. Ces parseurs sont particulièrement adaptés pour les documents volumineux car ils ne chargent pas l'ensemble de ceux-ci en mémoire, et sont donc peu gourmands en ressources.

3.2.3 Comparaison de DOM et SAX

	Avantages	Inconvénients
DOM	+ Possibilité d'accès aléatoire à n'importe quel élément + Très utilisé + Recommandation du W3C	- Inadapté pour les documents volumineux
SAX	+ Peu gourmand en ressources	- Pas de représentation de l'ensemble du document

Les deux modèles DOM et SAX sont très différents, par leur implémentation et leurs utilisations. Nous ne pouvons pas préconiser l'utilisation d'un modèle plutôt que l'autre, car tout dépend de l'application. En général, lorsqu'il s'agit de transformer un document volumineux ou d'en extraire des informations en une seule fois, SAX est plus adapté. Lorsqu'on a besoin d'une représentation interne pour manipuler les différents éléments ou accéder à leur contenu de manière aléatoire, l'utilisation de DOM est recommandée. Les applications utilisant XML ont souvent recours aux deux modèles parallèlement.

3.2.4 JAXP et les différentes implémentations de parseurs

JAXP (Java API for XML Processing) est une API développée par Sun pour le traitement de documents XML par DOM, SAX ou XSLT. En définissant une interface commune pour ces traitements, JAXP permet au développeur de choisir quel processeur utiliser sans changement de code. Comme leur nom l'indique, les processeurs JAXP ne fonctionnent qu'en environnement Java. Voici la liste des plus répandus :

- Parseurs JAXP :
  - Apache Crimson (anciennement Sun Project X) : parseur XML supportant SAX et DOM
  - Apache Xerces 1 (anciennement IBM XML4J) : parseur XML supportant SAX et DOM
  - Apache Xerces 2 : parseur XML entièrement réécrit supportant SAX et DOM
  - GNU JAXP : parseur XML supportant SAX2 et DOM niveau 2

- Processeurs XSLT JAXP :
  - Apache Xalan-J XSLT : Processeur XSLT
  - Saxon XSLT : Processeur XSLT

La fondation Apache est un des acteurs majeurs des technologies XML et Java. Elle propose notamment avec Xalan et Xerces un moyen simple et efficace d'effectuer des traitements XSLT sur des documents XML. De plus, il est très facile d'intégrer Xalan et Xerces à un serveur Apache Tomcat, ce qui permet de déclencher ces traitements par une Servlet. Pour la maquette « client léger », nous utilisons Xalan 2 et Xerces 2. Nous préconisons d'utiliser ces librairies car elles sont peu coûteuses et fiables.

Pour les implémentations non-Java de parseurs, on distingue principalement MSXML de Microsoft qui est maintenant disponible dans sa version 4.0. Il offre des parseurs SAX et DOM et un processeur XSLT

performants. Le principal avantage de MSXML est qu'il est intégré à Internet Explorer. C'est donc grâce à lui que nous avons réalisé la maquette « application locale ».

	Avantages	**Inconvénients**
Parseurs et processeurs JAXP	+ Indépendants de la plate-forme utilisée + On peut choisir l'implémentation que l'on souhaite utiliser + Facilement intégrable à des applications Java	- Uniquement utilisable avec Java
MSXML	+ Performance + Intégration possible avec VB ou ASP + Intégré à Internet Explorer	- Pas portable - Pas d'intégration avec Java - Une seule implémentation

3.3 TRANSFORMATION DE DOCUMENTS XML AVEC XSLT ET XPATH

3.3.1 Présentation de XSLT

Le langage XSLT (eXtensible Stylesheet Language Transformations) permet de décrire des transformations permettant de créer un nouveau document XML ou un document HTML à partir d'un document XML. Il est utilisé principalement pour créer différentes vues d'un même document pour les présenter à un utilisateurs donné, mais il peut être utilisé pour effectuer des traitements beaucoup plus complexes. XSLT a été conçu pour être utilisé avec XSL-FO (eXtensible Stylesheet Language – Formatting Objects), mais dans la majorité des cas on l'utilise indépendamment. XSLT et XSL-FO constituent ensemble XSL (eXtensible Stylesheet Language), le langage de feuilles de styles de XML. XSLT est devenu une recommandation du W3C le 16 Novembre 1999. Il est donc stable et est déjà implémenté par plusieurs applications et bibliothèques.

http://www.w3.org/TR/xslt : Recommandation du W3C pour le langage XSLT

Son fonctionnement repose sur des règles ou *templates* qui indiquent au processeur quel traitement il doit effectuer lorsqu'il rencontre tel ou tel élément. Il permet de sélectionner certains éléments grâce au langage XPath pour appliquer les règles. On peut aussi définir des règles nommées paramétrables que l'on appellera comme des fonctions. Enfin il existe des instructions de test et de boucle qui rapprochent XSLT d'un vrai langage de programmation.

3.3.2 Quelques exemples d'utilisation de XSLT

Cette section a pour but de présenter le fonctionnement de XSLT avant de se pencher sur son utilisation dans la maquette, qui peut s'avérer un peu complexe au premier abord. Voici donc un premier exemple très simple qui permet de créer un document HTML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

   <xsl:template match="/">
      <html>
         <head>
            <title>Premier exemple</title>
         </head>
         <body>
            Voici un premier exemple simple.
         </body>
      </html>
   </xsl:template>

</xsl:stylesheet>
```

Ce document XSLT débute comme tout document XML par la déclaration XML. Ensuite vient l'élément <xsl:stylesheet>, racine du document. Elle déclare le domaine nominal de XSLT et la version du langage que l'on utilise.

La balise <xsl:template> indique le début d'une règle. L'attribut « match » indique à quel élément elle s'applique. En l'occurrence, il s'agit de la règle qui s'applique à la racine du document. C'est la première à être traitée. Elle sera traitée dans tous les cas, quel que soit le contenu du document.

Le code HTML qui vient ensuite constituera le document en sortie. On constate donc dans cet exemple que le code produit sera toujours le même quel que soit le document source. L'intérêt est donc très limité, si ce n'est pour présenter la structure minimale d'un document XSLT. Voici le document résultant visualisé par Microsoft Internet Explorer :

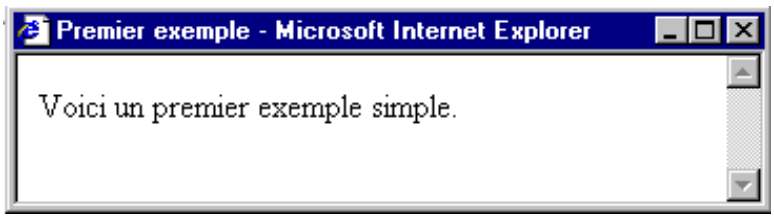


Figure 22 : Premier exemple d'utilisation de XSLT

Voici maintenant un exemple un peu plus intéressant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

   <xsl:template match="/">
      <html>
         <head>
            <title>Un autre exemple</title>
         </head>
         <body>
            Voici la liste des zones du règlement :
            <ol>
               <xsl:apply-templates select="reglement/zone"/>
            </ol>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="zone">
      <li>
         <xsl:value-of select="@type"/>
      </li>
   </xsl:template>

</xsl:stylesheet>
```

On remarque l'utilisation dans la règle principale de la balise <xsl:apply-templates> qui indique au processeur qu'il doit ici appliquer les règles qui concernent les éléments sélectionnés par l'expression indiquée par l'attribut « select ». En l'occurrence, cette expression sélectionne tous les éléments « zone » du règlement.

Le processeur applique donc la règle qui concerne les éléments « zone ». C'est celle qui est décrite par l'élément `<xsl:template match="zone">`. Cette règle indique que le processeur doit insérer une balise HTML de liste <li>, ainsi que la valeur du type de la zone (balise `<xsl:value-of select="@type"/>`).

Cet exemple permet donc de créer une liste avec les types de toutes les zones du règlement. Lorsqu'on applique cette transformation à notre exemple de règlement, cela donne le résultat suivant :

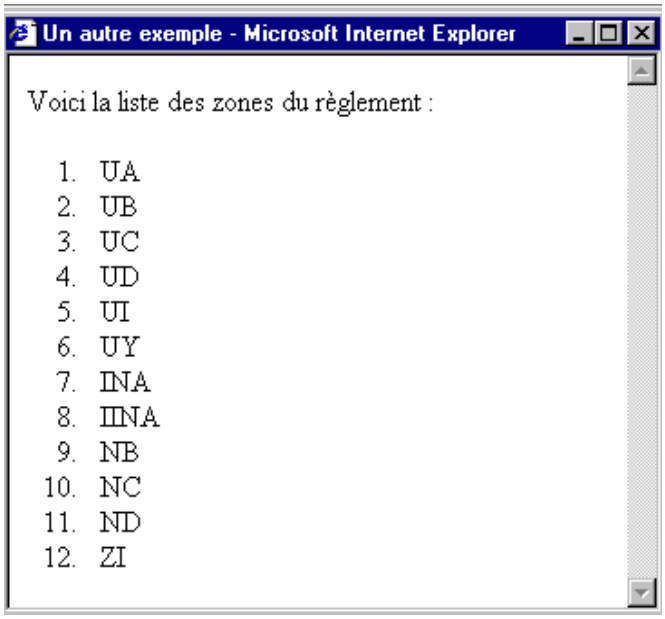


Figure 23 : Deuxième exemple d'utilisation de XSLT

Ces deux exemples donnent un aperçu des possibilités de XSLT. Il est maintenant intéressant de voir comment son utilisation avec XPath permet de faire des traitements plus complexes.

3.3.3 Présentation de XPath

XPath est le langage utilisé pour sélectionner un ou plusieurs fragments dans un document XML. Il est utilisé dans XSLT pour sélectionner les fragments auxquels on applique une règle. Il est devenu une recommandation du W3C en même temps que XSLT. Les spécifications complètes de XPath sont disponibles à l'adresse suivante :

http://www.w3.org/TR/xpath : Recommandation pour XPath version 1.0

Sa syntaxe est proche de celle des URL, avec quelques ajouts importants. Nous présenterons uniquement ici la syntaxe abrégée, plus simple et plus compacte. Il importe de distinguer les chemins relatifs et les chemins absolus. Un chemin absolu commence par le caractère '/' qui indique la racine du document. Il peut ensuite être suivi par un chemin relatif. Par exemple, « /reglement/zone » sélectionne toutes les zones du règlement.

Un chemin relatif est exprimé à partir d'un contexte. Un contexte correspond en fait à un nœud courant. Par exemple, si le contexte est le nœud correspondant à la zone « UA », l'expression « description/paragraphe » sélectionnera tous les paragraphes de la description de cette zone.

Le caractère '@' est utilisé pour sélectionner un attribut. Par exemple, l'expression « /reglement/zone/@type » sélectionne l'attribut « type » de toutes les zones du règlement.

Le caractère '.' désigne le nœud courant. Il peut être nécessaire dans certains cas.

Les caractères '..' désignent le nœud père du nœud courant.

Les caractères '//' permettent de parcourir l'ensemble des descendants d'un nœud. Par exemple, l'expression « /reglement//texte » sélectionne tous les éléments « texte » du règlement.

Les crochets '[ ]' indiquent un prédicat qui permet de filtrer certains éléments. Par exemple, l'expression « /reglement/zone[@type= 'UA']/description » sélectionne l'élément « description » de la zone de type « UA ».

Il existe également des fonctions pour indiquer la position d'un élément dans une liste. Par exemple, l'expression « /reglement/zone[position( ) = last( )] » sélectionne la dernière zone dans le règlement.

Les exemples précédents donnent un aperçu des moyens d'adressage fournis par XPath. Pour mieux comprendre son fonctionnement en association avec XSLT, le meilleur moyen est de s'intéresser à un exemple concret.

3.3.4 Sélection et mise en forme d'un article du règlement avec XSLT et XPath

On utilise ici XSLT pour présenter un extrait du règlement sous forme de page web. On cherche à sélectionner les paragraphes d'un article qui correspondent à une zone et à un secteur donnés. Pour cela, on utilise l'expression XPath suivante :

```
//zone[@type=$typeZone]/article[@numero=$numArticle]/paragraphe[secteur/@type=$typeSecteur or not(secteur)]/texte
```

On peut déjà noter l'utilisation de termes précédés par le caractère '$'. Ce sont des paramètres de la feuille XSLT. Nous verrons plus loin comment on peut changer leurs valeurs pour accéder aux différentes parties du règlement. L'expression étant quelque peu complexe, il est intéressant de la décomposer.

`//zone[@type=$typeZone]` : Sélectionne la zone de type « typeZone »

`/article[@numero=$numArticle]` : Sélectionne l'article de numéro « numArticle »

`/paragraphe[secteur/@type=$typeSecteur or not(secteur)]` : Sélectionne le ou les paragraphes qui contiennent soit un élément secteur de type « typeSecteur », soit aucun élément secteur.

`/texte` : Sélectionne l'élément texte.

Cette expression illustre les possibilités du langage XPath pour adresser une partie du document. Voyons maintenant comment elle est intégrée dans la feuille XSLT pour permettre la sélection et la mise en forme d'une partie du document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

   <xsl:param name="typeZone">UA</xsl:param>
   <xsl:param name="typeSecteur">UA</xsl:param>
   <xsl:param name="numArticle">0</xsl:param>

   <xsl:template match="/">
      <html>
         <head>
            <title>Informations</title>
         </head>
         <body>
            <p align="center" style="font-weight:bold">Informations</p>
            <p style="font-weight:bold">Zone : <xsl:value-of select="$typeZone"/></p>

            <xsl:if test="$typeSecteur != $typeZone">
               <p style="font-weight:bold">Secteur : <xsl:value-of select="$typeSecteur"/></p>
            </xsl:if>

            <xsl:choose>
               <xsl:when test="$numArticle='0'">
                  <p style="font-weight:bold">Description générale</p>
               </xsl:when>
               <xsl:otherwise>
                  <p style="font-weight:bold">Article <xsl:value-of select="$numArticle"/> :
<xsl:value-of select="//zone[@type=$typeZone]/article[@numero=$numArticle]/@titre"/></p>
```

```
               </xsl:otherwise>
            </xsl:choose>

            <xsl:choose>
               <xsl:when test="$numArticle='0'">
                  <xsl:apply-templates  select="//zone[@type=$typeZone]/description/paragraphe[
secteur/@type=$typeSecteur or not(secteur)]/texte"/>
               </xsl:when>
               <xsl:otherwise>
                  <xsl:apply-templates                 select="//zone[@type=$typeZone]/article
[@numero=$numArticle]/paragraphe[secteur/@type=$typeSecteur or not(secteur)]/texte"/>
               </xsl:otherwise>
            </xsl:choose>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="texte">
      <p><xsl:apply-templates/></p>
   </xsl:template>

</xsl:stylesheet>
```

Cette feuille XSLT est relativement simple. Elle contient uniquement deux règles. La seule difficulté réside ici dans les expressions XPath.

```
       <xsl:param name="typeZone">UA</xsl:param>
       <xsl:param name="typeSecteur">UA</xsl:param>
       <xsl:param name="numArticle">0</xsl:param>
```

On déclare ici les trois paramètres mentionnés plus haut, « typeZone », « typeSecteur » et « numArticle », et on leur affecte une valeur par défaut. Le paramètre « typeZone » correspond au type de la zone pour laquelle on souhaite obtenir des informations. Le paramètre « numArticle » est le numéro de l'article demandé. S'il est égal à 0, cela correspond à la description générale de la zone. Enfin, le paramètre « typeSecteur » correspond au type du secteur pour lequel on veut sélectionner les informations. S'il est égal à « typeZone », cela correspond à la partie générale de la zone.

```
<html>
       <head>
              <title>Informations</title>
       </head>
       <body>
              <p align="center" style="font-weight:bold">Informations</p>
              <p style="font-weight:bold">Zone : <xsl:value-of select="$typeZone"/></p>
```

Le code HTML contenu dans la règle sera recopié tel quel dans le document en sortie. On peut noter l'utilisation de la balise <xsl:value-of> pour inclure la valeur du paramètre « typeZone » dans le texte.

```
<xsl:if test="$typeSecteur != $typeZone">
       <p style="font-weight:bold">Secteur : <xsl:value-of select="$typeSecteur"/></p>
</xsl:if>
```

L'instruction <xsl:if> permet d'effectuer des traitements conditionnels. Ici, la ligne avec le nom du secteur n'apparaîtra que si le paramètre « typeSecteur » est différent du paramètre « typeZone ».

```
<xsl:choose>
       <xsl:when test="$numArticle='0'">
              <p style="font-weight:bold">Description générale</p>
```

```
        </xsl:when>
        <xsl:otherwise>
                <p  style="font-weight:bold">Article  <xsl:value-of  select="$numArticle"/>  :
<xsl:value-of select="//zone[@type=$typeZone]/article[@numero=$numArticle]/@titre"/></p>
        </xsl:otherwise>
</xsl:choose>
```

L'instruction <xsl:choose> permet également d'effectuer des traitements conditionnels, mais avec plusieurs choix possibles. Dans l'exemple ci-dessus, cela permet d'écrire la ligne « Description générale » si le paramètre « numArticle » est égal à 0. Dans le cas contraire, on écrit le numéro de l'article et son titre.

```
<xsl:choose>
        <xsl:when test="$numArticle='0'">
                <xsl:apply-templates     select="//zone[@type=$typeZone]/description/paragraphe[
secteur/@type=$typeSecteur or not(secteur)]/texte"/>
        </xsl:when>
        <xsl:otherwise>
                <xsl:apply-templates                        select="//zone[@type=$typeZone]/article
[@numero=$numArticle]/paragraphe[secteur/@type=$typeSecteur or not(secteur)]/texte"/>
        </xsl:otherwise>
</xsl:choose>
```

Cette expression est semblable à la précédente, sauf qu'elle fait appel à l'instruction <xsl:apply-templates>. Cela signifie qu'il faut ici appliquer les règles qui correspondent aux éléments désignés par l'attribut « select ». Il s'agit ici de l'expression XPath que nous avons détaillée précédemment dans le cas d'un article, ou d'une expression semblable lorsqu'on recherche la description générale de la zone. Les éléments sélectionnés sont de type « texte », donc la règle qui s'applique est la suivante :

```
        <xsl:template match="texte">
                <p><xsl:apply-templates/></p>
        </xsl:template>
```

Cette règle insère simplement des balises de début et de fin de paragraphe. L'appel à <xsl:apply-templates> sans attribut applique la règle par défaut, c'est à dire qu'elle recopie le contenu de l'élément courant dans le document en sortie. Au final, la transformation du règlement XML en utilisant cette feuille crée le document HTML suivant :

```
<html>
<head>
<META http-equiv="Content-Type"
content="text/html; charset=UTF-16">
<title>Informations</title>
</head>
<body>
<p align="center" style="font-
weight:bold">Informations</p>
<p style="font-weight:bold">Zone : NB</p>
<p style="font-weight:bold">Secteur : NBa</p>
<p style="font-weight:bold">Article 6 :
IMPLANTATION DES CONSTRUCTIONS PAR RAPPORT
AUX VOIES ET EMPRISES PUBLIQUES</p>
<p>6.1.  En  NBa,  les  constructions
peuvent  être  édifiées  sur l'alignement de
la voie. Lorsqu'elles  ne  sont  pas
édifiées  sur  l'alignement,  elles doivent
respecter, par rapport à celui-ci, une marge
de recul au moins égale à 5 mètres.</p>
<p>6.3.  Dans l'ensemble de la zone,  des
dispositions  différentes peuvent  être
autorisées  dans  le cas  d'une  construction
complétant  un alignement de façades
existant, à condition de ne pas le
prolonger.</p>
</body>
</html>
```

Les processeurs XSLT Xalan et MSXML fournissent des fonctions pour fixer les paramètres avant d'effectuer la transformation. Cela se fait très simplement, comme dans l'exemple suivant (pour Xalan) :

```
xslTransformer.setParameter("typeZone", typeZone);
xslTransformer.setParameter("typeSecteur", typeSecteur);
xslTransformer.setParameter("numArticle", numArticle);
```

Cet exemple illustre les possibilités du langage XSLT pour traiter des documents XML. Il est intéressant de noter que l'arborescence dans la partie gauche de la maquette est également générée dynamiquement par une transformation XSLT à partir du règlement XML. Dans la maquette « client léger », la légende est également générée par XSLT à partir du document SVG du POS. Ce langage peut donc être utilisé pour générer du HTML, du XML avec tous ses langages spécialisés, ou du texte simple. Les sources de ces deux documents XSLT sont fournis en annexe.

3.4 AUTRES LANGAGES DE REQUETE SUR DES DOCUMENTS XML

Le langage XPath permet très simplement de sélectionner certains éléments d'un document XML. Cela peut s'apparenter à une requête comme celles qu'on effectue en SQL sur une base de données. Il existe plusieurs langages qui permettent ainsi d'effectuer des requêtes sur des documents XML.

3.4.1 Historique

Le besoin pour un langage de requête est apparu très rapidement dès que XML a commencé à être utilisé. De nombreuses propositions ont été faites et plusieurs langages sont apparus. On peut notamment citer XQL et XML-QL en 1998. Ces langages étant développés indépendamment les uns des autres, le risque de voir se multiplier les standards est apparu. En décembre 1998, le W3C a donc organisé le « QL'98 Workshop » rassemblant les acteurs majeurs du domaine. Cela a permis d'étudier les besoins et de rassembler les propositions pour un langage de requête pour XML. Le W3C a depuis créé un groupe de travail sur ce sujet, afin de standardiser un langage. Le seul langage normalisé permettant jusqu'à un certain point d'effectuer des requêtes sur un document XML est aujourd'hui XPath. Nous l'avons déjà présenté dans la section précédente. Nous nous focaliserons donc ici sur XQL, un ancêtre de XPath encore utilisé, et sur XQuery.

3.4.2 XQL

XQL se présente comme un langage permettant d'effectuer des requêtes sur un document XML. En pratique, il se rapproche beaucoup de XPath dont il est un des ancêtres, à quelques différences de syntaxe près. Le principe d'un processeur XQL est de traiter un document XML avec une requête XQL, et de produire un document XML contenant les éléments résultant de la requête. Dans l'exemple de notre maquette, nous avons vu

que la requête permettant de sélectionner l'article 9 du règlement concernant la zone UA et le secteur UAa est la suivante en XPath :

```
//zone[@type='UA']/article[@numero='9']/paragraphe[secteur/@type='UAa' or not(secteur)]/texte
```

En XQL, on aurait l'expression suivante :

```
//zone[@type='UA']/article[@numero='9']/paragraphe[secteur/@type='UAa'  $or$  $not$  secteur]
/texte
```

Les différences avec XPath sont minimes. Un processeur XQL, en traitant cette requête, donne le résultat suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
query:
//zone[@type='UA']/article[@numero='9']/paragraphe[secteur/@type='UAa'$or$$not$secteur]/texte
-->
<xql:result xmlns:xql="http://metalab.unc.edu/xql/">
       <texte>9.1. Dans toute la zone UA, les constructions nouvelles à usage artisanal ne
peuvent être autorisées que si leur emprise au sol n'excède pas: 300 m2.</texte>
       <texte>9.2.  Dans le secteur UAa, le coefficient actuel  d'emprise  au sol  des
constructions ne peut être dépassé à l'occasion  de  construction; nouvelles.</texte>
</xql:result>
```

Le problème est que XQL n'effectue aucune mise en forme sur les résultats. Il faut alors traiter ce document avec une transformation XSLT pour enfin avoir un résultat présentable. On comprend alors que l'utilisationde XPath directement avec XSLT est bien plus pratique.

3.4.3 XQuery

XQuery est un langage de requête dérivé de Quilt, qui s'inspire de plusieurs autres langages dont XQL et XPath. Il est en cours de normalisation par le W3C, mais n'en est encore qu'au stade de « Working Draft ».

http://www.w3.org/TR/xquery/ : Working Draft du 20 Décembre 2001 pour XQuery

Il faut noter que le développement de XQuery 1.0 se fait en parallèle avec XPath 2.0. Ces deux langages sont en fait très similaires même s'ils sont développés par des groupes de travail différents. En fait, XQuery offre des fonctionnalités similaires à celles du couple XSLT-XPath, avec en plus un typage fort des données, et sans les règles de XSLT. Les types de données utilisables sont ceux définis par XML Schema, et tous ceux définis par l'utilisateur avec la syntaxe des schémas XML. Ces différences illustrent le fait que XSLT et XQuery sont destinés à deux applications différentes. XSLT sert à transformer des **documents**. XQuery sert à effectuer des requêtes sur des **données**. Examinons tout d'abord un exemple de requête avec XQuery, qui permet de connaître le nombre de zones du règlement :

```
<resultat>
   LET $zones := //zone
   RETURN
      <reglement nbZones={ count($zones) }>
      </reglement>
</resultat>
```

Le résultat sera le document XML suivant :

```
<resultat>
       <reglement nbZones='12'/>
```

```
</resultat>
```

On aurait bien sûr pu obtenir le même résultat avec XSLT. Il est cependant important de remarquer que la syntaxe de XQuery est beaucoup plus proche de celle des langages de requête courants (SQL par exemple). Il est en effet destiné à offrir le même genre de fonctionnalités. Il permet notamment d'interroger plusieurs documents XML, contrairement à XSLT. De plus, il est prévu qu'il permette de mettre à jour ou d'effacer des portions du document XML. Cela ne sera toutefois probablement pas le cas pour XQuery 1.0, mais seulement pour les versions ultérieures. Parallèlement à XQuery, on peut également noter l'existence de XQueryX, un langage de requête offrant les mêmes fonctionnalités que XQuery, mais avec une syntaxe XML.

3.4.4 Avantages et inconvénients

	Avantages	**Inconvénients**
XPath	+ Utilisation avec XSLT + Syntaxe compacte	- Pas de typage des données
XQL	+ Encore souvent utilisé	- N'apporte rien par rapport à XPath - Appelé à disparaître
XQuery	+ Typage des données + Adapté aux requêtes sur des données	- Pas encore de recommandation du W3C

3.5 FEUILLES DE STYLE CSS POUR LA REPRESENTATION GRAPHIQUE DES ZONES ET SECTEURS

3.5.1 Présentation de CSS

Les CSS (Cascading Style Sheets) ont été initialement conçues pour le langage HTML. Elles s'appliquent toutefois aussi bien aux documents XML, surtout en ce qui concerne CSS-2 (datant de 1998). La syntaxe de CSS n'a rien à voir avec XML, mais elle est simple à apprendre et à utiliser. Une feuille CSS se base également sur un ensemble de règles, dont le principe est le suivant : on définit un sélecteur, et on applique des valeurs à certaines de ses propriétés. Par exemple, on peut formater un titre de la manière suivante :

```
titre {
      font-weight : bold ;
      font-size : 14pt ;
      font-family : Garamond ;
      font-style : normal
}
```

En incluant cette feuille à un document XML ou HTML, tous les éléments « titre » seront affichés en gras, avec une taille de police de 14 points, etc.

Le langage SVG permet d'utiliser des feuilles de style CSS ou XSL pour mettre en forme les différents objets utilisés. Cela dit, Adobe SVG Viewer ne permet pas d'utiliser directement XSL. Nous préférerons donc nous concentrer ici sur CSS.

Les objets graphiques de SVG possèdent un attribut « class » qui indique au visualisateur qu'il doit leur appliquer le style correspondant à cette classe. Par exemple, si on attache une feuille de style CSS avec la règle suivante :

```
.UA {stroke:rgb(0,0,0);fill:rgb(166,0,0)}
```

et si le document contient l'objet suivant :

```
<rect class="UA" x="5" y="105" width="20" height="10" />
```

Ce rectangle sera affiché avec une couleur de remplissage rouge sombre et un contour noir. On peut aisément concevoir l'avantage que ces feuilles de style procurent pour créer une légende au document SVG du P.O.S.

3.5.2 Légende automatique avec CSS

Notre carte du P.O.S. est constituée d'un ensemble de zones et de secteurs d'un type donné. C'est ce type qui va déterminer la couleur utilisée pour le remplissage de chaque objet graphique. On peut réaliser cela automatiquement en donnant ce type comme valeur de l'attribut « class » de chaque objet. Ensuite, la feuille de style suivante permet de l'afficher avec la bonne couleur :

```
.NB {fill:rgb(215,200,10)}
.ND {fill:rgb(50,160,45)}
.INA {fill:rgb(250,175,165)}
.IINA {fill:rgb(250,175,165)}
.UD {fill:rgb(255,190,120)}
.UA {fill:rgb(166,0,0)}
.UI {fill:rgb(130,25,135)}
```

```
.UY {fill:rgb(130,25,135)}
.ZI {fill:rgb(130,25,135)}
.UC {fill:rgb(255,125,40)}
.UB {fill:rgb(255,25,0)}
.NC {fill:rgb(155,210,85)}

.NBa {fill:rgb(215,200,10)}
.NBb {fill:rgb(215,200,10)}
.NDa {fill:rgb(50,160,45)}
.NDc {fill:rgb(50,160,45)}
.NDi {fill:rgb(50,160,45)}
.UDa {fill:rgb(255,190,120)}
.UAa {fill:rgb(166,0,0)}
.UIa {fill:rgb(130,25,135)}
.UIb {fill:rgb(130,25,135)}
.UIc {fill:rgb(130,25,135)}
.UIi {fill:rgb(130,25,135)}
.UCa {fill:rgb(255,125,40)}
.UBa {fill:rgb(255,25,0)}
.IINAa {fill:rgb(250,175,165)}
.IINAb {fill:rgb(250,175,165)}
.INAa {fill:rgb(250,175,165)}
.INAb {fill:rgb(250,175,165)}
.INAL {fill:rgb(250,175,165)}
```

L'avantage est double. D'une part, on sépare l'aspect sémantique (le type de la zone) de l'aspect visuel (sa représentation). D'autre part, on peut réutiliser cette feuille de style en l'associant également à la légende de la carte. Les couleurs utilisées seront alors les mêmes. Si on désire ensuite changer la couleur des zones UA par exemple, il suffit de modifier la feuille de style, et les couleurs seront modifiées dans la carte et dans la légende. L'association d'une feuille de style se fait grâce à l'instruction suivante :

```
<?xml-stylesheet href="mapstyle.css" type="text/css"?>
```

La légende sera ensuite constituée de rectangles de la couleur d'une zone et du texte correspondant.

```
<g id="ZonesUI">
<rect class="UI" x="5" y="225" width="20" height="10" onclick="selectZone(evt.target)"/>
<text x="30" y="235" style="font-family:Arial;font-size:14;">UI</text>
</g>
```

La classe du rectangle détermine la couleur de remplissage utilisée.

Le schéma suivant positionne l'utilisation simultanée de CSS et XSL pour la constitution de l'arborescence et de la légende dans la maquette :

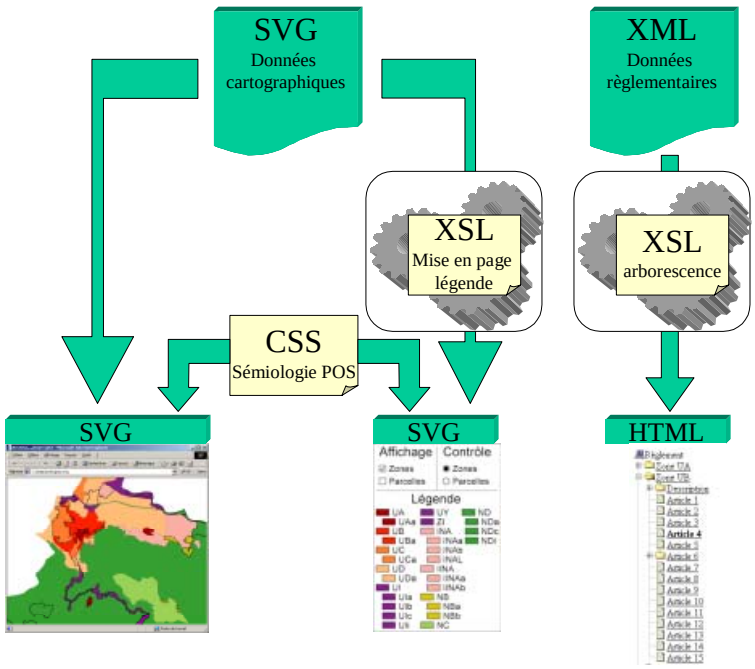


Figure 24 : Utilisation de CSS et XSL dans la maquette

3.5.3 Comparaison entre XSL et CSS

Même si XSL et CSS permettent de créer et d'utiliser des feuilles de style, ces deux langages sont très différents dans leur principe. Les feuilles XSL seront utilisées de préférence lorsqu'on souhaite modifier la structure même du document pour en créer une vue différente. Les feuilles CSS seront utilisées quant à elles lorsque l'on souhaite simplement appliquer un style aux différentes parties du document.

Pour les documents SVG, l'utilisation des CSS pour appliquer un style aux éléments graphiques est recommandée car cette opération est particulièrement simple. Toutefois, il est possible d'effectuer une transformation XSLT sur le serveur qui aboutit au même résultat dans la maquette « client léger ». Nous ne le recommandons pas en général car cela complique inutilement la mise en place de l'application et augmente sensiblement le volume des données à télécharger. Dans certains cas (inclusion d'un fragment de document SVG dans un autre document), il n'est pas possible d'utiliser une feuille de style CSS externe. Il est alors nécessaire d'effectuer une transformation avec XSLT pour appliquer le style désiré. Cela dit, il existe souvent des moyens de contourner ce problème.

	Avantages	**Inconvénients**
CSS	+ Syntaxe simple et compacte + Facilement utilisable avec SVG	- Syntaxe non XML - Permet uniquement des opérations simples
XSL	+ Permet de réaliser des transformations complexes + Syntaxe XML	- Pas d'intégration directe avec Adobe SVG Viewer - Moins facile à utiliser avec SVG

3.6 EDITION AVEC XSL-FO

Une des fonctionnalités de la maquette est de pouvoir éditer un document contenant les informations pour une parcelle donnée. En pratique, ce document est au format PDF (facile à visualiser et à imprimer) et contient un fragment de la carte montrant la parcelle et ses alentours, une légende, et le règlement de la zone et du secteur indiqués par l'utilisateur. Pour le réaliser, nous utilisons le langage XSL-FO qui permet de décrire la mise en page d'un document.

Note : Le format PDF présente de nombreux avantages par rapport à SVG en ce qui concerne la mise en page. Notamment, il permet de paginer un document.

3.6.1 Présentation de XSL-FO

Le langage XSL-FO fait partie de la recommandation du W3C pour XSL du 15 Octobre 2001. Il est donc normalisé et stable.

http://www.w3.org/TR/xsl/ : Recommandation du W3C pour XSL

Il permet de décrire la mise en page d'un document en vue de son édition. Les outils qui traitent des documents XSL-FO permettent de créer des documents PDF, PostScript, etc. En pratique, il n'est jamais utilisé tel quel car sa syntaxe est lourde. L'utilisation la plus courante est de traiter un document XML avec une transformation XSLT afin de produire le document XSL-FO. Il sera ensuite traité par un outil spécialisé pour créer une représentation en PDF ou en PostScript par exemple. Il existe des outils permettant de créer les feuilles de style et d'effectuer ces transformations de manière automatique. Par exemple, XSL Formatter 2.0 permet entre autres de créer la mise en page d'un document XML avec une interface graphique.

http://www.antennahouse.com/axf20/AXF20topEN.htm : XSL Formatter de Antenna House

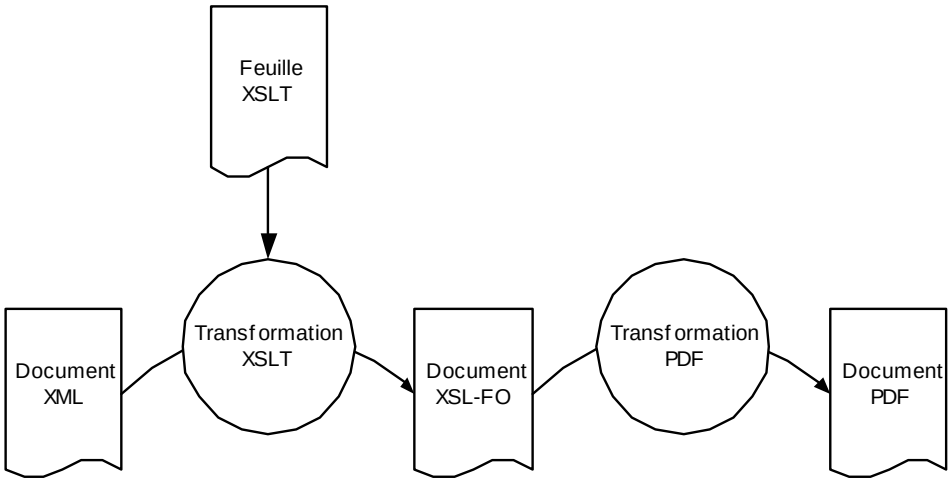


Figure 25 : Principe d'utilisation de XSL-FO

Un document XSL-FO typique est divisé en deux parties. Tout d'abord, l'élément `<fo:layout-master-set>` permet de décrire la mise en page générale des pages (taille des marges, …). Ensuite, l'élément `<fo:page-sequence>` décrit le contenu de ces pages. Ce contenu est découpé en blocs qui correspondent généralement à des paragraphes avec un certain style (police, taille des caractères, couleur, etc.). Ces blocs peuvent bien entendu contenir du texte, mais aussi des éléments externes (images, …) ou des éléments d'un autre type contenus dans le document XSL-FO lui-même (typiquement des graphiques SVG).

3.6.2 Utilisation de XSL-FO pour éditer une fiche parcellaire en format PDF

Dans le cadre de la maquette, nous avons utilisé XSL-FO pour définir la mise en page des éléments graphiques (carte et légende) et du règlement. La fiche parcellaire produite a l'aspect suivant :

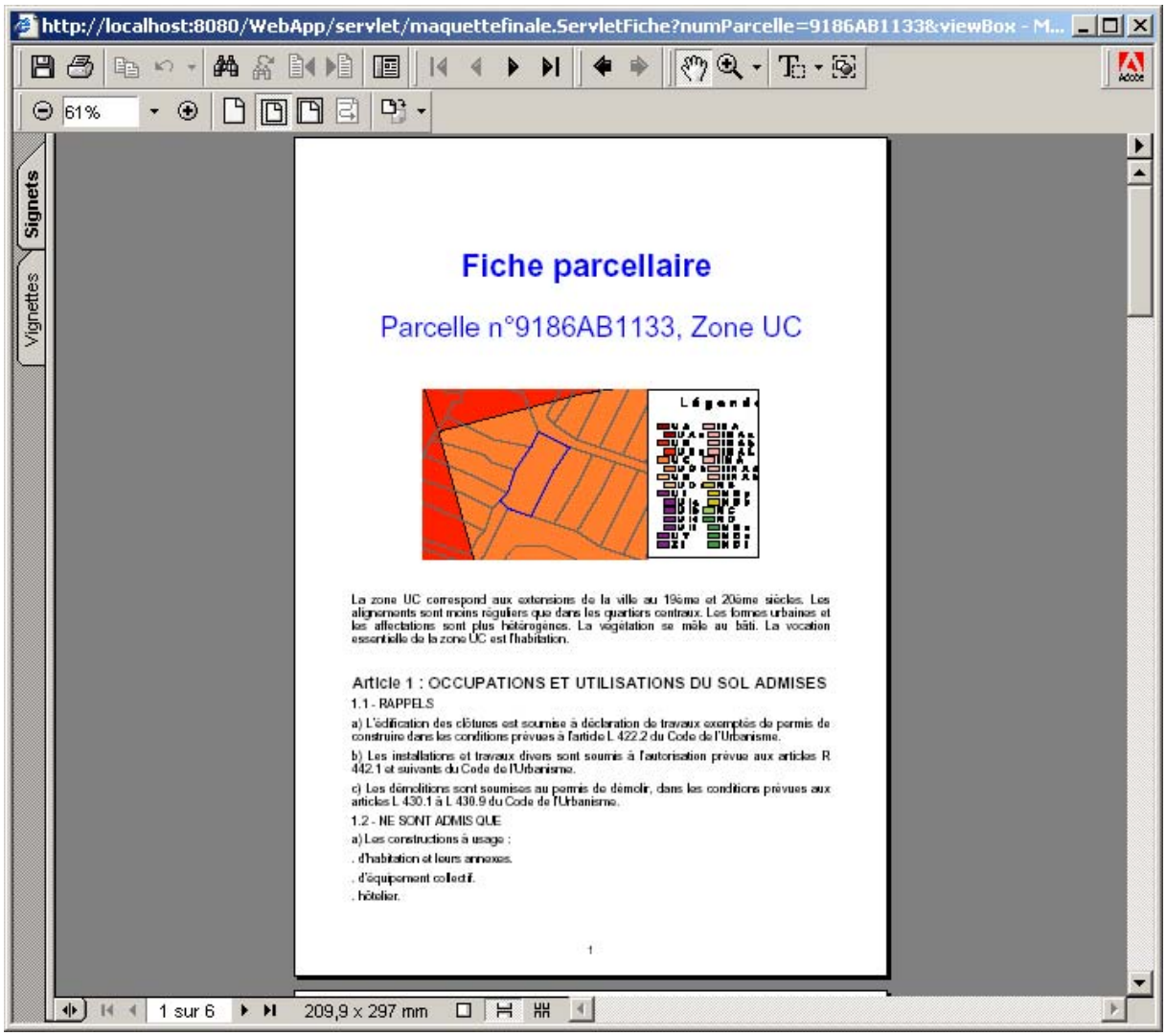


Figure 26 : Première page d'une fiche parcellaire

Le document est produit grâce à trois transformations XSLT. La première crée la vue désirée de la carte à partir du document SVG du POS. La seconde crée la légende à partir du même document. La troisième crée le document FO en incluant ces deux SVG et la partie du règlement demandée. Nous ne détaillerons pas ici les transformations SVG → SVG, mais seulement la génération du document XSL-FO. Pour cela, examinons la transformation XSLT utilisée pour créer celui-ci :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
	<xsl:output method="xml"/>
```

Le document commence bien entendu par la déclaration XML, la balise <xsl:stylesheet> avec les domaines nominaux utilisés, et la balise indiquant que cette transformation produit un document XML.

```
	<xsl:param name="svg"/>
```

```
	<xsl:param name="svgleg"/>
	<xsl:param name="typeZone"/>
	<xsl:param name="typeSecteur"/>
	<xsl:param name="numParcelle"/>
```

Les paramètres « svg » et « svgleg » permettent d'inclure les fragments de document SVG de la carte et de la légende. Les paramètres « typeZone », « typeSecteur » et « numParcelle » permettent d'indiquer la parcelle concernée, ainsi que la zone et le secteur dont on souhaite obtenir le règlement.

```
	<xsl:attribute-set name="default">
		<xsl:attribute name="font-size">12pt</xsl:attribute>
		<xsl:attribute name="font-weight">normal</xsl:attribute>
		<xsl:attribute name="font-style">normal</xsl:attribute>
		<xsl:attribute name="text-align">justify</xsl:attribute>
		<xsl:attribute name="color">black</xsl:attribute>
		<xsl:attribute name="space-before">0.2cm</xsl:attribute>
	</xsl:attribute-set>
	<xsl:attribute-set name="titre">
		<xsl:attribute name="font-size">32pt</xsl:attribute>
		<xsl:attribute name="color">blue</xsl:attribute>
		<xsl:attribute name="font-weight">bold</xsl:attribute>
		<xsl:attribute name="text-align">center</xsl:attribute>
		<xsl:attribute name="space-before">2cm</xsl:attribute>
	</xsl:attribute-set>
	<xsl:attribute-set name="sousTitre">
		<xsl:attribute name="font-size">28pt</xsl:attribute>
		<xsl:attribute name="color">blue</xsl:attribute>
		<xsl:attribute name="font-weight">normal</xsl:attribute>
		<xsl:attribute name="text-align">center</xsl:attribute>
		<xsl:attribute name="space-before">1cm</xsl:attribute>
		<xsl:attribute name="space-after">0.5cm</xsl:attribute>
	</xsl:attribute-set>
	<xsl:attribute-set name="titreArticle">
		<xsl:attribute name="font-size">16pt</xsl:attribute>
		<xsl:attribute name="color">black</xsl:attribute>
		<xsl:attribute name="font-weight">bold</xsl:attribute>
		<xsl:attribute name="space-before">1cm</xsl:attribute>
	</xsl:attribute-set>
```

Ces groupes d'attributs définissent les styles qui seront utilisés ensuite pour créer les différents éléments textuels (titre, corps de texte, …).

```
<xsl:template match="/">
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns="http://www.w3.org/2000/svg">
```

On débute le document FO par la balise <fo:root>, qui comme son nom l'indique est la racine d'un document FO. Les domaines nominaux utilisés sont ceux de FO et de SVG, car les fragments de documents SVG seront inclus directement dans le document FO.

```
<fo:layout-master-set>
	<fo:simple-page-master master-name="contents" page-height="29.7cm" page-width="21cm">
		<fo:region-before extent="1cm"/>
		<fo:region-after extent="1cm"/>
		<fo:region-start extent="1cm"/>
		<fo:region-end extent="1cm"/>
		<fo:region-body margin-bottom="2cm" margin-top="2cm" margin-left="2cm" margin-
right="2cm"/>
```

```
        </fo:simple-page-master>

        <fo:page-sequence-master master-name="contents-Seq">
                <fo:repeatable-page-master-reference master-reference="contents"/>
        </fo:page-sequence-master>
</fo:layout-master-set>
```

Cette section définit la mise en page générale du document, c'est à dire la taille de la page et les marges utilisées.

```
<fo:page-sequence master-reference="contents-Seq">
        <fo:static-content flow-name="xsl-region-before">
        </fo:static-content>
        <fo:static-content flow-name="xsl-region-after">
                <fo:block font-size="10pt" font-family="Helvetica" text-align="center">
                        <fo:page-number/>
                </fo:block>
        </fo:static-content>
        <fo:flow flow-name="xsl-region-body">
                <fo:block xsl:use-attribute-sets="titre">
                        Fiche parcellaire
                </fo:block>
                <fo:block xsl:use-attribute-sets="sousTitre">
                        Parcelle  n°<xsl:value-of  select="$numParcelle"/>,  Zone  <xsl:value-of
select="$typeZone"/>
                        <xsl:if test="$typeSecteur!=$typeZone">
                                , Secteur <xsl:value-of select="$typeSecteur"/>
                        </xsl:if>
                </fo:block>
                <fo:block space-before="1cm" space-after="1cm" text-align="center">
                        <fo:external-graphic>
                                <xsl:attribute name="src"><xsl:value-of select="$svg"/>
                                </xsl:attribute>
                        </fo:external-graphic>
                        <fo:external-graphic>
                                <xsl:attribute name="src"><xsl:value-of select="$svgleg"/>
                                </xsl:attribute>
                        </fo:external-graphic>
                </fo:block>
                <fo:block>
                        <xsl:apply-templates select="//zone[@type=$typeZone]/description"/>
                        <xsl:apply-templates select="//zone[@type=$typeZone]/article"/>
                </fo:block>
        </fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>
```

La section indiquée par la balise <fo:page-sequence> définit le contenu du document. On peut définir un en-tête et un pied de page qui seront répétés sur toutes les pages grâce à la balise <fo:static-content>. On indique ici le numéro en bas de chaque page.

Ensuite, la balise <fo:flow> définit le contenu textuel et graphique du document. Les éléments sont placés dans des blocs (balise <fo:block>) qui correspondent à des paragraphes dont on peut définir le style. Pour cela, on inclut les ensembles d'attributs définis au début du document grâce à la balise <xsl:use-attribute-sets>. C'est un moyen simple d'éviter de réécrire à chaque fois la liste de tous les attributs. On définit ici des blocs pour le titre le sous-titre, les documents SVG externes (référencés grâce à la balise <fo:external-graphic>), puis pour le règlement.

```
<xsl:template match="description">
```

```
	<xsl:apply-templates select="paragraphe[secteur/@type=$typeSecteur or not(secteur)]/texte"/>
</xsl:template>

<xsl:template match="article">
	<fo:block xsl:use-attribute-sets="titreArticle">
		Article <xsl:value-of select="@numero"/> : <xsl:value-of select="@titre"/>
	</fo:block>
	<xsl:apply-templates select="paragraphe[secteur/@type=$typeSecteur or not(secteur)]/texte"/>
</xsl:template>
```

Ces deux règles permettent d'inclure la description d'une zone et les articles du règlement avec leur titre dans le document FO.

```
	<xsl:template match="texte">
		<fo:block xsl:use-attribute-sets="default">
			<xsl:apply-templates/>
		</fo:block>
	</xsl:template>

</xsl:stylesheet>
```

Cette dernière règle transforme les éléments « texte » sélectionnés dans le règlement en blocs avec le style par défaut.

Cet exemple donne un aperçu du langage XSL-FO et de ses possibilités. Nous avons présenté la transformation permettant de produire un document FO plutôt que le document FO lui-même car ce dernier est trop volumineux pour être facilement interprétable. Pour transformer ce document en un document PDF, nous utilisons Apache FOP, un ensemble de bibliothèques Java permettant de traiter les documents XSL-FO. L'utilisation de FOP se fait par l'intermédiaire d'une servlet ressemblant à celles utilisées pour générer l'arborescence ou un article du règlement. Pour traiter les documents SVG, FOP utilise Batik. Certains éléments ne sont pas traités de la même façon qu'avec Adobe SVG Viewer. Notamment nous avons constaté les bugs suivants :

- Dans certains cas les documents SVG inclus causent la production d'un document PDF invalide (cela dépend manifestement de la taille du document dans la page).
- Les polices de caractères ne sont pas toujours représentées correctement.
- Il n'est pas possible de créer le double trait pour les secteurs.
- Il n'est pas possible de créer d'effets de transparence (limitation due au format PDF).

3.7 INTERFACE GRAPHIQUE AVEC SVG ET JAVASCRIPT

Un des principaux avantages du format SVG est de permettre certaines interactions avec l'utilisateur grâce à un langage de script. Ces opérations permettent de modifier dynamiquement la représentation DOM du document SVG. On peut ainsi créer, modifier ou supprimer tous les éléments graphiques du document en temps réel. Cette partie décrit les possibilités que nous avons explorées en ce qui concerne le document SVG et ses interactions avec Javascript.

3.7.1 Généralités

Les différents aspects illustrés dans cette partie fonctionnent bien avec le visualisateur Adobe SVG Viewer et le navigateur Microsoft Internet Explorer. Nous ne pouvons pas garantir que cela sera le cas avec d'autres visualisateurs ou d'autres navigateurs qui supportent plus ou moins bien le Javascript. Les fonctions Javascript sont appelées à partir du visualisateur lorsque l'utilisateur agit sur les objets graphiques (selon les attributs onclick, onmouseover, etc.). Ces fonctions peuvent ensuite manipuler le modèle DOM de n'importe quel objet du document. Le visualisateur interprète automatiquement les modifications et les fait apparaître à l'écran. Les possibilités sont donc théoriquement illimitées. En voici quelques exemples.

3.7.2 Surlignage des contours et effets de transparence sur les zones

Dans la maquette réalisée, lorsque l'utilisateur passe le curseur de la souris au-dessus d'une zone ou d'un secteur, la couleur de celui-ci devient plus pâle. Lorsqu'il sélectionne une parcelle, le contour de celle-ci devient bleu. Et lorsqu'il sélectionne un type de zone ou de secteur, le contour des éléments de ce type deviennent rouges.

Pour réaliser cela, la méthode la plus évidente serait de modifier directement la transparence de l'élément en question, ou la couleur et l'épaisseur de son contour. Cela serait réalisé grâce à ces quelques lignes de code par exemple :

```
function mouseOverPoly (evt) {
	var poly = evt.getTarget();
	poly.style.setProperty('fill-opacity','0.5');
}
```

Malheureusement, cette méthode pose quelques problèmes. Puisque les zones et les secteurs sont superposés, le fait de rendre un secteur transparent ne change rien visuellement (on voit toujours la zone au-dessous qui est de la même couleur). De plus, les contours des zones se recouvrent souvent, il est donc possible que le changement de couleur d'un contour ne soit pas ou peu visible. Pour résoudre ces problèmes, nous utilisons d'autres éléments graphiques « path » qui seront dessinés après tout le reste de la carte, dans une couche que nous appelons couche cosmétique. En pratique, il suffit de mettre les éléments de cette couche à la fin du document SVG, car le dessin des objets se fait dans l'ordre du document. Lorsque l'on veut surligner un objet ou son contour, il suffit de modifier l'attribut « d » de ces éléments cosmétiques pour qu'ils se superposent à l'objet, puis de les rendre visibles.

Par exemple, pour rendre la couleur d'une zone ou d'un secteur plus pâle, nous utilisont un élément « path » de couleur blanche et à moitié transparent. Lorsque le curseur de la souris passe sur une zone ou un secteur, nous superposons cet élément à l'objet en question et nous le rendons visible. Cela est réalisé grâce aux quelques lignes de code suivantes :

```
function mouseOverPoly (evt) {
	var fog = document.svgMap.getSVGDocument().getElementById("fog");
	fog.setAttribute("d", evt.getTarget().getAttribute("d"));
	fog.style.setProperty('visibility', 'visible');
}
```

L'élément « fog » est défini ainsi dans le document SVG :

```
<path    id="fog"    style="stroke:none;fill-opacity:0.5;fill:white;visibility:hidden;pointer-
events:none"/>
```

Lorsque l'on souhaite surligner un contour, on procède de la même manière avec des éléments cosmétiques n'ayant pas de couleur de remplissage, mais uniquement un contour de couleur rouge ou bleue.

3.7.3 Réalisation d'un double trait pour le contour des secteurs

Dans une carte de P.O.S., les secteurs sont représentés avec un contour constitué d'un trait plein à l'extérieur et d'un trait en pointillé à l'intérieur. Il faut noter que le format SVG permet de créer des contours avec tous les types de pointillés imaginables, mais qu'il n'existe pas de manière simple de créer ainsi un double trait.

Pour résoudre ce problème, nous avons recours à un objet nommé « mask », qui permet de limiter la zone où il est possible de dessiner. Le dessin du double contour se fait alors en deux étapes.

1. On dessine un trait en pointillé qui représentera l'intérieur du contour avec le masque approprié.
2. On dessine un trait épais en utilisant un masque pour que seule la partie extérieure du trait soit dessinée.

On obtient ainsi un double trait, avec à l'intérieur des pointillés, et à l'extérieur un trait plein. Cela donne le résultat suivant :

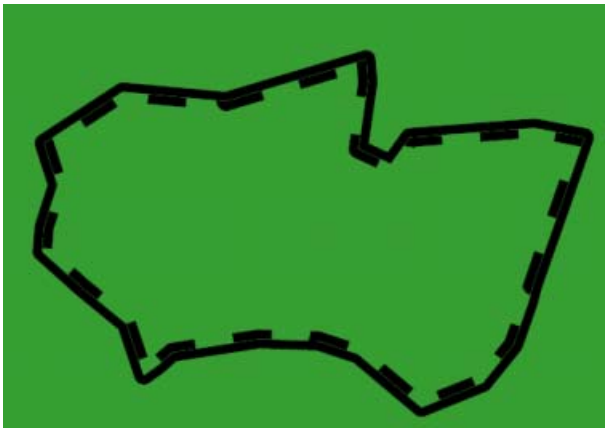


Figure 27 : Exemple de double trait réalisé en SVG

Nous n'utilisons pas de Javascript pour cela, mais uniquement des objets graphiques SVG. Tout d'abord, nous définissons les deux masques :

```
<defs>
<g id="ListeSecteurs">
<use xlink:href="#Secteur1"/>
<use xlink:href="#Secteur2"/>
<use xlink:href="#Secteur3"/>
…
<use xlink:href="#Secteur73"/>
<use xlink:href="#Secteur74"/>
</g>

<mask id="OuterMask" maskUnits="userSpaceOnUse" x="0" y="0" width="8000" height="6000">
  <rect x="0" y="0" width="8000" height="6000" style="fill:white"/>
  <use xlink:href="#ListeSecteurs" style="fill:black;stroke:none"/>
</mask>
```

```
<mask id="InnerMask" maskUnits="userSpaceOnUse" x="0" y="0" width="8000" height="6000">
  <use xlink:href="#ListeSecteurs" style="fill:white;stroke:none"/>
</mask>
</defs>
```

La balise <use> permet de réutiliser l'objet référencé par l'attribut « xlink:href ». Les masques ci-dessus sont définis comme des images en noir et blanc. Lorsqu'ils sont associés à un objet, les parties de l'objet se superposant aux parties blanches seront dessinées, celles qui seront sur les parties noires ne le seront pas. Ce sont donc ici de simples masques binaires, mais il est possible avec SVG de créer des masques plus complexes.

Ensuite, on utilise deux objets graphiques correspondant aux étapes citées ci-dessus pour dessiner tous les contours des secteurs :

```
<use xlink:href="#ListeSecteurs" style="pointer-events:none;stroke-linejoin:round;
stroke:black;stroke-width:10;stroke-dasharray:20,20;fill:none" mask="url(#InnerMask)"/>

<use xlink:href="#ListeSecteurs" style="pointer-events:none;stroke-linejoin:round;
stroke:black;stroke-width:10;fill:none" mask="url(#OuterMask)"/>
```

Il faut noter que s'il est possible de créer ainsi un double trait en utilisant plusieurs objets graphiques différents, il n'est pas possible de créer des triples ou quadruples traits.

3.7.4 Gestion d'une carte miniature

Pour faciliter la navigation sur la carte du P.O.S., nous utilisons une carte miniature indiquant la région actuellement affichée sur la grande carte. Cette carte miniature est un autre document SVG composé simplement de deux objets. Le premier est une image au format GIF représentant l'ensemble de la carte. Le deuxième est un rectangle transparent bleu indiquant quelle partie de la carte est actuellement représentée. On peut déplacer ce rectangle en cliquant dessus et en le faisant glisser jusqu'à l'endroit désiré. Lorsqu'on relâche le bouton de la souris, la vue principale est redessinée en fonction de la nouvelle position du rectangle. Si l'on navigue directement sur la carte en utilisant les fonctions de zoom du visualisateur, la position du rectangle sur la carte miniature est recalculée automatiquement.

Toutes ces opérations sont réalisées grâce aux attributs « currentTranslate » et « currentScale » du document SVG, qui permettent de connaître ou de modifier la position actuelle dans la carte. Pour modifier la position du rectangle, on utilise la fonction suivante qui se déclenche avec les évènements « onzoom » et « onscroll » du document SVG :

```
function changeMinimap(evt) {
	var docelem = document.svgMap.getSVGDocument().getDocumentElement();
	var guide = document.svgMiniMap.getSVGDocument().getElementById("guide");
	guide.setAttribute("x", new String((3000 - docelem.currentTranslate.x*4
/docelem.currentScale)/8400*200));
	guide.setAttribute("y", new String((1200 - docelem.currentTranslate.y*4
/docelem.currentScale)/6300*150));
	guide.setAttribute("width", new String(2400/8400*200/docelem.currentScale));
	guide.setAttribute("height", new String(1800/6300*150/docelem.currentScale));
}
```

On calcule simplement les nouvelles valeurs des attributs « x », « y », « width » et « height » du rectangle en fonction de la nouvelle position de la carte indiquée par les variables « currentTranslate » et « currentScale ».

3.7.5 Zoom sur une parcelle

Une des fonctionnalités de la maquette est de pouvoir sélectionner une parcelle par son numéro et de zoomer sur celle-ci. Il faut noter que nous atteignons ici une des limites du visualisateur Adobe SVG Viewer. En effet, les possibilités de zoom avec celui-ci sont limitées. Les parcelles étant souvent très petites par rapport au reste de la carte, il n'est pas possible de zoomer sur celles-ci en manipulant simplement les variables « currentScale » et « currentTranslate » comme précédemment.

Nous sommes donc obligés de changer la « viewBox » de la carte, que nous calculons à partir des coordonnées de la parcelle sur laquelle nous souhaitons zoomer. Nous avons donc ajouté un bouton permettant de revenir à la vue normale. Nous avons également supprimé les possibilités de navigation et de repérage sur la carte miniature qui seraient devenues trop complexes à gérer dans ce cas. Cette fonction de zoom sur une parcelle se déroule alors en trois étapes :

1. Recherche de la parcelle et affichage d'un message d'erreur si elle n'existe pas.
2. Calcul du rectangle englobant.
3. Changement de la « viewBox » pour afficher la parcelle et ses alentours.

Celles-ci sont réalisées avec la fonction Javascript suivante :

```
function zoomParcelle(text) {
	var doc = document.svgMap.getSVGDocument();
	var parc = doc.getElementById(text);
	if (parc==null) {
		alert("Cette parcelle n'existe pas");
	} else {
		var minX=1000000;
		var minY=1000000;
		var maxX=0;
		var maxY=0;

		var path = parc.getAttribute("d");

		searchValue=new RegExp("M|L|Z|\n|\r","g");
		searchValue.global = true;
		path = path.replace(searchValue, "");

		searchValue=new RegExp(" +","g");
		searchValue.global = true;
		path = path.replace(searchValue, " ");

		var coords = path.split(" ");
		for (var i=0;i<coords.length;i=i+2) {
			var x = new Number();
			var y = new Number();
			x = eval(coords[i]);
			y = eval(coords[i+1]);
			if (x<minX) minX=x;
			if (x>maxX) maxX=x;
			if (y<minY) minY=y;
			if (y>maxY) maxY=y;
		}

		var width = maxX-minX;
		var height = maxY-minY;

		doc.getDocumentElement().setCurrentScale(1);
		doc.getDocumentElement().currentTranslate.x=0;
		doc.getDocumentElement().currentTranslate.y=0;
		doc.getDocumentElement().setAttribute("viewBox", "" + (minX-width/2) + " " +
(minY-height/2) + " " + (width*2) + " " + (height*2));
```

```
                var outerStroke = doc.getElementById("OuterStroke");
                outerStroke.style.setProperty('stroke-width', '0.2');
                var innerStroke = doc.getElementById("InnerStroke");
                innerStroke.style.setProperty('stroke-width', '0.2');
                innerStroke.style.setProperty('stroke-dasharray', '0.4,0.4');

                var bluePath = doc.getElementById("bluepath");
                bluePath.setAttribute("d", parc.getAttribute("d"));
                bluePath.style.setProperty('visibility', 'visible');
                var guide = document.svgMiniMap.getSVGDocument().getElementById("guide");
                guide.style.setProperty("visibility", "hidden");
        }
}
```

3.8 LIENS AVEC XLINK

XLink est un mécanisme permettant de créer et de décrire des liens entre des ressources. Ces liens peuvent être des simples liens hypertexte comme en HTML, ou plus complexes. Nous avons déjà vu un exemple d'utilisation de XLink à la section 3.7.3 pour la réalisation d'un double trait comme contour de secteur géographique. Le but était d'appliquer plusieurs styles différents à un même chemin sans réécrire la définition de ce chemin. Dans ce cas, on crée à chaque fois un lien avec l'objet graphique qui définit le chemin en utilisant XLink. C'est une utilisation typique avec le langage SVG, mais XLink peut être utilisé dans bien d'autres cas.

Une des applications les plus intéressantes de XLink est de décrire des liens hypertextes plus complexes que ce qui est possible aujourd'hui avec l'élément HTML <A>. Ce dernier permet uniquement de définir un lien entre deux ressources, et ce lien doit être décrit dans l'une de celles-ci. XLink permet de définir des liens entre plus de deux ressources et ceux-ci peuvent être décrits dans un autre document (ce qui permet de séparer le contenu d'un document et ses liens avec d'autres documents).

XLink fait l'objet d'une recommandation du W3C depuis le 27 Juin 2001. Celle-ci est consultable à l'adresse suivante :

http://www.w3.org/TR/xlink/ : Recommandation du W3C pour XLink

Le lien vers le document Word du règlement est réalisé dans le SVG de la légende grâce à XLink. Le code qui permet cela se présente ainsi :

```
<a xlink:href="reglement.doc" target="docwindow">
	<text id="liendoc" x="30" y="295" style="font-family:Arial;font-size:14;">
		Règlement Word (.doc)
	</text>
</a>
```

4. OUTILS DE MANIPULATION DE DONNEES XML ET SVG

Cette partie présente les outils existants pour éditer, convertir et visualiser des fichiers XML ou SVG.

4.1 EDITION DE DOCUMENTS XML

4.1.1 Microsoft XML Notepad

C'est un éditeur XML très simple offrant uniquement les fonctionnalités de base. Il permet donc de créer des éléments, des attributs, du texte. Il donne une représentation arborescente du document XML et permet de naviguer dans celle-ci. On peut tout de même noter qu'il permet de valider un document XML avec une DTD. Il est téléchargeable gratuitement à l'adresse suivante :

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/xmlpaddownload.asp

Voici un aperçu de l'édition d'un document avec XML Notepad :

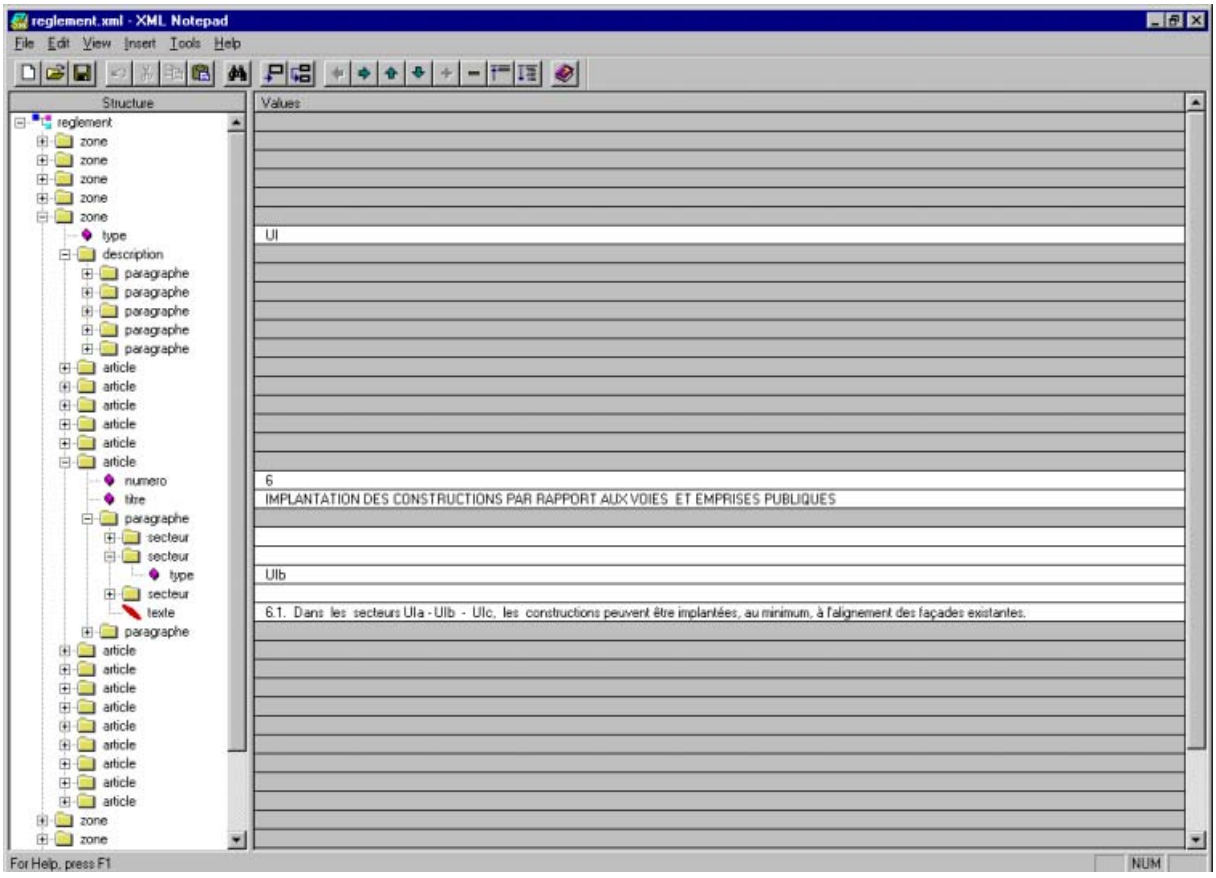


Figure 28 : Microsoft XML Notepad

4.1.2 eXcelon Stylus Studio

Stylus Studio, de la corporation eXcelon, est un éditeur XML nettement plus évolué. Il permet de visualiser et de modifier un document XML sous forme de texte, d'arbre ou de grille. Il est validant, c'est à dire qu'il permet de déterminer si un document XML est valide selon la DTD ou le schéma associé. Il permet également de générer automatiquement un schéma ou une DTD à partir d'un document. Il offre ensuite la possibilité de compléter automatiquement les noms d'éléments, les attributs et les balises fermantes. De plus, il offre un support de XPath pour effectuer des requêtes sur le document. Enfin, il permet de construire graphiquement une feuille de style XSLT.

C'est cet éditeur que nous avons utilisé pour créer le document XML contenant le règlement. Il présente l'avantage d'être simple et clair. Une version d'évaluation est disponible à l'adresse suivante :

http://www.stylusstudio.com/download.asp

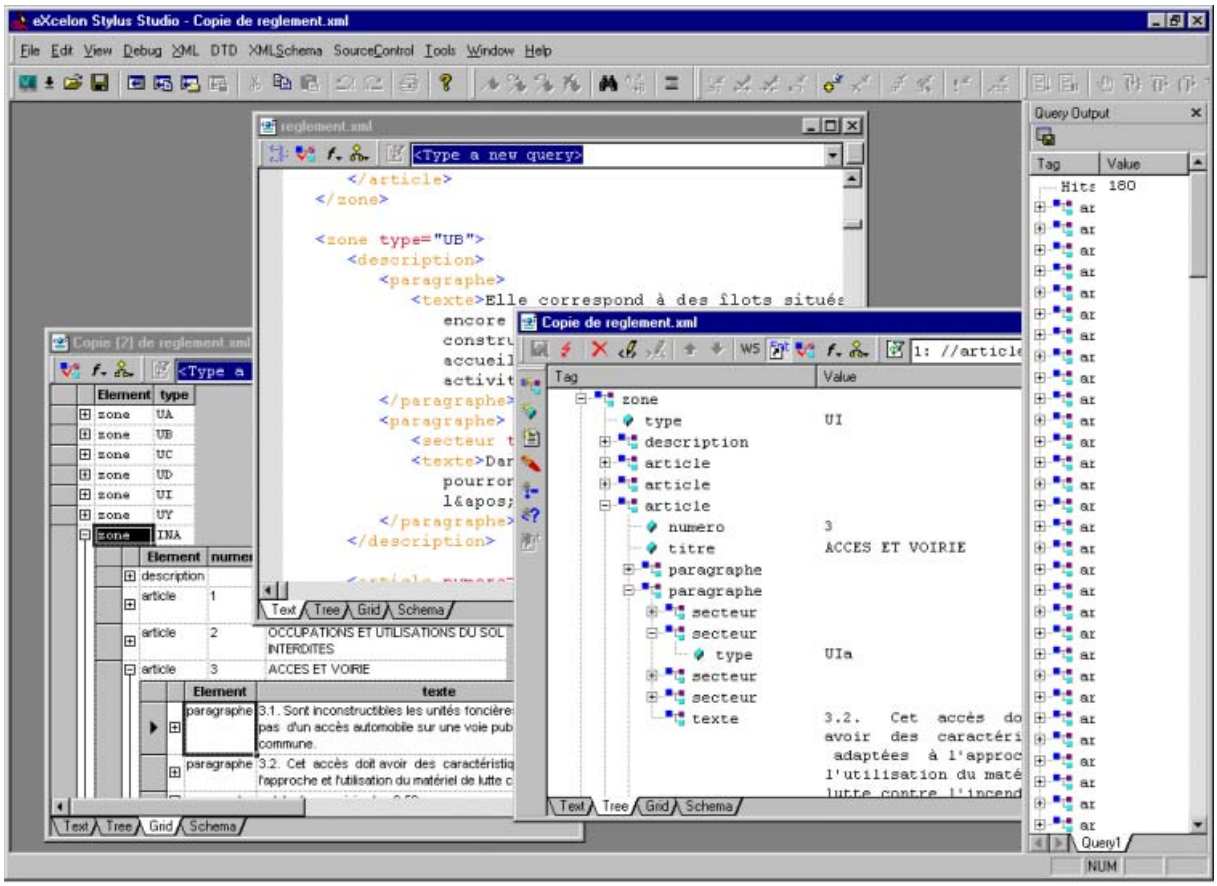


Figure 29 : eXcelon Stylus Studio

4.1.3 XMetal de SoftQuad

XMetal est un éditeur XML qui se rapproche d'un traitement de texte. Il permet de visualiser et d'éditer le document sous forme de texte simple ou d'arbre. Il est validant. Il permet surtout d'associer au document une feuille de style CSS qui permet de mettre en forme les différents éléments. On peut alors utiliser XMetal comme un traitement de texte en sélectionnant le style des différents paragraphes. De plus, il est possible de créer des macros pour personnaliser l'utilisation du logiciel. Il est par exemple possible de créer une macro pour importer des document depuis Microsoft Word, ce qui peut s'avérer très pratique. XMetal est à la fois facile à utiliser et

très puissant lorsqu'on souhaite réaliser des fonctions plus complexes. Une version d'évaluation peut être téléchargée à l'adresse suivante :

http://www.softquad.com/top_frame.sq?page=products/xmetal/content_xmetal_intro.html

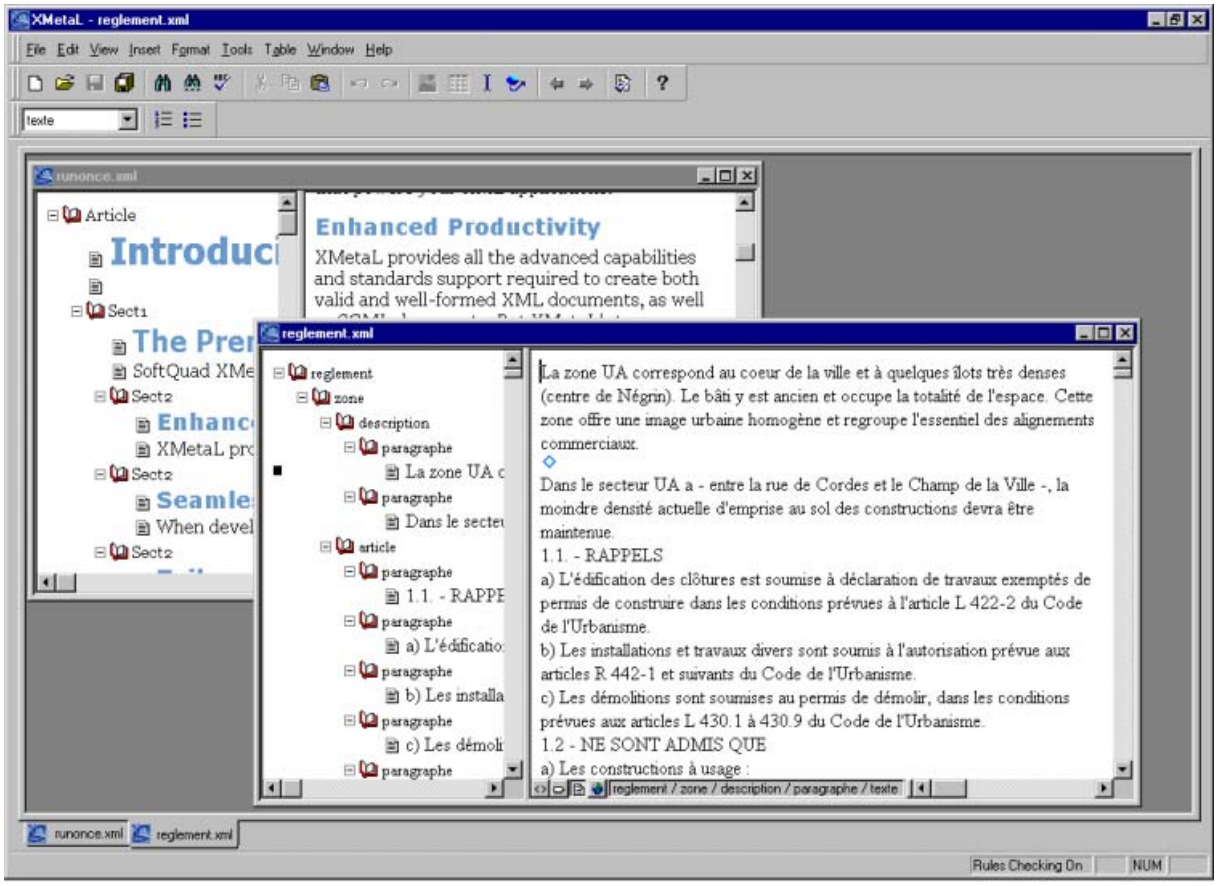


Figure 30 : XMetal

4.1.4 Arbortext Epic

Epic offre dans l'ensemble les mêmes possibilités que XMetal, sauf qu'il utilise des feuilles de style XSL et qu'il ne propose pas de fonctions de macros. Il est cependant moins simple à utiliser au premier abord.

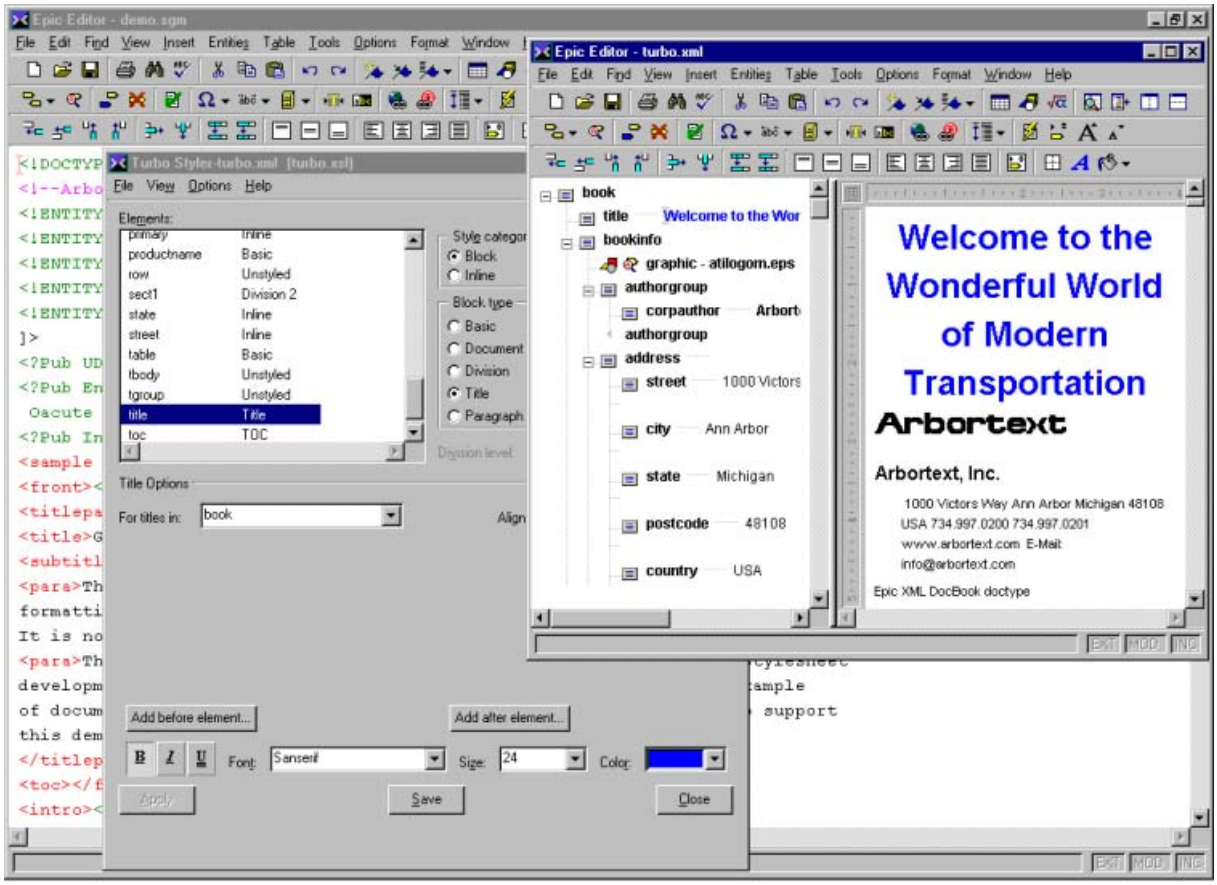


Figure 31 : Epic Editor

4.1.5 Comparatif et préconisations

	Avantages	Inconvénients
Microsoft XML Notepad	+ Simplicité + Gratuit	- Peu de fonctionnalités
eXcelon Stylus Studio	+ Possibilité de tout faire graphiquement + Validation des documents	- Pas de macros - Pas de possibilité de style
Softquad XMetal	+ Simplicité + Possibilités de macros + Feuilles de style CSS	- Pas de validation selon un schéma
Arbortext Epic	+ Feuilles de style XSL	- Complexité - Pas de macros

Pour résumer, on peut distinguer trois catégories d'éditeurs XML. Tout d'abord, les logiciels comme XML Notepad offrent assez peu de fonctionnalités mais présentent l'avantage d'être gratuits. Ensuite un logiciel comme eXcelon Stylus Studio peut être utilisé aussi bien dans une approche **document** XML que dans une approche **données** XML. Son support des schémas XML est en effet un atout important lorsqu'on souhaite ensuite utiliser les données pour effectuer des traitements. Enfin, il existe des traitements de texte XML comme Epic ou XMetal qui sont clairement plus adaptés pour la rédaction de documents, mais qui ne supportent pas les schémas.

Dans le cadre de l'édition d'un document de règlement, un logiciel comme Stylus Studio est bien adapté, mais on peut envisager d'autres alternatives. Par exemple, on peut écrire une macro permettant d'importer le document Word du règlement sous XMetal. Le problème qui se pose alors est de savoir si ce document présente toujours la même structure. Malheureusement, il n'existe pas de règle définie pour la rédaction du règlement du P.O.S. On ne peut donc pas garantir qu'une fonction d'import puisse fonctionner dans tous les cas.

On peut également envisager le développement d'un éditeur spécifique pour le règlement du P.O.S. L'avantage serait d'avoir un outil simple et fiable, pour lequel on peut développer des fonctionnalités spécifiques. Toutefois, il faut noter qu'un tel développement a un coût bien plus élevé que la license d'un produit déjà existant.

Notre préconisation est donc d'utiliser plutôt un logiciel tel que Stylus Studio si l'objectif est simplement de créer une version XML du règlement. Si des fonctionnalités plus spécifiques sont souhaitées, XMetal est plus adapté car il est facilement adaptable en créant des macros supplémentaires (export du document dans un autre format par exemple).

4.2 VISUALISATION DE DOCUMENTS SVG

Cette partie présente certains des outils existants pour visualiser des documents SVG. Une liste plus complète des éditeurs, convertisseurs et visualisateurs SVG est disponible à l'adresse suivante :

http://www.w3.org/Graphics/SVG/SVG-Implementations

4.2.1 Adobe SVG Viewer

C'est le visualisateur le plus utilisé aujourd'hui. Il est disponible sous forme de plug-in, ce qui permet de l'utiliser avec n'importe quel navigateur web. Il est stable et supporte presque toutes les spécifications de SVG 1.0, ce qui en fait pour nous le visualisateur de référence. Il est téléchargeable sur le site d'Adobe :

http://www.adobe.com/svg/viewer/install/main.html

4.2.2 CSIRO SVG Toolkit

C'est un visualisateur en Java qui permet notamment de convertir un document SVG en une image JPEG. Par contre, il ne supporte pas toutes les spécifications de SVG (notamment les CSS), et ses possibilités et performances sont très limitées. On peut le trouver à l'adresse suivante :

http://sis.cmis.csiro.au/svg/

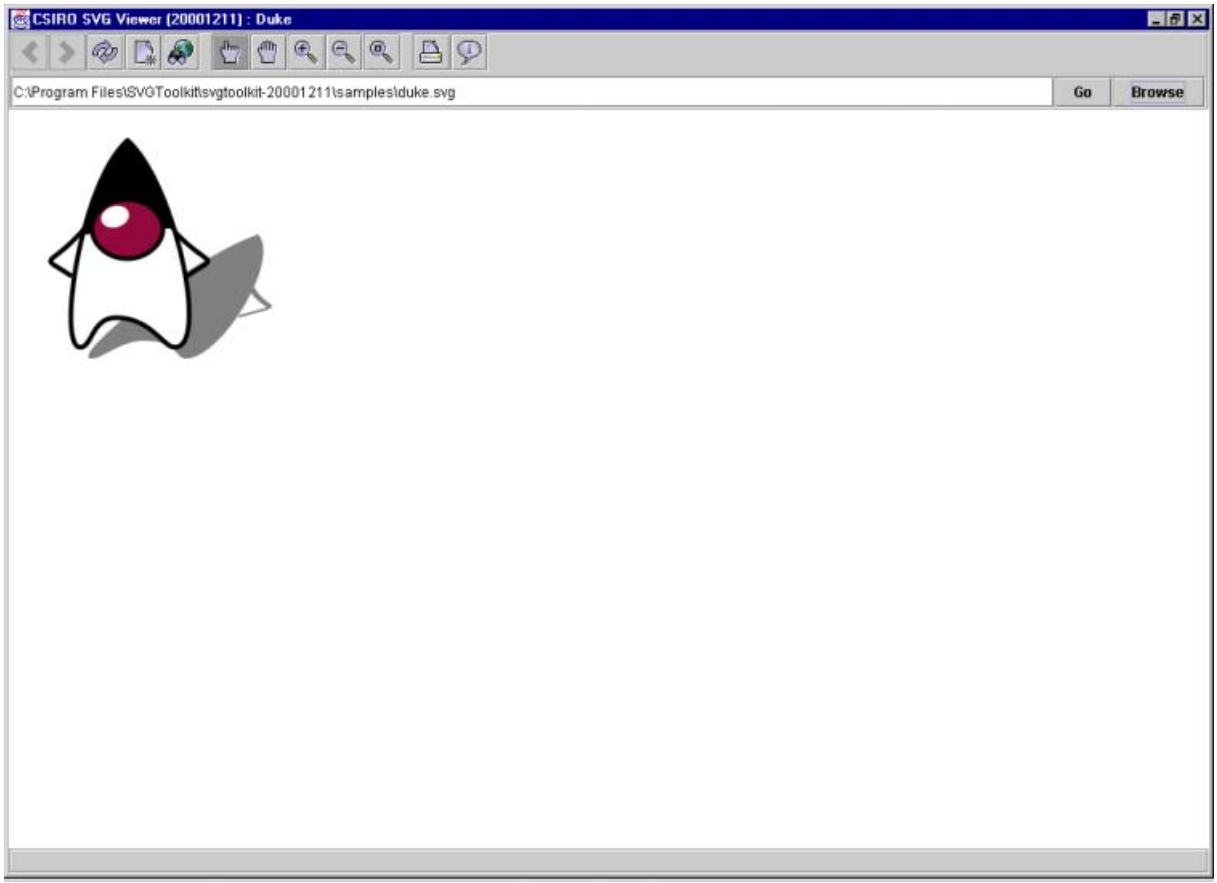


Figure 32 : CSIRO SVG Toolkit

4.2.3 IBM SVGView

C'est un autre visualisateur en Java offrant à peu près les mêmes fonctionnalités que celui de CSIRO. Il ne supporte pas non plus les CSS. Ses performances sont décevantes par rapport aux outils plus récents. On peut le télécharger à l'adresse suivante :

http://www.alphaworks.ibm.com/aw.nsf/download/svgview

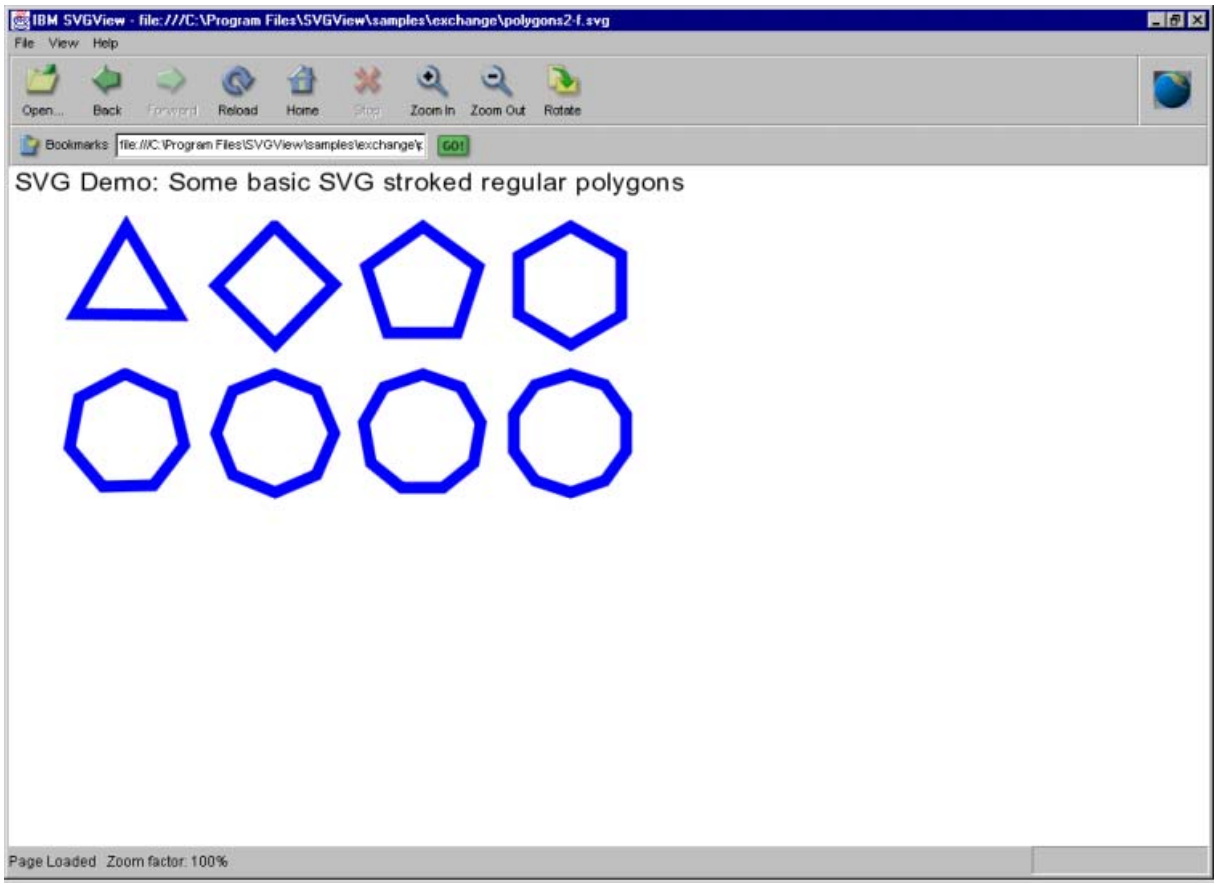


Figure 33 : IBM SVGView

4.2.4 Apache Batik SVG Toolkit

Batik est une librairie Java offrant des fonctions pour afficher, transformer et manipuler des documents SVG. Elle est facilement incorporable à n'importe quelle application Java. Elle est utilisée par plusieurs logiciels pour la gestion des SVG. On peut citer notamment Apache FOP, ILOG's Jviews et Oracle Corp.'s JDeveloper9i. Elle offre un très bon support des spécifications de SVG et de bonnes performances. De plus, elle est fournie avec des outils permettant de visualiser et convertir facilement des documents SVG. On peut la télécharger gratuitement à l'adresse suivante :

http://xml.apache.org/batik/dist/

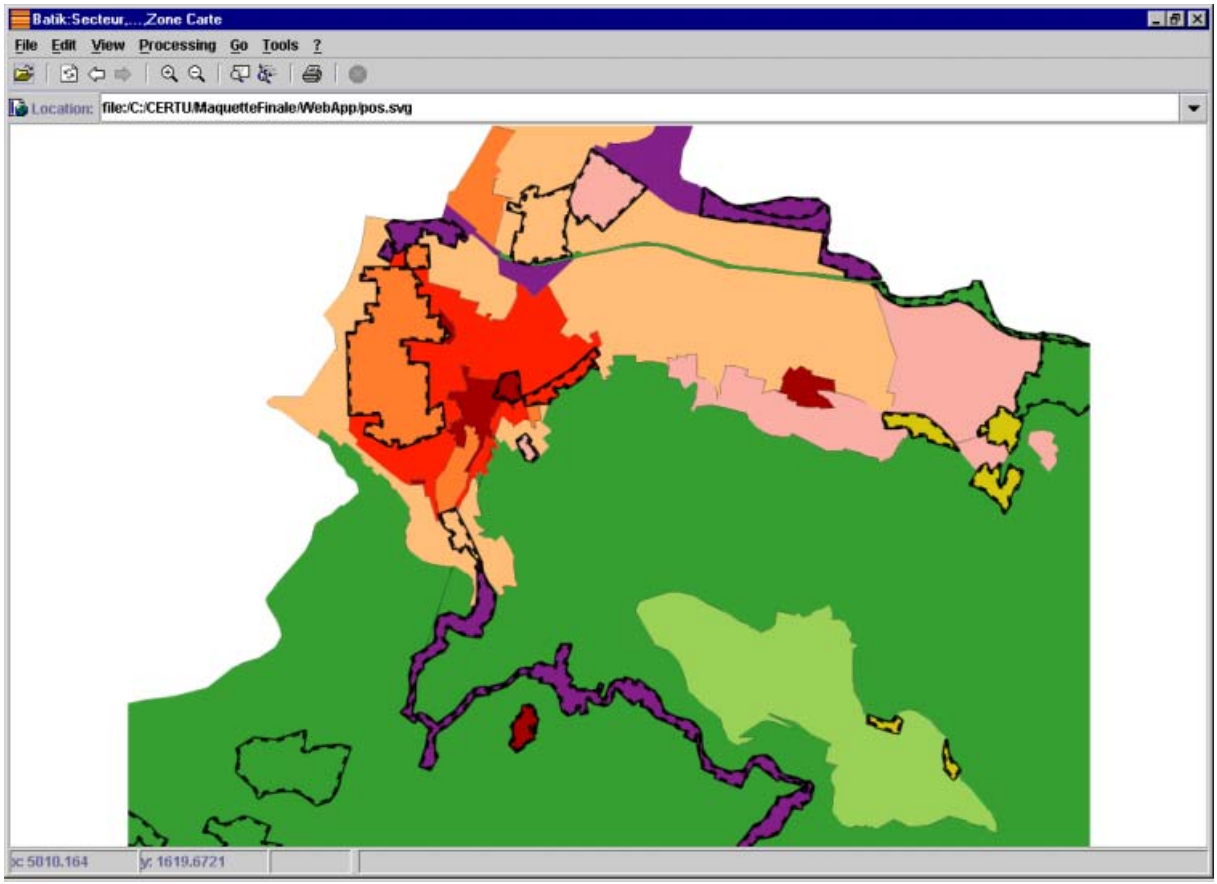


Figure 34 : Batik SVG Browser

4.2.5 Préconisations

En matière de visualisation SVG, le choix d'un outil approprié est très simple. Si l'on souhaite visualiser un SVG sur une page Web, il faut utiliser Adobe SVG Viewer. Si on cherche à intégrer un composant SVG à une application Java ou à visualiser un document indépendant, la librairie Batik d'Apache est la plus appropriée. Les autres outils existant à ce jour ne sont pas intéressants en comparaison.

4.3 GENERATION DE SVG

4.3.1 Jasc WebDraw

WebDraw est un éditeur de documents SVG. Il permet de créer tous les objets graphiques de base, des animations et de modifier leurs propriétés avec une interface graphique. Il est relativement complet mais ne supporte pas les CSS. Il est surtout destiné, comme son nom l'indique, à la création de contenu graphique pour le web. Il est possible de télécharger une version d'évaluation à l'adresse suivante :

http://www.jasc.com/download_4.asp?prod=WP1

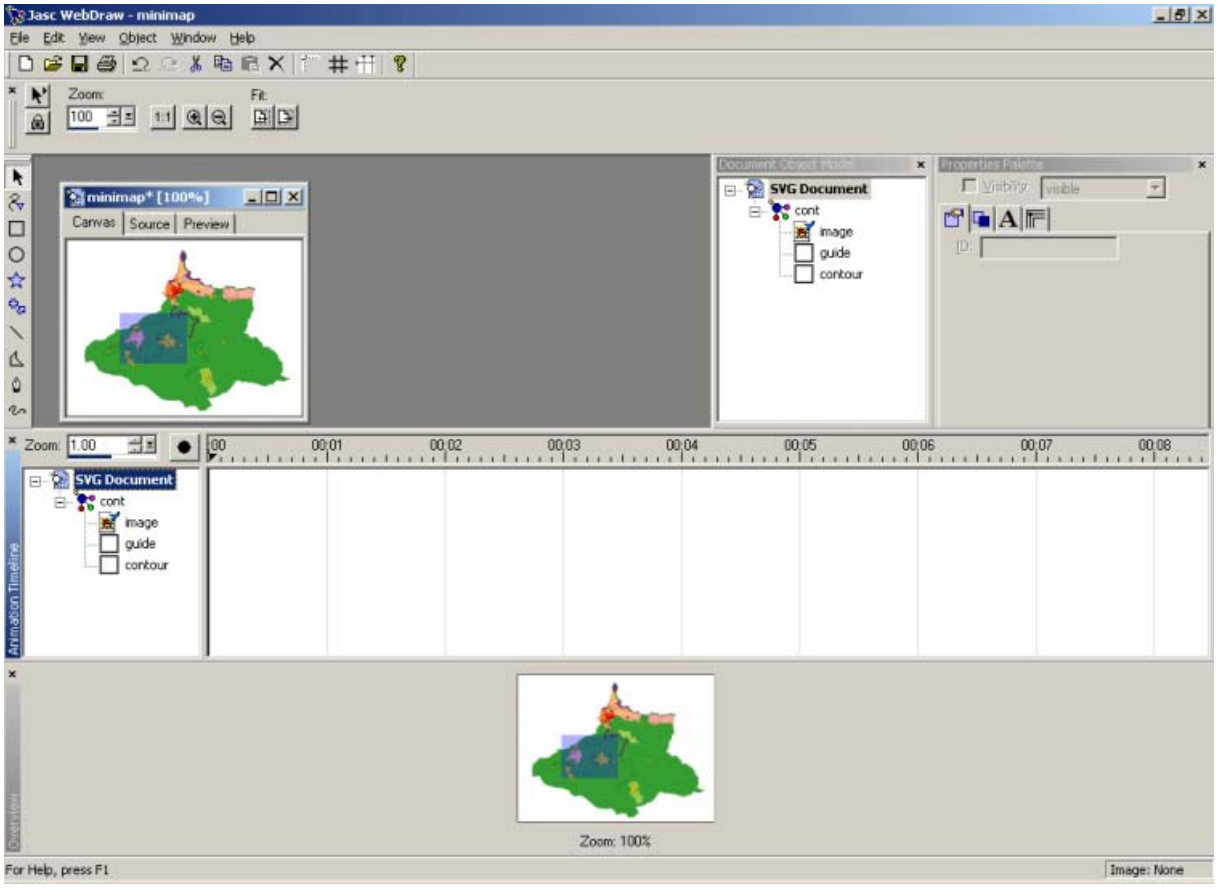


Figure 35 : Jasc WebDraw

4.3.2 W3C Amaya

Amaya est un outil du W3C servant de démonstrateur pour les nouvelles technologies Web. Il supporte actuellement HTML, XHTML, MathML, SVG, CSS et HTTP. En ce qui concerne SVG, les fonctionnalités offertes par Amaya sont limitées. Il se présente comme un éditeur graphique simple à utiliser. On peut le télécharger gratuitement à l'adresse suivante :

http://www.w3.org/Amaya/User/BinDist.html

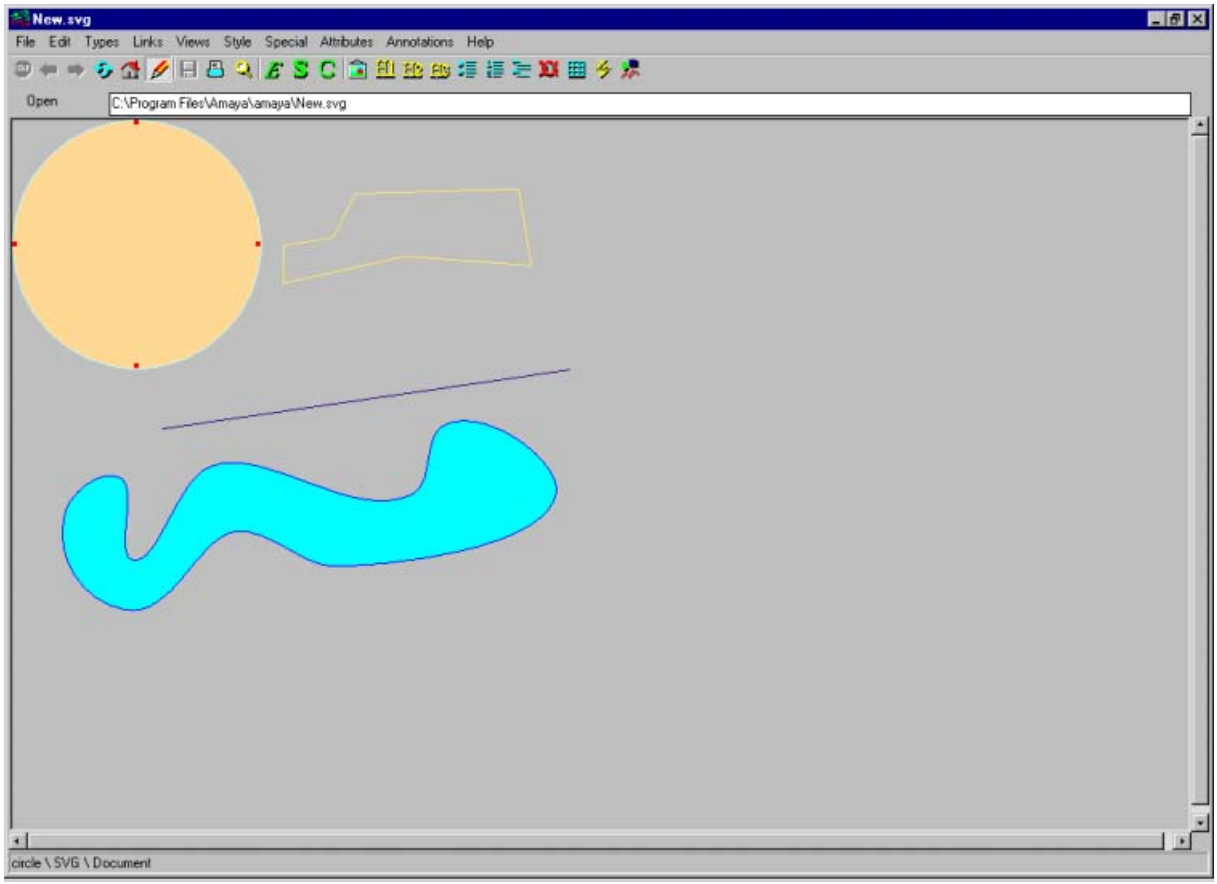


Figure 36 : W3C Amaya

4.3.3 Conversion MapInfo → SVG

Les éditeurs de SVG peuvent être utiles pour créer des contenus graphiques sur le Web, mais le problème qui se pose à nous dans le cadre de la maquette est plutôt de convertir des données géographiques en format SVG. On s'intéresse principalement à la conversion de fichiers TAB, format propriétaire du logiciel MapInfo. Il existe deux utilitaires permettant d'effectuer cette conversion, SVGMapMaker et Map2SVG.

Ces deux utilitaires s'intègrent à MapInfo et permettent d'exporter une carte au format SVG. Avec Map2SVG, seules les données géographiques sont exportées. Il est impossible de récupérer les attributs associés, alors que ceux-ci sont indispensables pour la maquette réalisée. Son utilisation n'est donc pas préconisée dans ce cas.

SVGMapMaker offre quelques possibilités supplémentaires, comme le réglage du niveau de détail qui peut s'avérer utile si on cherche une meilleure précision dans la reproduction des zones. De plus, il permet d'exporter également les données attributaires, mais seulement dans un tableau Javascript. C'est celui-ci que nous avons utilisé pour récupérer les données nécessaires à la maquette.

Il faut toutefois noter qu'un grand nombre de changements ont du être apportés aux fichiers générés afin de les rendre exploitables comme nous le souhaitions. En particulier, il a fallu transférer les données des tableaux Javascript dans le SVG (données sur les types de zone et de secteur), réorganiser les objets graphiques par type et modifier la feuille de style CSS. Ces opérations sont longues et répétitives. De plus, les coordonnées des points ne sont pas retranscrites directement. Il y a une perte de précision non négligeable et il est impossible de savoir dans l'absolu quelles sont les coordonnées géographiques d'une zone. Si l'on veut généraliser la publication sur le web de cartes de P.O.S. (ou de cartes semblables), il sera donc indispensable de développer un outil spécifique.

Voici un exemple pour illustrer les pertes de précision lors de la conversion avec SVGMapMaker. Nous avons créé un document MapInfo simple contenant un très long trait et une croix au milieu.

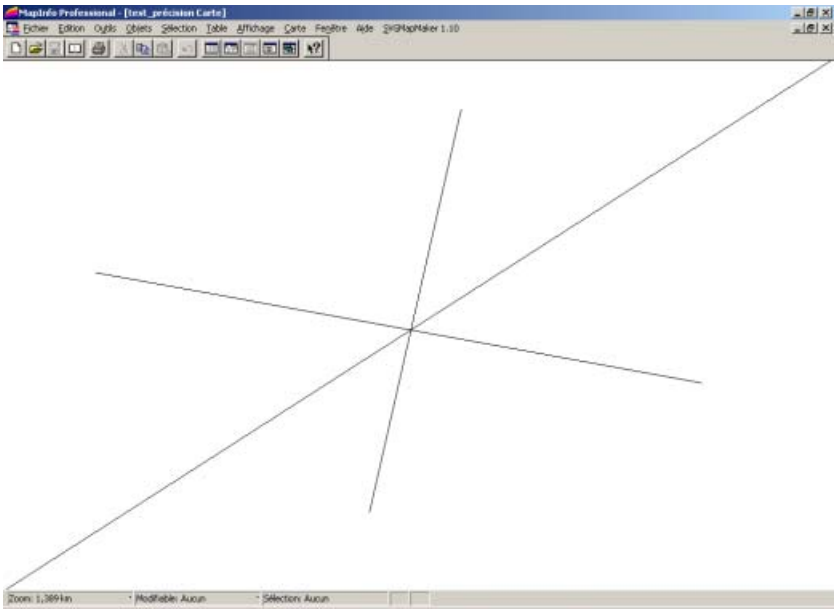


Figure 37 : Test de précision avec MapInfo

Après conversion avec SVGMapMaker au niveau de précision le moins élevé, voici ce que nous obtenons :

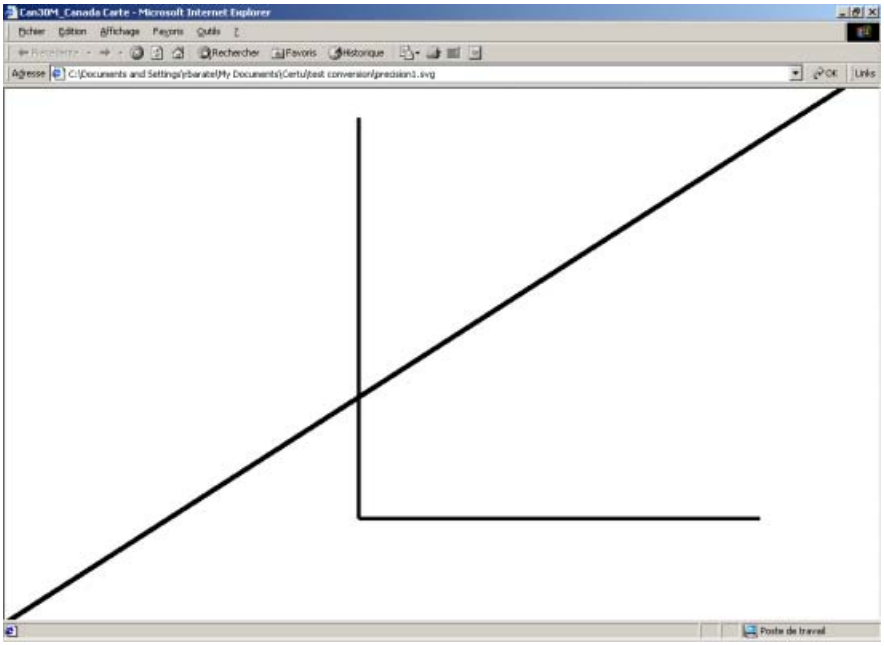


Figure 38 : Conversion avec le niveau de précision 1

Voici le même exemple avec un niveau de précision intermédiaire :

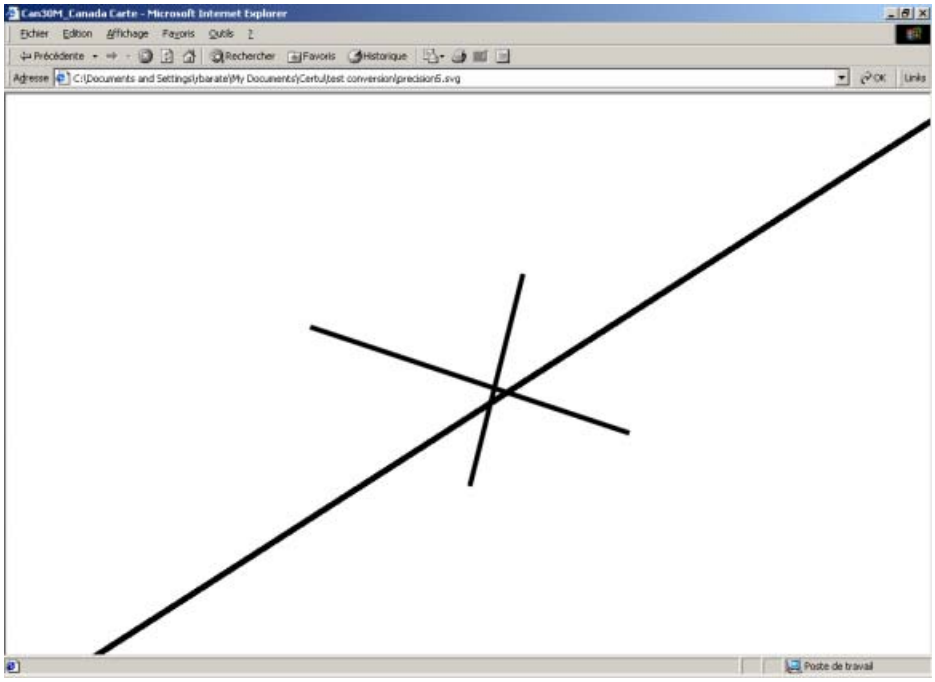


Figure 39 : Conversion avec le niveau de précision 5

Et enfin, le même exemple avec le meilleur niveau de précision :

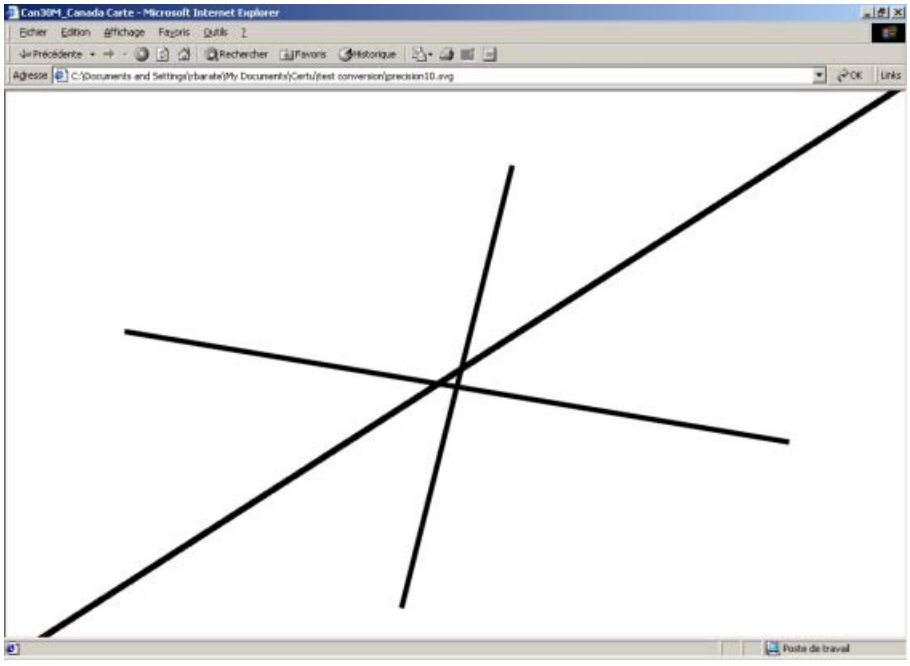


Figure 40 : Conversion avec le niveau de précision 10

On arrive ainsi à une bonne précision, mais la conversion n'est pas pour autant parfaite. Pour cela, il serait nécessaire de développer un outil spécifique ou d'adapter ceux existants.

4.3.4 Conversion DXF → SVG

Dans certains cas, les données géographiques que l'on souhaite publier ne sont pas au format TAB mais au format DXF (format d'AutoCAD). Il existe également des outils pour convertir ces données en SVG. Nous avons utilisé la bibliothèque de Gardos Software nommée gsDXF2SVG.dll, téléchargeable gratuitement à l'adresse suivante :

http://business.hol.gr/gardos/download.htm

Il faut noter qu'elle fonctionne bien avec les données au format DXF R13 ou plus ancien. Pour les données plus récentes, il faut tout d'abord les transcrire en DXF R13. Cela se fait simplement sous AutoCAD avec la commande « Enregistrer sous » et en choisissant le format « DXF R13 ». Contrairement aux conversions depuis MapInfo, cet utilitaire transcrit directement les coordonnées des points avec une bonne précision. Il y a donc peu de perte d'information. Le problème se situe plutôt au niveau du format DXF lui-même, qui est un format de dessin et non pas de description de données géographiques. Il ne permet pas de définir des zones et de leur associer un type, mais seulement de tracer des traits et écrire du texte. Le résultat en SVG sera donc un ensemble de lignes et de textes, ce qui n'est pas suffisant pour notre maquette. Il faut donc réaliser un travail supplémentaire très important pour définir des zones et des secteurs et leur associer un type. Ce travail est de plus impossible ou très difficile à automatiser. C'est le principal obstacle à la publication sur le web de cartes au format DXF. Voici un exemple de SVG produit par gsDXF2SVG pour un document de POS :

```
<text x="634752.9" y="228961.5"
 style="font-family:Verdana; font-size:40; fill:black">
&#x004E;&#x0043;a
</text>
<text x="635602.9" y="229932.2"
 style="font-family:Verdana; font-size:40; fill:lime">
&#x004E;&#x0044;
</text>
<polyline points="635241.1,229335.2 635304.8,229346 635318.6,229374.4
                  635324.4,229369.5 635354.8,229379.3 635379.4,229374.4
                  635514.7,229391.1 635562.8,229420.5 635489.2,229567.6
                  635432.3,229513.7 635393.1,229482.3 635335.2,229454.8
                  635295,229401.9 635257.7,229359.7 635241.1,229335.2"
 style="fill:none; stroke:lime; stroke-width:8" />
<text x="635380.2" y="229463.1"
 style="font-family:Verdana; font-size:40; fill:lime">
&#x004E;&#x0044;
</text>
<polyline points="636176.1,230210.1 636187.2,230221.3 636235.3,230335.1
                  636230.6,230367.5 636247.3,230532.1 636309.3,230536.8
                  636338,230563.6 636368.1,230613.2"
 style="fill:none; stroke:lime; stroke-width:8" />
<polyline points="636101.4,230122.8 636229.4,230364.9 636239.3,230573
                  636328.5,230597.8 636417.8,230731.5 636526.8,230677
                  636616,230662.2 636655.6,230573 636779.6,230528.4
                  636893.6,230538.3 636918.4,230548.2"
 style="fill:none; stroke:black; stroke-width:11.07942" />
<polyline points="636795.4,231261.7 636794.3,231239.6 636782.5,231218.4
                  636673,231129.4 636673,231115.2 636646.9,231079.8
                  636650.1,231067.9 636621.8,231030.1"
 style="fill:none; stroke:lime; stroke-width:8" />
<text x="636123.3" y="230949.1"
 style="font-family:Verdana; font-size:40; fill:lime">
&#x004E;&#x0043;
</text>
```

Les polygones sont en fait représentés par des objets « polyline » et les textes sont décrits par le code ASCII des lettres (par exemple C pour la lettre « D »). Il serait relativement facile de changer les

polygones en objets « path » et de remettre les textes sous une forme normale. Par contre, il n'y a pas de moyen simple pour associer les textes (qui correspondent en fait aux types de zone et de secteur) et les polygones correspondants.

5. ECHANGE DE DONNEES A DISTANCE : SVG OU GML ?

5.1 LIMITES DU FORMAT SVG

Nous avons vu avec cette maquette que le langage SVG est bien adapté pour publier des informations géographiques sur le Web. Mais qu'en est-il de l'échange de données entre SIG ou de l'utilisation de SVG comme format standard pour un SIG ? SVG est un langage permettant de décrire des graphiques et des animations. Il n'est absolument pas prévu pour décrire des informations géographiques. Il ne permet par exemple pas d'associer des propriétés définies par l'utilisateur aux différents polygones représentant des zones. Il pourrait donc décrire les objets graphiques eux-mêmes (leurs coordonnées), mais pas leurs propriétés. Dans la maquette, nous avons utilisé l'attribut « class » pour affecter un type à chaque zone ou secteur, mais nous ne pourrions pas associer plus d'informations en utilisant le seul document SVG.

De plus, nous avons vu qu'il existe des outils pour générer du SVG à partir de MapInfo, mais l'inverse n'est pas possible. Il faut donc réserver l'utilisation de SVG à la publication des données (sur le Web ou d'autres médias).

5.2 GML

Le langage GML (Geographic Markup Language) est standardisé par le groupe OpenGIS. Sa deuxième version date du 20 Février 2001. C'est également un langage XML, mais son objectif est de permettre la description de données géographiques. Son principe de base est simplement de décrire des objets, avec des propriétés définies par l'utilisateur, dont certaines peuvent représenter des coordonnées géographiques. Il pourrait donc être utilisé pour décrire les objets graphiques contenus dans les tables de MapInfo. Voici un exemple de document GML qui pourrait être utilisé pour décrire une zone géographique :

```
<Zone fid ="ZUB1">
        <Type>UB</Type>
        <COS>2.0</COS>
        <gml:extentOf>
                <gml:Polygon srsName=”http://www.opengis.net/gml/srs/epsg.xml#4326">
                        <gml:outerBoundaryIs>
                                <gml:LinearRing>
                                        <gml:coordinates>
        491888.999999459,5458045.99963358  491904.999999458,5458044.99963358
        491908.999999462,5458064.99963358 491924.999999461,5458064.99963358
        491925.999999462,5458079.99963359 491977.999999466,5458120.9996336
        491953.999999466,5458017.99963357
                                        </gml:coordinates>
                                </gml:LinearRing >
                        </gml:outerBoundaryIs>
                </gml:Polygon>
        </gml:extentOf>
</Zone >
```

On peut ainsi lier pour un même objet des informations textuelles (ici le type de zone et le C.O.S.) et des informations grahiques (un polygone). Ce format est bien mieux adapté que SVG pour la manipulation et l'échange des données géographiques. De plus, il offre tous les avantages des langages XML en termes de transformation par XSLT. On peut aisément écrire une feuille de style XSLT qui transforme un document GML en un document SVG qu'on peut présenter sur une page Web par exemple.

Toutefois, il faut noter que ce format est encore très jeune et qu'il existe à ce jour très peu d'outils pour le manipuler. Les premiers devraient faire leur apparition dans les mois qui viennent. Les développeurs qui veulent utiliser GML dès aujourd'hui sont donc obligés de développer leur propre code. Et surtout, GML ne fait pas l'objet d'une recommandation du W3C. On ne peut donc pas affirmer que ce standard va s'imposer. Vu que les serveurs de données géographiques sont de plus en plus souvent des serveurs Oracle, il se peut que le standard en matière de description de données géographiques devienne le format choisi par Oracle et non pas GML. On peut toutefois noter que s'il n'existe pas d'outils effectuant directement la conversion MapInfo ←→ GML, le composant MapXtend utilise GML pour diffuser des informations vers des mobiles.

5.3 PROPOSITION D'ARCHITECTURE

Nous proposons de s'appuyer sur GML pour définir une structure de document, à la fois pour l'aspect règlementaire du POS et pour les données cartographiques. Cette définition est formalisée par un schéma XML qui est une extension de celui de GML.

Notre préconisation est donc d'utiliser GML pour l'échange de données entre systèmes différents, et éventuellement pour la manipulation des données si les prochaines générations de SIG reconnaissent ce format en standard. Nous recommandons toutefois de rester très prudent sur ce point car même si GML dispose d'atouts non négligeables, rien ne permet d'affirmer qu'il sera le standard de demain en matière de données géographiques. Pour la consultation sur le web, par contre, l'utilisation de SVG reste plus adaptée car il existe déjà des visualisateurs performants et que ce format permet de réaliser des effets visuels qui s'avèrent utiles pour une présentation claire et attractive. Les informations textuelles pourront être présentées sur demande en transformant une partie du document GML en un document HTML grâce à une feuille XSLT. L'architecture à mettre en place sera donc la suivante :

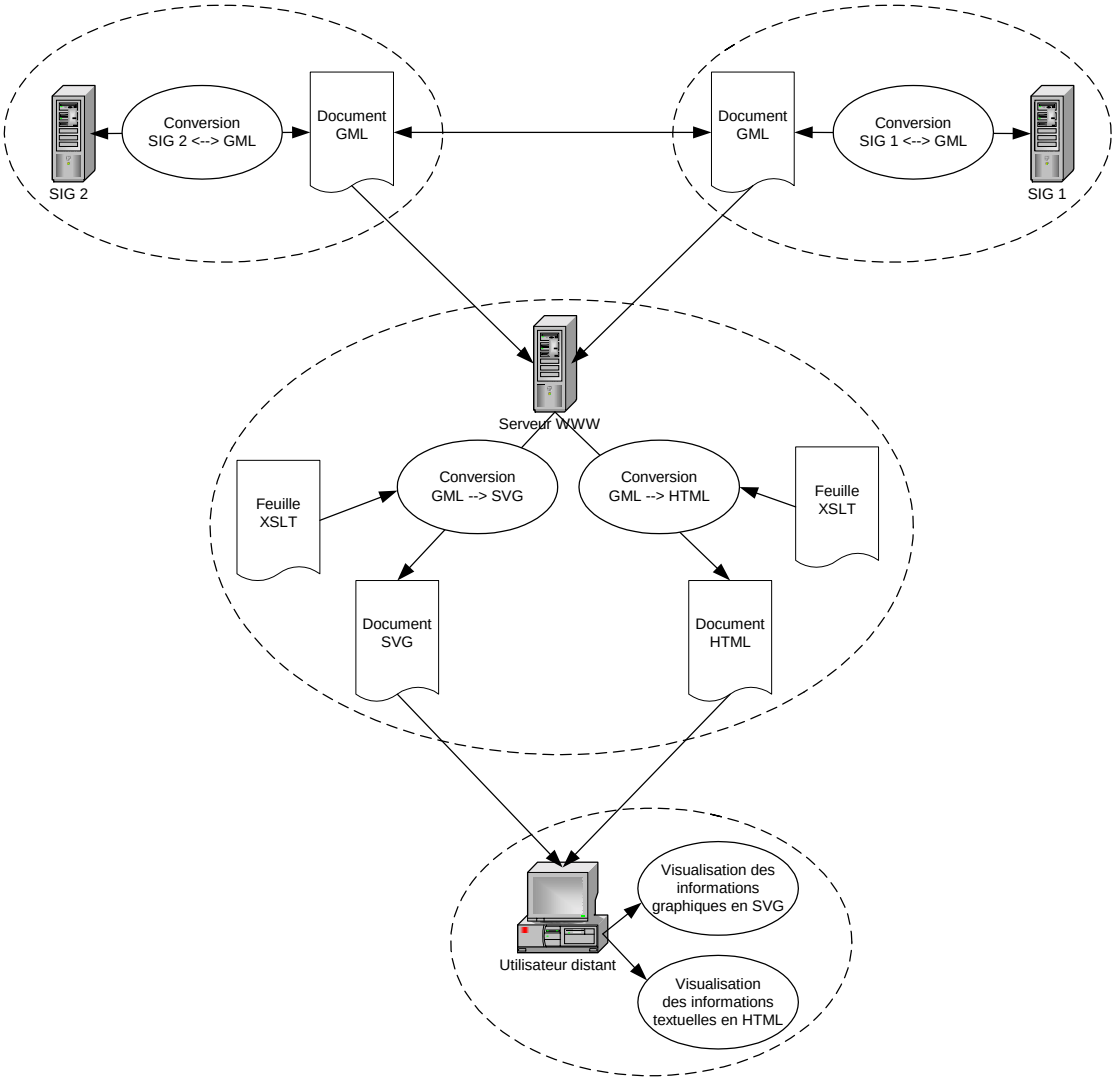
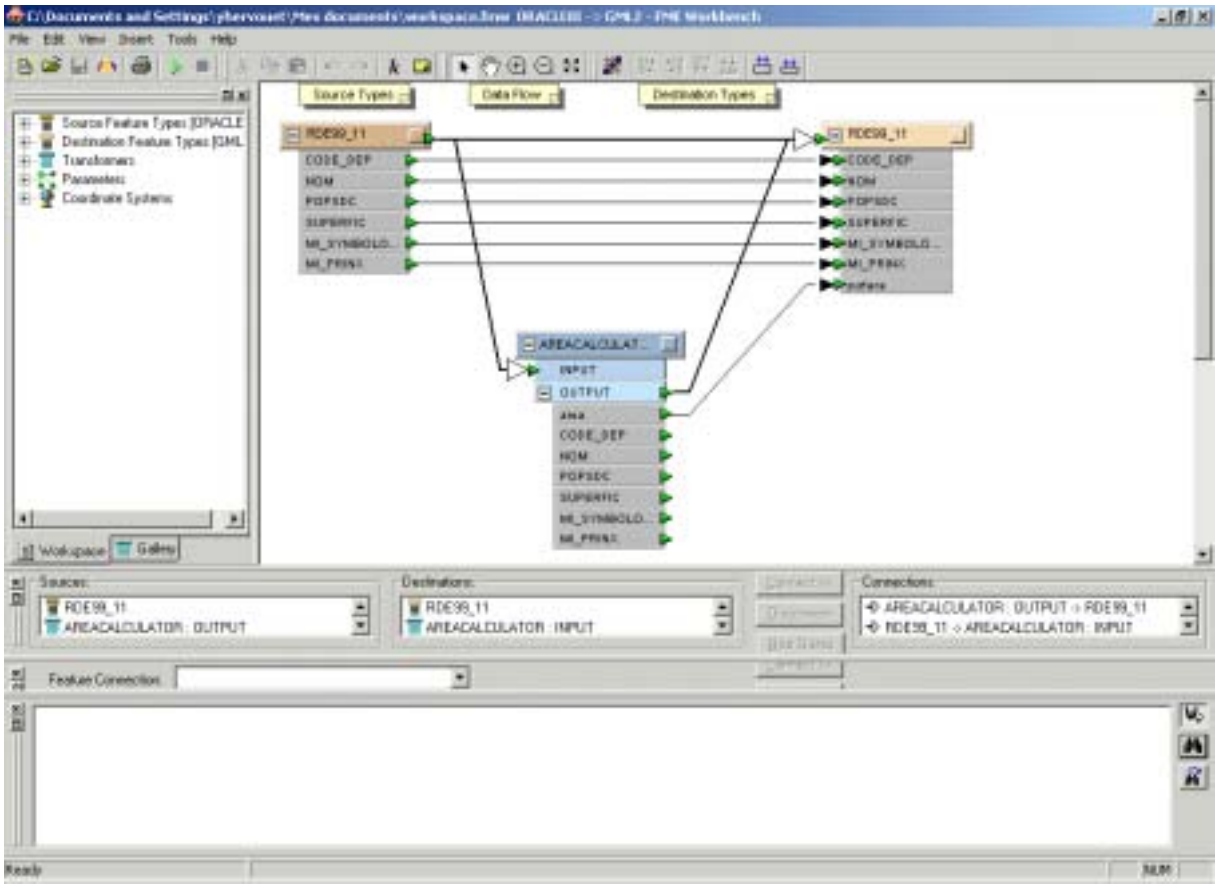


Figure 41 : Architecture préconisée pour l'échange et la consultation de données géographiques

5.4 OUTILS

Il existe encore peu d'outil spécifiquement conçu pour manipuler GML. La société **Safe Software** commerciale une suite de logiciels FME, qui permet de manipuler tous types de format de données géographiques, y compris GML. L'outil Workbench ci-dessous permet de convertir des données d'un format vers un autre, en appliquant des fonctions de calculs, des opérateurs géométriques et gère notamment les conversions de projection.



La société **Galdos Software** propose un outil qui permet de créer des feuilles de style visuellement pour mettre en forme des documents GML en produisant du SVG. Cet outil, Galdos Map Style Editor, est disponible à :

http://mapstyles.galdosinc.com/MapStyle.html

Il s'agit d'un outil encore peu mature et relativement pauvre.

6. AUTRES LANGAGES ET TECHNOLOGIES DE L'UNIVERS XML

6.1 SERVICES WEB (SOAP, WSDL ET UDDI)

Les services web font partie des applications de XML les plus susceptibles de révolutionner les applications web dans les mois à venir. Il est donc intéressant d'avoir un aperçu sur leur utilité et leur fonctionnement, même s'ils n'ont pas d'application directe dans notre contexte.

Un exemple courant de l'application des services web est celui d'une agence de voyage. Celle-ci propose à ses clients d'acheter un billet de train, de réserver l'hôtel et de prendre une assurance. Dans une agence traditionnelle, un employé se chargera de contacter les organismes concernés et de faire les démarches nécessaires. Si on se place maintenant dans le cadre d'une agence proposant les mêmes services sur le web, il faudra contacter l'application de réservation de billets de la société de chemin de fer, l'application de réservation de l'hôtel et celle de l'assurance. Toutes ces applications fonctionnent en des lieux différents, sur des systèmes différents, mais il faut pourtant qu'elles puissent communiquer. Le rôle des services web est le même que celui de l'employé de l'agence : Rechercher les services disponibles et contacter ces applications pour leur demander d'effectuer les opérations nécessaires.

Au cours de la dernière décennie sont nés plusieurs protocoles permettant de faire communiquer des applications à distance. On notera en particulier le développement de CORBA et de DCOM, les deux plus utilisés. Les services web s'en inspirent très fortement, avec en supplément la possibilité de faire dialoguer les applications quel que soit le langage utilisé pour les développer et le système sur lequel elles s'exécutent. De plus, ils sont très simples à implémenter et se basent comme leur nom l'indique sur le web. Ces qualités font des services web la pierre angulaire de l'évolution des logiciels en tant que services. Techniquement, les services web sont basés sur XML. Cela signifie que les langages qui leur permettent de fonctionner sont des spécialisations de XML et qu'ils sont facilement extensibles. Leur fonctionnement est décrit principalement par trois spécifications : SOAP, WSDL et UDDI, mais d'autres langages existent qui enrichissent les possibilités des services web.

SOAP (Simple Object Access Protocol) spécifie la manière utilisée par les services web pour communiquer entre eux. Il décrit en particulier la structure des messages echangés, la représentation des données dans ceux-ci, le protocole d'appel de procédure distante et la définition de l'utilisation de HTTP comme couche de transport des messages. WSDL (Web Service Description Language) est le langage utilisé pour décrire les services eux-mêmes. Ils sont représentés ici comme un ensemble de terminaisons agissant sur les messages. UDDI (Universal Description, Discovery and Integration) est un annuaire des services Web proposés par les entreprises. Plus d'informations sur ces langages sont disponibles aux adresses suivantes :

http://www.w3.org/TR/SOAP/ : Note du W3C sur SOAP
http://www.w3.org/TR/wsdl : Note du W3C sur WSDL
http://www.uddi.org/ : Documentation sur UDDI

6.2 AUTRES LANGAGES NORMALISES

6.2.1 RDF

RDF (Resource Description Framework) est un langage de description de documents destiné notamment à faciliter leur indexation par des moteurs de recherche. Les spécifications de RDF font l'objet de plusieurs documents du W3C :

http://www.w3.org/TR/REC-rdf-syntax/ : Recommandation du W3C pour la syntaxe de RDF

http://www.w3.org/TR/rdf-schema/ : Recommandation candidate pour les schémas RDF

http://www.w3.org/TR/rdf-mt/ : Working Draft sur la théorie du modèle RDF

http://www.w3.org/TR/rdf-testcases/ : Working Draft contenant des tests pour RDF

6.2.2 SMIL

SMIL (Synchronized Multimedia Integration Language) est un langage permettant de décrire des présentations multimédia. Il fait l'objet d'une recommandation du W3C :

http://www.w3.org/TR/REC-smil/ : Recommandation du W3C pour SMIL

6.2.3 MathML

MathML (Mathematical Markup Language) permet la distribution, l'échange et le traitement d'expressions mathématiques. Une des applications est de pouvoir insérer des formules mathématiques dans des documents HTML ou XML. Ce langage fait l'objet d'une recommandation du W3C :

http://www.w3.org/TR/REC-MathML/ : Recommandation du W3C pour MathML

7. VOLUMETRIE ET SOLUTIONS

La gestion de gros volumes de données constitue la principale limite du SVG. Des tests de volumétrie exposés dans le chapitre 7.1.2 permettent de quantifier cette limite. Les solutions existantes pour la gestion de gros volumes de données cartographiques sont présentées ensuite, afin d'étudier dans quelle mesure elles peuvent être mises en œuvre avec SVG.

7.1 VOLUMETRIE DES DONNEES

7.1.1 Généralités

Nous avons vu qu'un des principaux problèmes de SVG est l'impossibilité de contrôler le volume des données à télécharger. Voici quelques considérations permettant de limiter la taille des fichiers SVG.

Un point intéressant des spécifications de SVG est que les outils traitant ce format doivent être capables de lire les fichiers SVG compressés au format ZIP. Cela permet de réduire le fichier à moins du tiers de sa taille originale, ce qui représente un gain très important.

Ensuite, on peut essayer de réduire le contenu du fichier lui-même. Dans notre document SVG de la carte du P.O.S., ce sont les coordonnées des points qui constituent la plus grande partie du fichier. On peut facilement réduire la longueur de celles-ci au prix d'une perte de précision. Suivant l'utilisation que l'on souhaite faire des données, cette perte peut être gênante ou non.

L'utilisation d'une feuille de style CSS séparée, comme nous l'avons fait, évite de répéter pour chaque objet la définition de tout son style. Là encore, on réalise une économie de place non négligeable.

7.1.2 Tests de volumétrie

Afin de tester les limites de l'utilisation de SVG en termes de volumétrie des données, nous avons essayé d'effectuer la conversion d'un fichier cadastral d'une grande communauté de communes depuis MapInfo. Le fichier original (.MAP) a une taille de 36.1 Mo. La conversion avec SVGMapMaker a échoué. On peut donc déjà conclure que cet utilitaire n'est pas parfaitement fiable. Toutefois, les différents autres tests que nous avons effectués nous permettent de dire que le fichier final aurait du avoir une taille comprise entre 50 et 60 Mo, pour un temps de conversion d'environ 3h sur un Pentium II 266 MHz.

Avant d'échouer, l'utilitaire a tout de même produit un fichier de 5.3 Mo. Nous avons donc réalisé les autres tests sur celui-ci. Une fois compressé, celui-ci n'occupe déjà plus que 953 ko. Pour ouvrir ce fichier avec Adobe SVG Viewer et Internet Explorer, il faut un temps de 1mn 30s avec un Pentium II 266 MHz. La mémoire vive occupée est de 70 Mo. Pour chaque opération de zoom, il faut ensuite compter environ 30s.

La conclusion de ces tests est que si SVG est bien adapté pour les petits documents (notre exemple de carte de P.O.S. a une taille de 100 ko non compressée), il devient inutilisable pour les trop gros volumes. Malheureusement, lorsque l'on veut inclure le cadastre d'une commune ou une couche raster en fond, la taille des fichiers nécessaires augmente très rapidement. La seule solution est alors de mettre en œuvre une architecture client-serveur plus complexe, dans laquelle seule la partie visible et ses alentours sont transférés au client à chaque requête.

7.2 MISE EN ŒUVRE D'UN SERVEUR SPATIAL

Cette étude a pour but d'étudier l'apport de XML et de SVG pour la publication de cartes sur le web. Il existe cependant d'autres technologies qui remplissent le même rôle. Il est donc intéressant de les comparer pour déterminer lesquelles sont les plus adaptées.

Dans nos deux maquettes, l'ensemble des données graphiques est transféré au client qui les affiche ensuite en mode autonome. Il est possible cependant pour des gros volumes de données de stocker les données sur un serveur, avec une indexation spatiale, et de fournir au client uniquement les données qu'il souhaite afficher. La mise à disposition des données spatiales dans une telle architecture peut se faire selon trois modes.

7.2.1 Mode client / serveur

L'accès aux données se fait au travers d'un outil SIG installé sur le poste client. Cette installation comprend le logiciel fournit par l'éditeur, la couche client du SGBD et souvent une couche de liaison entre les deux.

Par exemple :

Oracle 8i + Spatial ⇔ Net 8 ⇔ MapInfo DBMS ⇔ MapInfo

La mise en place de cette solution correspond à une utilisation très différente de notre besoin. Son déploiement est lourd et son coût beaucoup plus élevé. Ces outils permettent alors d'utiliser des fonctions SIG complexes ainsi que des fonctions d'éditions.

7.2.2 Mode Intranet par imagette

Ce mode de diffusion correspond plus à notre besoin : le client n'a besoin que d'un navigateur Web, qui affiche les imagettes (au format GIF ou JPEG) qui sont construites par le serveur à la demande. Cette solution est mise en œuvre sur beaucoup de sites Internet d'aide à l'itinéraire. Elle présente les caractéristiques suivantes :

- aucune donnée vectorielle ne circule entre le client et le serveur,
- elle ne nécessite pas la mise en œuvre d'applet ou de plug-in sur le client,
- les actions de l'utilisateurs sont interprétées par du code JavaScript, et systématiquement envoyées au serveur qui les traite (sélection d'un objet, bulle d'information, déplacement, zoom…).

Le niveau d'interactivité dans cette architecture dépend du réseau : toutes les modifications d'affichage sur la carte nécessitent la génération d'une image et son téléchargement, opération qui peut être longue comme très rapide sur un réseau local. Des développements complémentaires peuvent alors « simuler » la manipulation de données vectorielles et rendre l'application ergonomique. Cependant, ces développements complémentaires sont généralement plus lourds et plus complexes qu'avec la manipulation réelle de vecteurs.

7.2.3 Mode Intranet par applet java

Il existe deux possibilités pour la mise en œuvre d'applet java : une applet fonctionnelle qui contient la logique applicative, et une applet légère qui ne prend en charge que la visualisation.

Note : L'utilisation des applet est parfaitement adaptée à la visualisation d'imagettes même si sa plus-value est moins évidente que dans l'affichage de vecteurs. C'est donc cette dernière configuration que nous étudions ici.

Cette solution est un mélange des deux solutions précédentes :

- L'affichage s'appuie sur une applet,
- le client est un navigateur Web,
- les données échangées entre le client et le serveur sont vectorielles.

L'affichage est pris en charge par une applet Java qui est téléchargé par le poste client à sa première utilisation.

Cette applet est fournie par l'éditeur SIG et est capable d'interpréter un format de description de données vectorielles propriétaire, généré à la demande par un serveur d'application.

ESRI fournit une applet standard intégrant un ensemble de fonctionnalités de dessin et de navigation sans développement. MapInfo propose une applet « nue », les développements se réduisant à la construction de l'interface.

Cette solution est mise en avant dans les Intranet, où les navigateurs sont normalisés et les réseaux rapides. Sur Internet, le téléchargement de telles applet n'est pas envisageable (5 Mo dans le cas d'ESRI).

Note : des mécanismes de généralisation permettent de « simplifier » les données vectorielles afin de limiter le volume de données transférées au client quand celle-ci sont volumineuses. Par exemple, quand l'échelle d'affichage choisie par l'utilisateur fait qu'un polygone est trop petit pour être affiché avec tous ses points, le serveur ne transfert qu'un seul point. Les temps d'affichage et de téléchargement sont ainsi réduits tandis qu'il n'y a aucune différence visuelle.

Ces mécanismes impliquent que le zoom sur des données requerra, à partir d'une certaine limite, de demander au serveur des données plus précises.

7.2.4 Mise en œuvre avec SVG

On peut assimiler le fonctionnement d'un viewer SVG à celui du mode Intranet par applet, puisqu'il fonctionne avec un navigateur (en général) et qu'il permet d'afficher des données vectorielles. Il présente cependant les différences suivantes :

- il ne se télécharge pas à la première utilisation comme une applet : il faut le télécharger et l'installer, et il sera certainement intégré au navigateur dans leurs prochaines versions,
- il est gratuit et indépendant des éditeurs,
- il n'intègre pas de fonctionnalités d'interactions avec un serveur.

Dans quelle mesure est-il possible de s'appuyer sur les fonctionnalités d'affichage d'un viewer SVG dans une architecture intégrant un serveur spatial ? La mise en œuvre d'une telle solution s'appuie sur les mêmes mécanismes que la génération d'imagettes à la demande. Ce qui implique :

- d'interpréter les actions de l'utilisateurs et les transmettre au serveur d'application (déplacement, zoom),
- d'extraire du serveur les données correspondant au besoin de l'utilisateur,
- de convertir les données vectorielles extraites en SVG,
- de renvoyer ces données au client.

A l'évidence, les mécanismes serveurs d'extractions des données fournis par les logiciels de SIG sont incontournables, SVG n'apportant à ce niveau aucune plus-value. Reste alors la conversion des données vectorielles en SVG.

La société Ilog propose les produits Jviews et Maps qui gèrent l'accès à des données spatiales stockées dans Oracle et leur affichage. Ces produits sont vendus sous la forme de composants logiciels Java et peuvent être mis en œuvre dans les architectures décrites ci-dessus : client/serveur, Intranet léger (prise en charge de la génération d'(images sur le serveur), Intranet lourd (fourniture d'une applet de visualisation).

Cette option présente deux avantages par rapport à nos besoins :

- la conversion au format SVG est fournie en standard, et peut éventuellement être personnalisée par programmation,
- l'orientation Java des produits permet de s'intégrer au mieux dans notre architecture cible, et de disposer des librairies de manipulation du XML.

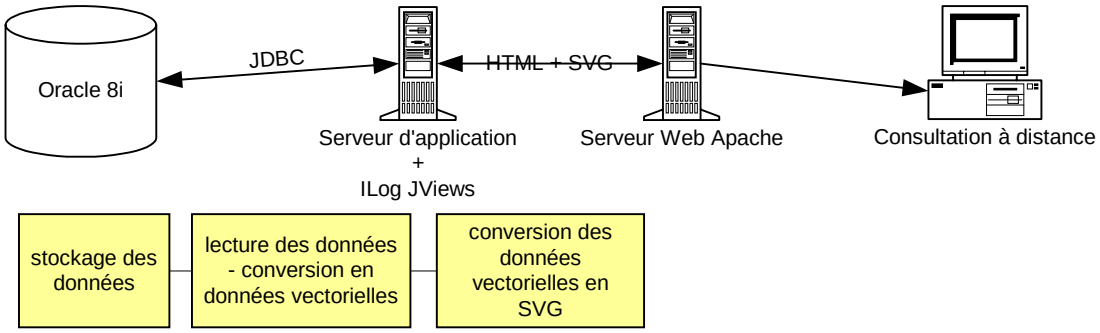


Figure 42 : Architecture proposée avec ILog JViews

Cette solution paraît séduisante et offre un certain nombre d'avantages par rapports aux autres solutions. Cependant, cette solution n'a pas été implémentée dans les prototypes et certains points doivent donc être pris en considération :

- Performance de la solution : la conversion SVG à la volée est certainement plus coûteuse en temps que la génération d'imagettes ;
- Nous n'avons pas trouvé d'éditeurs concurrents à Ilog sur ce créneau particulier et n'avons pas de retours d'expériences sur ce produit;
- A la différence des éditeurs de SIG, la mise en œuvre de la solution Ilog requiert du développement spécifique « cosmétique ».

8. IMPACTS SUR LES METIERS

L'objectif de ce chapitre est d'étudier l'impact des technologies XML sur les métiers autour de la gestion de données localisées, de leur constitution jusqu'à leur diffusion.

Cette étude ne se focalise pas sur le cas particulier du POS qui nous a servi de support tout au long de ce document, mais envisage de manière globale les différents métiers intervenant dans le cycle de vie de données géographiques.

Les définitions des différentes tâches de l'administrateur sont issus du manuel « Administrer les données localisées, une exigence pour les services - juin 2001» [2].

Les étapes du cycle de vie de données localisées sont :

- Production de données,
- Réception de données,
- Intégration des données,
- Administration des données,
- Mise à disposition des données.

L'utilisation d'XML à ces étapes impacte essentiellement les tâches afférant à la production de données (production, réception, intégration), et à leur mise à disposition (échange, consultation).

Les différents acteurs impliqués sur ces étapes sont :

- un producteur,
- un administrateur,
- un lecteur.

8.1 SCENARIO FONCTIONNEL RETENU

Cette étude a permis de situer SVG par rapport à la manipulation de données géographiques. Il apparaît que SVG permet d'afficher des cartes avec une richesse graphiques largement suffisante pour les besoins classiques de cartographie, mais n'a pas vocation à gérer des données géographiques proprement dit :

- il n'intègre pas de gestion de projection, ce qui complique la fusion de données de sources hétérogènes,
- il ne permet qu'une gestion très pauvre des données attributaires.

Le chapitre sur l'échange de données (§5) pose les limites du format SVG et introduit GML, un langage spécifique de description de données géographiques. GML est ouvert et peut être assimilé à une « boîte à outil SIG » qui peut être intégré à un modèle de données complexe. Quand un document intègre GML, il peut ensuite être transformé en SVG par une feuille de style qui élimine toutes les informations non graphiques.

Nous préconisons de définir un format pivot à partir de GML, en s'appuyant sur les mécanismes d'héritage de XML schéma, afin de véhiculer l'ensemble des informations géographiques et métiers et d'utiliser SVG comme format de visualisation, comme décrit dans le §5.

8.2 PRODUCTION DE DONNEES

Parmi les tâches principales assurées par l'administrateur, certaines sont directement associées à l'encadrement de la production de données.

Ces tâches sont :

- la définition des règles de production d'un lot de données,
- la réception d'un lot de données,

- la sélection des données nouvelles à intégrer au SI,
- la mise à jour du catalogue de données.

8.2.1 Définition des règles de production d'un lot de donnée

Cette tâche consiste à un ensemble de règles (utilisation de référentiels, normes, formats, critères de qualité géométrique, sémantique…), et constitue le cahier des charges des produits à intégrer dans le patrimoine.

Ce cahier des charges est à définir au préalable et est indispensable, que la production des données soit assurée par un service du MELT ou par un sous-traitant.

L'apport des technologies XML pour cette tâche est évident, et à plusieurs niveaux :

- la production d'un format pivot en XML Schema permet d'enrichir le cahier des charges d'une description détaillée des données à produire,
- le fournisseur des données a ainsi la possibilité de vérifier de manière automatique que les données produites sont conformes au Schema défini,
- le fournisseur dispose d'un schema XML dans un format que reconnaissent beaucoup d'outils, et peut ainsi être assisté dans sa production.

Note : L'apport de XML Schema pour la rédaction du cahier des charges ne remplace cependant pas la définition des critères de qualité géomatique ou sémantique.

8.2.2 Réception d'un lot de donnés

Cette tâche consiste à vérifier que le lot fourni par le producteur répond bien aux règles de production prédéfinies, avant qu'il puisse être intégré au patrimoine de données.

L'utilisation d'un format pivot pour cette étape permet d'assurer un premier niveau de vérification automatique sur le lot au niveau de la structure du document, et de la conformité de chaque attribut par rapport aux spécifications. Cette vérification est d'ailleurs à priori déjà effectuée par le fournisseur.

Ensuite, l'administrateur peut produire plusieurs documents construits à partir du document XML fourni et transformé par plusieurs feuilles de style XSL, et vérifier ainsi de manière précise :

- l'aspect cartographique : production d'un document SVG,
- l'aspect attributaire : production d'un document Excel.

XML lui fournit ainsi plusieurs niveaux de validation et une souplesse lui permettant de personnaliser ses outils de vérification.

8.2.3 Sélection des données nouvelles à intégrer

En fonction de l'évaluation des données produites et des besoins des utilisateurs, l'administrateur (ou le comité de pilotage) sélectionnera les données à intégrer au patrimoine commun.

En appliquant une feuille de style au document XML fourni, il est possible de filtrer les données en se basant sur les attributs ou éventuellement de supprimer des attributs inutiles. Par contre il n'est pas possible sans programmation d'effectuer des filtres géographiques.

8.2.4 Mise à jour du catalogue de données

La réalisation d'un catalogue de données correspond au souci de connaître et de faire connaître aux utilisateurs les données géographiques dont dispose le service, et qu'il peut proposer à la diffusion, en présentant leurs principales caractéristiques.

Dans le cas de l'utilisation d'XML pour la gestion des fiches constituant le catalogue, ces fiches peuvent être créées par transformation du document reçu au travers d'une feuille de style spécifiques. Ainsi, si le schéma XML associé au lot de données impose la saisie d'informations de description de ce lot, ces informations pourront être intégrées directement dans la fiche du catalogue.

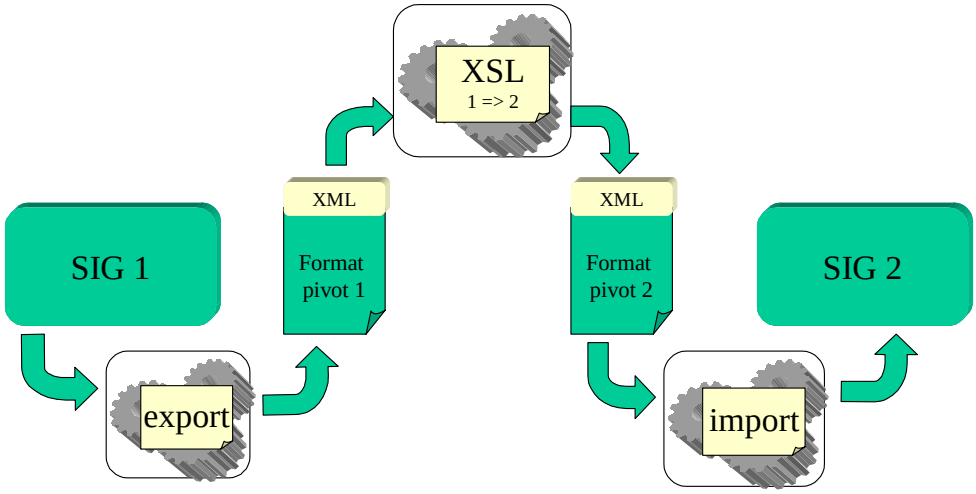
8.3 MISE A DISPOSITION DES DONNEES

Deux types d'utilisation des données est envisageables :

- import de ces données dans un autre référentiel géographique,
- consultation de ces données.

8.3.1 Echange de données

Dans le cas d'échange de données, de même que pour la récupération d'un lot de données, un format pivot à partir de GML est conseillé. Le schéma ci-dessous décrit ce processus d'échange :



Chaque référentiel géographique dispose d'un format pivot d'export ou d'import. Ainsi, la transformation d'un format pivot en un autre format pivot est grandement facilitée par les mécanismes de transformation par feuille de style :

- seules les informations nécessaires sont conservées dans la transformation,
- l'ouverture du standard XSLT permet éventuellement d'effectuer des transformations complexes, comme par exemple des conversions d'un système de projection vers un autre.

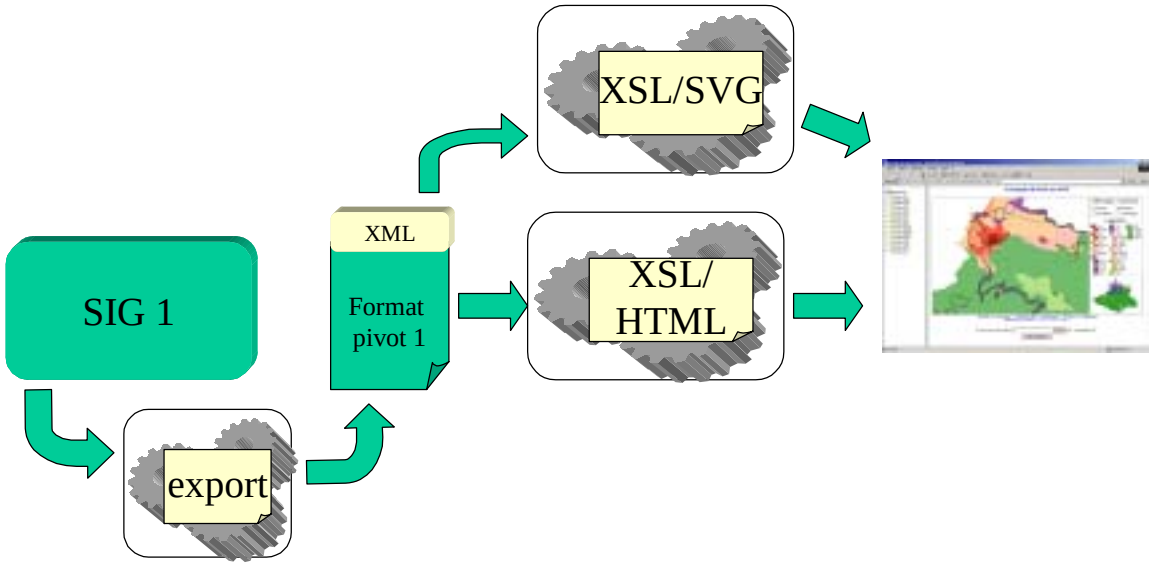
8.3.2 Consultation de données

On distingue deux types de consultations, qui trouve leur solution technique dans des architectures différentes :

- la consultation offline, avec la mise à disposition des données sur un CD-ROM,
- la consultation on-line sur un site Web.

Dans les deux cas, on peut s'appuyer sur la technologie SVG pour l'affichage, mais sa mise en œuvre est différente.

Pour la **consultation off-line**, on peut s'appuyer sur le même mécanisme d'export vers un format pivot, qui permet de filtrer les données en fonction de la commande (filtre sur les couches, restrictions géographiques) :



La conversion du format pivot vers SVG est assurée par deux feuilles de style :
une feuille de style qui contient la topologie et qui ne transforme que la composante graphique du fichier pivot,
une feuille de style qui transforme à la demande une extraction du fichier au format pivot en HTML.

La **consultation on-line** met en œuvre des mécanismes plus complexes, décrits dans le chapitre §7. Ces mécanismes permettent de manipuler des données plus volumineuses.

Annexes de l'étude

IDL_CERTU1/ETU_002 / 1.0

Client	: CERTU		
Entité	: SWORD/IDL		
Projet	: ETUDE EXPLOIRATOIRE XML-SVG		
Id Projet	: IDL_CERTU1		
Date (1)	: 25/02/02		
Etat	: ☒ **à valider**	☐ **validé**	
Diffusion	: ☐ **interne**	☒ **contrôlée**	☐ **libre**

(1) Date d'approbation (cf circuit de validation interne).

CIRCUIT DE VALIDATION

Version	Rédaction			Vérification			Approbation		
	NOM	DATE	VISA	NOM	DATE	VISA	NOM	DATE	VISA
1.0	Renaud Barate	21/02/2002		Yann Hervouët	25/02/02		Eric Bouvet	25/02/02	

HISTORIQUE DES EVOLUTIONS

Version	Objet de la version (citer les fiches de réception de document prises en compte)
1.0	Initialisation du document

LISTE DE DIFFUSION

Destinataire	Fonction	Nombre d'exemplaires	Support
Jacques BALME	CERTU – pôle géomatique	1	Electronique
Martine CHATAIN	CERTU – pôle géomatique	1	Electronique
Denis CHABRIER	CERTU – pôle géomatique	1	Electronique

DOCUMENTS REFERENCES

Origine	N°	Titre	Référence	Usage (*)

(*) : indiquer le contexte de citation du document : à lire au préalable, documents de référence, documents complémentaires, ...

SOMMAIRE

1. SOURCES HTML 1

1.1 PAGE D'ACCUEIL (INDEX.HTML) 2
1.2 PAGE PRINCIPALE (POS.HTML) 2
1.3 FORMULAIRE POUR L'EDITION D'UNE FICHE PARCELLAIRE (FORMPARCELLE.HTML) 3

2. SOURCES SVG ET JAVASCRIPT 5

2.1 CARTE DU P.O.S. (POS.SVG) 5
2.2 CARTE MINIATURE (MINIMAP.SVG) 7
2.3 TRAITEMENTS JAVASCRIPT (POS.JS) 7
2.4 SEMIOLOGIE DU P.O.S. (MAPSTYLE.CSS) 14

3. GENERATION D'UNE PAGE HTML CONTENANT UN ARTICLE DU REGLEMENT 16

3.1 LA SERVLET (SERVLETARTICLE.JAVA) 16
3.2 LA TRANSFORMATION XSLT (ARTICLE.XSL) 17

4. GENERATION D'UNE ARBORESCENCE POUR NAVIGUER DANS LE REGLEMENT 19

4.1 LA SERVLET (SERVLETARBO.JAVA) 19
4.2 LA TRANSFORMATION XSLT (ARBORESCENCE.XSL) 20

5. GENERATION DE LA LEGENDE DE LA CARTE 28

5.1 LA SERVLET (SERVLETLEGENDE.JAVA) 28
5.2 LA TRANSFORMATION XSLT (LEGENDE.XSL) 28

6. GENERATION D'UNE FICHE PARCELLAIRE 32

6.1 LA SERVLET (SERVLETFICHE.JAVA) 32
6.2 LA TRANSFORMATION XSLT GENERANT LE DOCUMENT FO (FICHE_PARCELLE-FO.XSL) 33
6.3 GENERATION DE LA CARTE SVG A INCLURE DANS LE DOCUMENT (PARCELLE_SVG-FO.XSL) 36
6.4 GENERATION DE LA LEGENDE A INCLURE DANS LA FICHE (LEGENDE-FO.XSL) 37

7. GENERATION DU REGLEMENT EN PDF 39

7.1 LA SERVLET (SERVLETREGLEMENT.JAVA) 39
7.2 GENERATION DU DOCUMENT FO (REGLEMENT-FO.XSL) 40
7.3 GENERATION DE LA CARTE A INCLURE DANS LE DOCUMENT (CARTE_SVG-FO.XSL) 42
7.4 GENERATION DE LA LEGENDE A INCLURE DANS LE DOCUMENT (LEGENDE-FO.XSL) 43

8. GENERATION DE LA CARTE DU P.O.S. EN PDF AU FORMAT A3 44

8.1 LA SERVLET (SERVLETCARTEA3.JAVA) 44
8.2 GENERATION DU DOCUMENT FO (CARTEA3-FO.XSL) 45
8.3 GENERATION DE LA CARTE SVG INCLUSE DANS LE DOCUMENT (CARTEA3_SVG-FO.XSL) 46
8.4 GENERATION DE LA LEGENDE INCLUSE DANS LE DOCUMENT (LEGENDEA3_SVG-FO.XSL) 47

- o O o -

1. INSTALLATION

1.1 INSTALLATION DE TOMCAT

La maquette réalisée utilise des servlets pour fonctionner. Elle nécessite donc un serveur permettant l'exécution de ces servlets. Nous avons utilisé pour cela Apache Tomcat 4.0.2, téléchargeable à l'adresse suivante :

http://jakarta.apache.org/tomcat/ : Page d'accueil d'Apache Tomcat

Il suffit ensuite de suivre les instructions d'installation du serveur Tomcat. Celui-ci nécessite l'installation préalable de Java, téléchargeable sur le site de Sun :

http://java.sun.com/j2se/1.4/ : Java 2 Standard Edition v1.4

1.2 COPIE DES FICHIERS

Une fois que le serveur Tomcat est installé, il faut copier les fichiers de la maquette (répertoire WebApp) dans le répertoire *webapps* de Tomcat. Les deux fichiers JAR (batik.jar et xml-apis.jar) doivent être copiés dans le répertoire *lib* de Tomcat.

1.3 LANCEMENT DE TOMCAT

Il suffit ensuite de lancer le serveur (Menu Démarrer/Programmes/Apache Tomcat 4.0/Start Tomcat). L'adresse à entrer dans un navigateur Web pour accéder à l'application est http://localhost:8080/WebApp/index.html. Si la machine utilisée n'est pas le serveur, il faut remplacer « localhost » par le nom du serveur.

Pour arrêter le serveur Tomcat, il faut accéder au menu « Menu Démarrer/Programmes/Apache Tomcat 4.0/Stop Tomcat ».

2. SOURCES HTML

2.1 PAGE D'ACCUEIL (INDEX.HTML)

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Exemple de POS en SVG</title>
<base target="_top">
</head>

<frameset frameborder="0" bordercolor="#0063BD" border="0" framespacing="0" cols="170,*">
  <frame src="servlet/maquettefinale.ServletArbo" name="menu" frameborder="NO"
bordercolor="#0063BD">
  <frame src="pos.html" name="pos" frameborder="NO" bordercolor="#0063BD"
  scrolling="auto" marginwidth="13" style="border-left: medium double">
</frameset>
</html>
```

2.2 PAGE PRINCIPALE (POS.HTML)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/REC-
html40/loose.dtd">
<!--Note: This file is the result of an export from MapInfo using DBx Geomatics 'SVGMapMaker'
Version 1.00 (Mapbasic program) licensed to evaluation
You may copy, modify, and distribute this file, if you include this notice and do not charge
for the distribution. This file is provided 'as is' without warranties of any kind, including
any implied warranties.-->
<html><head>
<title>Exemple de POS en SVG</title>
<meta name="generator" Content="DBx Geomatics SVG Export 1.00">
<script type = "text/javascript" language="JavaScript1.2" src="pos.js"></script>
<style type="text/css">
body {background-color: #ffffff}
h2 {font-family: Verdana, Arial; font-size: 18px; color: #0000AF}
span.instructions {font-family: Verdana, Arial; font-size: 10px; color: #0000AF}
a {font-family: Verdana, Arial; font-size: 9px; color: #7b7755}
</style>
</head>
<body>
<table width="100%"><tr><td align="center">
<h2><b>Exemple de POS en SVG</b></h2>
<table border="1" cellpadding="0" cellspacing="0" width="800" height="450">
<tr>
<td>
<embed src="pos.svg" width="600" height="450" name="svgMap" align="center" type="image/svg-
xml" pluginspage="http://www.adobe.com/svg/viewer/install/"></embed>
<noembed>An SVG Viewer such as the Adobe SVG Viewer plug-in is required to see this map.
</noembed>
</td>
<td>
<table cellpadding="0" cellspacing="0" width="200" height="450">
<tr><td>
```

```
<embed src="servlet/maquettefinale.ServletLegende" width="200" height="300" name="svgLegende"
align="center" type="image/svg-xml"
pluginspage="http://www.adobe.com/svg/viewer/install/"></embed>
<noembed>An SVG Viewer such as the Adobe SVG Viewer plug-in is required to see this map.
</noembed>
</td></tr>
<tr><td>
<embed src="minimap.svg" width="200" height="150" name="svgMiniMap" align="center"
type="image/svg-xml" pluginspage="http://www.adobe.com/svg/viewer/install/"></embed>
<noembed>An SVG Viewer such as the Adobe SVG Viewer plug-in is required to see this map.
</noembed>
</td></tr>
</table>
</td>
</tr>
</table>
<span class="instructions"> <b>Ctrl-Clic</b> pour <b>Zoomer</b>  |  <b>Ctrl-
Shift-Clic</b> pour <b>Dézoomer</b> |  <b>Alt-Clic</b> pour
<b>Recentrer</b></span><br>
<span class="instructions"> <b>Cliquer sur la carte</b> pour plus
d'informations.</span><br>
</td></tr>
</table>
<p align="center">
<form name="SelParcelle" method="get" ID="SVGForm" action="">
Zoom sur la parcelle n°
<INPUT TYPE="text" NAME="name" VALUE="" SIZE="20" MAXLENGTH="150">
<INPUT TYPE="button" NAME="go" VALUE="Zoom"
onclick="zoomParcelle(document.SelParcelle.name.value)">
(ex : 9186AB935)<br>
<INPUT TYPE="button" NAME="vue" VALUE="Vue normale" onclick="vueNormale()">
<INPUT TYPE="button" NAME="fiche" VALUE="Fiche parcellaire PDF" onclick="formParcelle()">
<INPUT TYPE="button" NAME="reglement" VALUE="Règlement PDF" onclick="reglementPDF()">
<INPUT TYPE="button" NAME="carte" VALUE="Carte A3 PDF" onclick="cartePDF()">
</form>
</p>
</body>
</html>
```

2.3 FORMULAIRE POUR L'EDITION D'UNE FICHE PARCELLAIRE (FORMPARCELLE.HTML)

```
<html>
<head>
<title>Edition d'une fiche parcellaire</title>
</head>
<body>
<form name="FicheParcelle" method="get" ID="SVGForm" action="">
<table><tr>
<td>N° de la parcelle</td>
<td><INPUT TYPE="text" NAME="num" VALUE="" SIZE="20" MAXLENGTH="150"></td>
</tr><tr>
<td>Type de zone</td>
<td><INPUT TYPE="text" NAME="zone" VALUE="" SIZE="20" MAXLENGTH="150"></td>
</tr><tr>
```

```
<td>Type de secteur</td>
<td><INPUT TYPE="text" NAME="secteur" VALUE="" SIZE="20" MAXLENGTH="150"></td>
</tr></table>
<p align="center">
<INPUT TYPE="button" NAME="go" VALUE="Edition de la fiche"
onclick="window.opener.ficheParcelle(document.FicheParcelle.num.value,
document.FicheParcelle.zone.value, document.FicheParcelle.secteur.value)">
</p>
</form>
</body>
</html>
```

3. SOURCES SVG ET JAVASCRIPT

3.1 CARTE DU P.O.S. (POS.SVG)

Ce document est trop volumineux pour être présenté au complet. Certaines parties (indiquées par …) ont donc été coupées.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="mapstyle.css" type="text/css"?>
<!--<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd" >-->
<!--Note: This file is the result of an export from MapInfo using DBx Geomatics 'SVGMapMaker' Version 1.00 (Mapbasic program) licensed to evaluation
You may copy, modify, and distribute this file, if you include this notice and do not charge for the distribution. This file is provided 'as is' without warranties of any kind, including any implied warranties.-->
<svg xmlns:xlink="http://www.w3.org/1999/xlink" xml:space="preserve" width="600" height="450" viewBox="2500 1000 2400 1800" enableZoomAndPanControls="true" style="shape-rendering:geometricPrecision;text-rendering:geometricPrecision;image-rendering:optimizeQuality" onzoom="changeMinimap(evt);" onscroll="changeMinimap(evt);">
<title>Secteur,...,Zone Carte</title>
<desc>Map Zoom: 9.19629 km | Center (604037.7,1829399.37) | Min (596628.62,1823505.1) | Max (611446.89,1835293.63) | Coordinate System: Lambert II Carto - Paris</desc>

<defs>

<!-- Liste de tous les secteurs géographiques -->
<g id="ListeSecteurs">
<use xlink:href="#Secteur1"/>
<use xlink:href="#Secteur2"/>
<use xlink:href="#Secteur3"/>
<use xlink:href="#Secteur4"/>
…
<use xlink:href="#Secteur73"/>
<use xlink:href="#Secteur74"/>
</g>

<!-- Définition des masques -->
<mask id="OuterMask" maskUnits="userSpaceOnUse" x="0" y="0" width="8000" height="6000">
  <rect x="0" y="0" width="8000" height="6000" style="fill:white"/>
  <use xlink:href="#ListeSecteurs" style="fill:black;stroke:none"/>
</mask>

<mask id="InnerMask" maskUnits="userSpaceOnUse" x="0" y="0" width="8000" height="6000">
  <use xlink:href="#ListeSecteurs" style="fill:white;stroke:none"/>
</mask>
</defs>

<!-- Image raster en fond -->
<image x="3350" y="1550" width="200" height="200" xlink:href="images/background.jpg" />

<!-- Représentation des zones et secteurs géographiques -->
<g id="Zones" style="visibility:visible;fill-opacity:0.7;pointer-events:visible;stroke:none" onclick="showInfo(evt.target)"
onmouseover="mouseOverPoly(evt)" onmouseout="mouseOutPoly(evt)">

<g id="ZonesUA" class="UA">
<path id="Zone5" d="M4266 1667 L4266 1668 4252 1663 4223 1661 4221 1664 4225 1671 4235 1675 4248 1683 4251 1689 4260 1699 4265 1705 4227 1707 4225 1703 4206 1704 4203 1704 4195 1703 4177
```

```
1701 4167 1699 4170 1690 4157 1688 4144 1682 4147 1672 4124 1667 4117 1662 4117 1661 4118 1661
4123 1663 4128 1662 4135 1638 4130 1633 4152 1601 4164 1611 4179 1599 4189 1603 4189 1607 4208
1613 4206 1621 4220 1626 4221 1625 4234 1629 4262 1635 4263 1634 4263 1635 4265 1665 4265 1666
4265 1667 4266 1667
 Z" />
<path id="Zone33" d="M3415 1776 L3412 1781 3411 1782 3405 1779 3404 1780 3401 1780 3402 1777
3393 1775 3387 1799 3385 1807 3374 1824 3360 1844 3353 1857 3351 1866 3338 1864 3364 1820 3367
1816 3373 1805 3374 1799 3375 1798 3378 1787 3372 1785 3373 1780 3374 1772 3364 1771 3367 1753
3362 1752 3360 1754 3357 1755 3354 1754 3353 1752 3343 1747 3337 1759 3343 1760 3342 1770 3346
1770 3340 1806 3317 1800 3320 1789 3313 1786 3316 1771 3311 1770 3303 1771 3302 1766 3300 1766
3300 1764 3296 1765 3295 1759 3306 1755 3301 1745 3307 1742 3307 1745 3311 1745 3310 1739 3313
1738 3313 1739 3318 1738 3318 1739 3325 1737 3324 1734 3334 1730 3335 1728 3333 1723 3332 1722
3329 1707 3326 1708 3325 1702 3321 1702 3319 1686 3310 1686 3297 1686 3297 1679 3297 1673 3300
1666 3306 1658 3311 1663 3328 1664 3338 1663 3347 1661 3346 1654 3336 1626 3336 1624 3331 1598
3352 1603 3351 1615 3351 1619 3347 1627 3382 1633 3388 1634 3389 1634 3410 1634 3430 1634 3432
1625 3443 1622 3471 1616 3474 1629 3476 1635 3477 1657 3478 1681 3472 1685 3469 1686 3472 1693
3469 1694 3462 1696 3464 1701 3463 1702 3464 1705 3465 1704 3466 1707 3458 1709 3457 1705 3454
1691 3448 1692 3440 1691 3439 1691 3430 1691 3427 1698 3426 1702 3423 1708 3420 1713 3418 1713
3415 1722 3412 1729 3406 1727 3405 1740 3402 1758 3408 1760 3410 1761 3409 1765 3408 1765 3407
1773 3415 1776
 Z" />
<path id="Zone38" d="M3236 1895 L3226 1897 3224 1897 3207 1897 3207 1898 3207 1884 3242 1884
3242 1894 3236 1895
 Z" />
<path id="Zone47" d="M3313 1497 L3314 1512 3317 1527 3317 1528 3318 1531 3314 1534 3308 1542
3303 1549 3289 1542 3282 1542 3282 1533 3297 1526 3301 1525 3300 1524 3300 1521 3294 1521 3294
1511 3288 1503 3282 1508 3276 1508 3276 1501 3276 1452 3307 1490 3308 1490 3313 1497
 Z" />
<path id="Zone48" d="M3457 2537 L3459 2537 3461 2535 3462 2534 3463 2532 3464 2533 3465 2533
3463 2526 3464 2514 3463 2509 3461 2504 3458 2501 3464 2491 3466 2491 3470 2488 3472 2486 3473
2479 3474 2476 3476 2477 3480 2467 3484 2464 3485 2456 3485 2450 3493 2450 3497 2448 3499 2448
3500 2448 3500 2451 3501 2451 3506 2451 3505 2455 3505 2456 3506 2455 3506 2456 3510 2454 3514
2464 3515 2464 3517 2463 3517 2476 3519 2479 3521 2484 3520 2485 3519 2493 3516 2507 3515 2515
3506 2501 3505 2501 3504 2502 3511 2526 3514 2530 3508 2534 3499 2539 3492 2541 3483 2542 3486
2550 3474 2559 3468 2563 3467 2558 3458 2548 3454 2546 3457 2544 3457 2541 3455 2541 3457 2537
 Z" />
<g id="SecteursUAa" class="UAa" style="fill:none">
<path id="Secteur47" d="M3431 1634 L3432 1625 3443 1622 3471 1616 3476 1635 3478 1681 3467
1688 3454 1691 3452 1681 3449 1681 3437 1681 3427 1680 3414 1677 3421 1663 3428 1643 3431 1634
 Z" />
<path id="Secteur48" d="M3474 2559 L3468 2563 3467 2558 3457 2548 3454 2546 3456 2544 3457
2541 3455 2541 3457 2537 3459 2537 3460 2535 3461 2534 3463 2532 3464 2532 3463 2526 3463 2514
3462 2509 3460 2504 3458 2501 3464 2491 3466 2491 3465 2491 3470 2488 3471 2486 3472 2479 3473
2476 3476 2477 3480 2467 3483 2464 3485 2456 3485 2450 3493 2450 3497 2448 3498 2448 3500 2448
3500 2451 3506 2451 3505 2456 3510 2454 3514 2464 3517 2463 3517 2476 3519 2479 3521 2484 3520
2485 3519 2493 3516 2507 3515 2515 3514 2515 3505 2501 3504 2502 3511 2526 3514 2530 3508 2534
3499 2539 3492 2541 3483 2542 3486 2550 3474 2559
 Z" />
</g>
</g>
…
<!-- Réalisation du double trait pour les secteurs -->
<use id="InnerStroke" xlink:href="#ListeSecteurs" style="pointer-events:none;stroke-
linejoin:round;stroke:black;stroke-width:10;stroke-dasharray:20,20;fill:none"
mask="url(#InnerMask)"/>
<use id="OuterStroke" xlink:href="#ListeSecteurs" style="pointer-events:none;stroke-
linejoin:round;stroke:black;stroke-width:10;fill:none" mask="url(#OuterMask)"/>


</g>


<!-- Représentation des parcelles -->
<g id="Parcelles" style="stroke:rgb(102,102,102);fill:none;visibility:hidden;pointer-
events:none;stroke-linejoin:round;stroke-width:0.1" onclick="alert('Parcelle n°' +
evt.target.getAttribute('id'));">
<path id="9186AB1090" d="M3533 1697 L3530 1702 3530 1701 3532 1697 3531 1697 3531 1696 3533
1697
 Z" />
<path id="9186AB1162" d="M3537 1661 L3540 1662 3536 1670 3532 1676 3529 1675 3533 1669 3535
1665 3537 1661
 Z" />
```

```
<path id="9186AB1163" d="M3536 1670 L3540 1662 3543 1663 3539 1670 3539 1671 3535 1677 3532
1676 3536 1670
 Z" />
<path id="9186AB1164" d="M3539 1671 L3539 1670 3543 1663 3546 1664 3542 1671 3542 1673 3538
1679 3535 1677 3539 1671
 Z" />
…
<path id="9186AB184" d="M3391 1640 L3394 1642 3390 1648 3388 1651 3385 1650 3386 1649 3387
1646 3391 1640
 Z" />
<path id="9186AB186" d="M3382 1649 L3379 1648 3380 1647 3381 1644 3384 1638 3385 1638 3388
1639 3384 1645 3382 1649
 Z" />
 </g>


<!-- Couche cosmétique -->
<g id="cosmetic">
<path id="fog" style="stroke:none;fill-opacity:0.5;fill:white;visibility:hidden;pointer-
events:none"/>
<path id="bluepath" style="stroke-linejoin:round;stroke-
width:0.1;stroke:blue;fill:none;visibility:hidden;pointer-events:none"/>
<path id="redpath" style="stroke-
linejoin:round;stroke:red;fill:none;visibility:hidden;pointer-events:none"/>
</g>


<!-- Eléments nécessaires à l'affichage des "Tooltips" -->
<g id="tooltips">
<rect id="TooltipRect" x="0" y="0" width="500" height="75" rx="25" ry="25"
style="visibility:hidden;fill:rgb(225,225,200);stroke-width:1;
stroke:rgb(0,0,128);opacity:0.8;pointer-events:none"></rect>
<text id ="Tooltip" style="fill:rgb(0,0,0);visibility:hidden;font-weight:normal; font-
family:'Arial';font-size:60;text-anchor:middle;pointer-events:none" x="0" y="0">!</text>
<animate id="fade" attributeName="opacity" begin="Zones.mouseover" dur="1s" from="0" to="1"
fill="freeze" />
</g>


</svg>
```

3.2 CARTE MINIATURE (MINIMAP.SVG)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<svg xmlns:xlink="http://www.w3.org/1999/xlink" id="cont" width="200" height="150" viewBox="0
0 200 150" zoomAndPan="disable" externalResourcesRequired="true">


<image x="0" y="0" width="200" height="150" xlink:href="images/minimap.gif" />


<rect id="guide" x="71" y="29" width="57" height="43" style="fill:blue;fill-opacity:0.3;"
cursor="move" onmouseout="DoOnMouseOut( evt )" onmousedown="DoOnMouseDown( evt )"
onmouseup="DoOnMouseUp( evt )" onmousemove="DoOnMouseMove( evt )"/>


<rect id="contour" x="0" y="0" width="200" height="150" style="stroke:black;fill:none"/>


</svg>
```

3.3 TRAITEMENTS JAVASCRIPT (POS.JS)

```
var formWindow;
```

```
// Lors du clic sur une zone, ouvre une fenêtre avec la description correspondante
function showInfo (poly) {
	var typeZone, typeSecteur;
	if (poly.getAttribute("id").slice(0,4) == "Zone") {
		typeZone = poly.parentNode.getAttribute("class");
		typeSecteur = typeZone;
	} else if (poly.getAttribute("id").slice(0,7) == "Secteur") {
		typeSecteur = poly.parentNode.getAttribute("class");
		typeZone = poly.parentNode.parentNode.getAttribute("class");
	}

	var infoWindow=open("servlet/maquettefinale.ServletArticle?typeZone=" + typeZone +
"&typeSecteur=" + typeSecteur +
"&numArticle=0","info","toolbar=0,directories=0,menu=0,scrollbars=1,location=0,resizable=1,sta
tus=0,top=50,left=200,width=700,height=600,menubar=0,hotkeys=0");
	infoWindow.focus();
	parent.frames['menu'].location.href = "servlet/maquettefinale.ServletArbo?typeZone=" +
typeZone + "&typeSecteur=" + typeSecteur + "&numArticle=0";
}

// Ouvre le formulaire nécessaire pour une fiche parcellaire
function formParcelle() {
	formWindow=open("formParcelle.html","form","toolbar=0,directories=0,menu=0,scrollbars=1
,location=0,resizable=1,status=0,top=200,left=10,width=300,height=200,menubar=0,hotkeys=0");
	formWindow.focus();
}

// Fonction appelée lors du passage de la souris sur une zone
function mouseOverPoly (evt) {
	var fog = document.svgMap.getSVGDocument().getElementById("fog");
	fog.setAttribute("d", evt.getTarget().getAttribute("d"));
	fog.style.setProperty('visibility', 'visible');

	showToolTip(evt);
}

// Fonction appelée lorsque la souris sort d'une zone
function mouseOutPoly (evt) {
	var fog = document.svgMap.getSVGDocument().getElementById("fog");
	fog.style.setProperty('visibility', 'hidden');

	hideToolTip(evt);
}

// Affiche le "Tooltip"
function showToolTip (evt) {
	var target = evt.getTarget();
	var svgdoc = target.getOwnerDocument();
	var svgdocElement = svgdoc.getDocumentElement();
	var scale = svgdocElement.getCurrentScale();
	var translateX = svgdocElement.getCurrentTranslate().getX();
	var translateY = svgdocElement.getCurrentTranslate().getY();
	var pixel = 4 / scale;
	var offsetX = parseFloat(2500) - translateX * pixel;
	var offsetY = parseFloat(1000) - translateY * pixel;

	var svgobj = svgdoc.getElementById ("Tooltip");
	svgobj.setAttribute ('x', Math.round(offsetX + evt.getClientX() * pixel));
	svgobj.setAttribute ('y', (Math.round(offsetY + evt.getClientY() * pixel-52/scale)));
	var svgstyle = svgobj.getStyle();
	svgstyle.setProperty ('visibility', 'visible');
	svgstyle.setProperty('font-size', 48/scale);
```

```
        svgobj.getFirstChild().setData(target.parentNode.getAttribute("class"));

        var txtlen=svgobj.getComputedTextLength()+40/scale;
        var svgobj = svgdoc.getElementById ("TooltipRect");
        svgobj.setAttribute ('x', (Math.round(offsetX + evt.getClientX() * pixel-txtlen/2)));
        svgobj.setAttribute ('y', (Math.round(offsetY + evt.getClientY() * pixel-104/scale)));
        svgobj.setAttribute ('width', txtlen);
        svgobj.setAttribute ('height',60/scale);
        svgobj.setAttribute ('rx', 20/scale);
        svgobj.setAttribute ('ry', 20/scale);
        var svgstyle = svgobj.getStyle();
        svgstyle.setProperty ('stroke-width', 1/scale);
        svgstyle.setProperty ('visibility', 'visible');
}

// Cache le "Tooltip"
function hideToolTip(evt) {
        var svgdoc = evt.getTarget().getOwnerDocument();
        svgdoc.getElementById("Tooltip").getStyle().setProperty('visibility', 'hidden');
        svgdoc.getElementById("TooltipRect").getStyle().setProperty('visibility', 'hidden');
}

// Affiche les zones et les secteurs
function showZones() {
        document.svgMap.getSVGDocument().getElementById("Zones").getStyle().setProperty('visibi
lity','visible');
        document.svgLegende.getSVGDocument().getElementById("zone_check").getStyle().setPropert
y('visibility','visible');
        document.svgLegende.getSVGDocument().getElementById("zone_uncheck").getStyle().setPrope
rty('visibility','hidden');
}

// Cache les zones et les secteurs
function hideZones() {
        document.svgMap.getSVGDocument().getElementById("Zones").getStyle().setProperty('visibi
lity','hidden');
        document.svgLegende.getSVGDocument().getElementById("zone_check").getStyle().setPropert
y('visibility','hidden');
        document.svgLegende.getSVGDocument().getElementById("zone_uncheck").getStyle().setPrope
rty('visibility','visible');
}

// Affiche les parcelles
function showParcelles() {
        document.svgMap.getSVGDocument().getElementById("Parcelles").getStyle().setProperty('vi
sibility','visible');
        document.svgLegende.getSVGDocument().getElementById("parcelle_check").getStyle().setPro
perty('visibility','visible');
        document.svgLegende.getSVGDocument().getElementById("parcelle_uncheck").getStyle().setP
roperty('visibility','hidden');
}

// Cache les parcelles
function hideParcelles() {
        document.svgMap.getSVGDocument().getElementById("Parcelles").getStyle().setProperty('vi
sibility','hidden');
        document.svgLegende.getSVGDocument().getElementById("parcelle_check").getStyle().setPro
perty('visibility','hidden');
        document.svgLegende.getSVGDocument().getElementById("parcelle_uncheck").getStyle().setP
roperty('visibility','visible');
}

// Active la couche "Zones"
function activeZones() {
```

```
	document.svgMap.getSVGDocument().getElementById("Parcelles").getStyle().setProperty('po
inter-events','none');
	document.svgLegende.getSVGDocument().getElementById("zone_radio_check").getStyle().setP
roperty('visibility','visible');
	document.svgLegende.getSVGDocument().getElementById("zone_radio_uncheck").getStyle().se
tProperty('visibility','hidden');
	document.svgLegende.getSVGDocument().getElementById("parcelle_radio_check").getStyle().
setProperty('visibility','hidden');
	document.svgLegende.getSVGDocument().getElementById("parcelle_radio_uncheck").getStyle(
).setProperty('visibility','visible');
}

// Active la couche "Parcelles"
function activeParcelles() {
	document.svgMap.getSVGDocument().getElementById("Parcelles").getStyle().setProperty('po
inter-events','visible');
	document.svgLegende.getSVGDocument().getElementById("zone_radio_check").getStyle().setP
roperty('visibility','hidden');
	document.svgLegende.getSVGDocument().getElementById("zone_radio_uncheck").getStyle().se
tProperty('visibility','visible');
	document.svgLegende.getSVGDocument().getElementById("parcelle_radio_check").getStyle().
setProperty('visibility','visible');
	document.svgLegende.getSVGDocument().getElementById("parcelle_radio_uncheck").getStyle(
).setProperty('visibility','hidden');
}

var selectedRect = null;

// Traite les clics sur les rectangles de la légende
function selectZone(rect) {
	var scale = document.svgMap.getSVGDocument().getDocumentElement().getCurrentScale();
	var redPath = document.svgMap.getSVGDocument().getElementById("redpath");

	// Cas où le clic s'effectue sur le rectangle sélectionné
	if (rect == selectedRect) {
		selectedRect = null;
		rect.style.setProperty("stroke-width", 1);
		rect.style.setProperty("stroke", "black");
		redPath.style.setProperty('visibility', 'hidden');

	// Cas où il n'y avait pas de rectangle sélectionné avant
	} else if (selectedRect == null) {
		selectedRect = rect;
		rect.style.setProperty("stroke-width", 3);
		rect.style.setProperty("stroke", "red");

		var groupe =
document.svgMap.getSVGDocument().getElementById(rect.parentNode.getId());
		var d = "";
		var path = null;
		for (var i=0 ; i<groupe.childNodes.length ; i++) {
			path = groupe.childNodes.item(i);
			if (path.nodeType == 1) {
				d += path.getAttribute("d");
			}
		}
		redPath.style.setProperty('stroke-width', 10/scale)
		redPath.setAttribute("d", d);
		redPath.style.setProperty('visibility', 'visible');

	// Cas où l'on change le rectangle sélectionné
	} else {
		selectedRect.style.setProperty("stroke-width", 1);
```

```
                selectedRect.style.setProperty("stroke", "black");

                selectedRect = rect;

                rect.style.setProperty("stroke-width", 3);
                rect.style.setProperty("stroke", "red");

                var groupe =
document.svgMap.getSVGDocument().getElementById(rect.parentNode.getId());
                var d = "";
                var path = null;
                for (var i=0 ; i<groupe.childNodes.length ; i++) {
                        path = groupe.childNodes.item(i);
                        if (path.nodeType == 1) {
                                d += path.getAttribute("d");
                        }
                }
                redPath.style.setProperty('stroke-width', 10/scale)
                redPath.setAttribute("d", d);
                redPath.style.setProperty('visibility', 'visible');
        }
}

// Affiche le règlement au format PDF
function reglementPDF() {
        var
pdfWindow=open("servlet/maquettefinale.ServletReglement","fiche","top=50,left=200,width=700,he
ight=600");
        pdfWindow.focus();
}

// Affiche la carte A3 au format PDF
function cartePDF() {
        var
pdfWindow=open("servlet/maquettefinale.ServletCarteA3","fiche","top=50,left=200,width=700,heig
ht=600");
        pdfWindow.focus();
}

// Affiche la fiche parcellaire au format PDF
function ficheParcelle(numParcelle, zone, secteur) {
        formWindow.close();
        var doc = document.svgMap.getSVGDocument();
        var parc = doc.getElementById(numParcelle);
        if (parc==null) {
                alert("Cette parcelle n'existe pas");
        } else {
                var minX=1000000;
                var minY=1000000;
                var maxX=0;
                var maxY=0;

                var path = parc.getAttribute("d");

                searchValue=new RegExp("M|L|Z|\n|\r","g");
                searchValue.global = true;
                path = path.replace(searchValue, "");

                searchValue=new RegExp(" +","g");
                searchValue.global = true;
                path = path.replace(searchValue, " ");
```

```
                var coords = path.split(" ");
                for (var i=0;i<coords.length;i=i+2) {
                        var x = new Number();
                        var y = new Number();
                        x = eval(coords[i]);
                        y = eval(coords[i+1]);
                        if (x<minX) minX=x;
                        if (x>maxX) maxX=x;
                        if (y<minY) minY=y;
                        if (y>maxY) maxY=y;
                }

                var width = maxX-minX;
                var height = maxY-minY;
                var viewBox = "" + (minX-width/2) + " " + (minY-height/2) + " " + (width*2) + "
" + (height*2);

                if (secteur=="")
                        secteur=zone;

                var pdfWindow=open("servlet/maquettefinale.ServletFiche?numParcelle=" +
numParcelle + "&viewBox=" + viewBox + "&typeZone=" + zone + "&typeSecteur=" +
secteur,"fiche","toolbar=0,directories=0,menu=0,scrollbars=1,location=0,resizable=1,status=0,t
op=50,left=200,width=700,height=600,menubar=0,hotkeys=0");
                pdfWindow.focus();
        }
}

// Zoom sur une parcelle
function zoomParcelle(text) {
        var doc = document.svgMap.getSVGDocument();
        var parc = doc.getElementById(text);
        if (parc==null) {
                alert("Cette parcelle n'existe pas");
        } else {
                var minX=1000000;
                var minY=1000000;
                var maxX=0;
                var maxY=0;

                var path = parc.getAttribute("d");

                searchValue=new RegExp("M|L|Z|\n|\r","g");
                searchValue.global = true;
                path = path.replace(searchValue, "");

                searchValue=new RegExp(" +","g");
                searchValue.global = true;
                path = path.replace(searchValue, " ");

                var coords = path.split(" ");
                for (var i=0;i<coords.length;i=i+2) {
                        var x = new Number();
                        var y = new Number();
                        x = eval(coords[i]);
                        y = eval(coords[i+1]);
                        if (x<minX) minX=x;
                        if (x>maxX) maxX=x;
                        if (y<minY) minY=y;
                        if (y>maxY) maxY=y;
                }
```

```
                var width = maxX-minX;
                var height = maxY-minY;

                doc.getDocumentElement().setCurrentScale(1);
                doc.getDocumentElement().currentTranslate.x=0;
                doc.getDocumentElement().currentTranslate.y=0;
                doc.getDocumentElement().setAttribute("viewBox", "" + (minX-width/2) + " " +
(minY-height/2) + " " + (width*2) + " " + (height*2));

                var outerStroke = doc.getElementById("OuterStroke");
                outerStroke.style.setProperty('stroke-width', '0.2');
                var innerStroke = doc.getElementById("InnerStroke");
                innerStroke.style.setProperty('stroke-width', '0.2');
                innerStroke.style.setProperty('stroke-dasharray', '0.4,0.4');

                var bluePath = doc.getElementById("bluepath");
                bluePath.setAttribute("d", parc.getAttribute("d"));
                bluePath.style.setProperty('visibility', 'visible');
                var guide = document.svgMiniMap.getSVGDocument().getElementById("guide");
                guide.style.setProperty("visibility", "hidden");
        }
}

// Retour à la vue normale
function vueNormale() {
        var doc = document.svgMap.getSVGDocument();
        doc.getDocumentElement().setCurrentScale(1);
        doc.getDocumentElement().currentTranslate.x=0;
        doc.getDocumentElement().currentTranslate.y=0;
        doc.getDocumentElement().setAttribute("viewBox", "2500 1000 2400 1800");

        var outerStroke = doc.getElementById("OuterStroke");
        outerStroke.style.setProperty('stroke-width', '10');
        var innerStroke = doc.getElementById("InnerStroke");
        innerStroke.style.setProperty('stroke-width', '10');
        innerStroke.style.setProperty('stroke-dasharray', '20,20');

        var bluePath = doc.getElementById("bluepath");
        bluePath.style.setProperty('visibility', 'hidden');

        var guide = document.svgMiniMap.getSVGDocument().getElementById("guide");
        guide.style.setProperty("visibility", "visible");
}


// Gestion de la vue miniature

function changeMinimap(evt) {
        var docelem = document.svgMap.getSVGDocument().getDocumentElement();
        var guide = document.svgMiniMap.getSVGDocument().getElementById("guide");
        guide.setAttribute("x", new String((3000 -
docelem.currentTranslate.x*4/docelem.currentScale)/8400*200));
        guide.setAttribute("y", new String((1200 -
docelem.currentTranslate.y*4/docelem.currentScale)/6300*150));
        guide.setAttribute("width", new String(2400/8400*200/docelem.currentScale));
        guide.setAttribute("height", new String(1800/6300*150/docelem.currentScale));
}

var pointeX=0;
var pointeY=0;
var press=0;
```

```
var guideorX=0;
var guideorY=0;

function DoOnMouseDown(evt) {
	var guide = document.svgMiniMap.getSVGDocument().getElementById("guide");
	press=1;
	pointeX = parseInt(evt.getClientX());
	pointeY = parseInt(evt.getClientY());
	guideorX = parseInt(guide.getAttribute("x"));
	guideorY = parseInt (guide.getAttribute("y"));
}


function DoOnMouseMove(evt) {
	var guide = document.svgMiniMap.getSVGDocument().getElementById("guide");
	if (press==1) {
		pointeXnew=parseInt(evt.getClientX());
		pointeYnew=parseInt(evt.getClientY());
		decalX=pointeXnew-pointeX;
		decalY=pointeYnew-pointeY;
		guideorX=decalX+ guideorX;
		guideorY=decalY+ guideorY;
		guide.setAttribute("x",guideorX);
		guide.setAttribute("y",guideorY);
		pointeX=parseInt(pointeXnew);
		pointeY=parseInt(pointeYnew);
	}
}


function DoOnMouseUp(evt) {
	var docelem = document.svgMap.getSVGDocument().getDocumentElement();
	press=0;
	docelem.currentTranslate.x = eval((3000-guideorX/200*8400)/4*docelem.currentScale);
	docelem.currentTranslate.y = eval((1200-guideorY/150*6300)/4*docelem.currentScale);
}

function DoOnMouseOut(evt) {
	press=0;
}
```

3.4 SEMIOLOGIE DU P.O.S. (MAPSTYLE.CSS)

```
.NB {fill:rgb(215,200,10)}
.ND {fill:rgb(50,160,45)}
.INA {fill:rgb(250,175,165)}
.IINA {fill:rgb(250,175,165)}
.UD {fill:rgb(255,190,120)}
.UA {fill:rgb(166,0,0)}
.UI {fill:rgb(130,25,135)}
.UY {fill:rgb(130,25,135)}
.ZI {fill:rgb(130,25,135)}
.UC {fill:rgb(255,125,40)}
.UB {fill:rgb(255,25,0)}
.NC {fill:rgb(155,210,85)}

.NBa {fill:rgb(215,200,10)}
```

```
.NBb {fill:rgb(215,200,10)}
.NDa {fill:rgb(50,160,45)}
.NDc {fill:rgb(50,160,45)}
.NDi {fill:rgb(50,160,45)}
.UDa {fill:rgb(255,190,120)}
.UAa {fill:rgb(166,0,0)}
.UIa {fill:rgb(130,25,135)}
.UIb {fill:rgb(130,25,135)}
.UIc {fill:rgb(130,25,135)}
.UIi {fill:rgb(130,25,135)}
.UCa {fill:rgb(255,125,40)}
.UBa {fill:rgb(255,25,0)}
.IINAa {fill:rgb(250,175,165)}
.IINAb {fill:rgb(250,175,165)}
.INAa {fill:rgb(250,175,165)}
.INAb {fill:rgb(250,175,165)}
.INAL {fill:rgb(250,175,165)}
```

4. GENERATION D'UNE PAGE HTML CONTENANT UN ARTICLE DU REGLEMENT

4.1 LA SERVLET (SERVLETARTICLE.JAVA)

```
package maquettefinale;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

public class ServletArticle extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
  private StreamSource xmlSource;
  private Transformer xslTransformer;
  private String contextPath;

  /**Initialiser les variables globales*/
  public void init() throws ServletException {
    contextPath = this.getServletContext().getRealPath("");
    try {
      TransformerFactory tFactory = TransformerFactory.newInstance();
      xslTransformer = tFactory.newTransformer(new StreamSource(contextPath +
"/article.xsl"));
      xmlSource = new StreamSource(contextPath + "/reglement.xml");
    } catch (Exception e) {e.printStackTrace();}
  }

  /**Traiter la requête HTTP Get*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    String typeZone = request.getParameter("typeZone");
    String typeSecteur = request.getParameter("typeSecteur");
    String numArticle = request.getParameter("numArticle");

    if (typeSecteur == null || typeSecteur.equals("null")) {
      typeSecteur = typeZone;
    }

    // Fixe les paramètres
    xslTransformer.setParameter("typeZone", (typeZone==null)?"null":typeZone);
    xslTransformer.setParameter("typeSecteur", (typeSecteur==null)?"null":typeSecteur);
    xslTransformer.setParameter("numArticle", (numArticle==null)?"null":numArticle);

    // Effectue la transformation et renvoie la réponse
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    try {
      xslTransformer.transform(xmlSource, new StreamResult(out));
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
```

```
  /**Nettoyer les ressources*/
  public void destroy() {
  }
}
```

4.2 LA TRANSFORMATION XSLT (ARTICLE.XSL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

	<!-- Déclaration des paramètres -->
	<xsl:param name="typeZone">UA</xsl:param>
	<xsl:param name="typeSecteur">UA</xsl:param>
	<xsl:param name="numArticle">0</xsl:param>

	<!-- Règle principale -->
	<xsl:template match="/">
		<html>
			<head>
				<title>Informations</title>
			</head>
			<body>
				<p align="center" style="font-weight:bold">Informations</p>
				<p style="font-weight:bold">Zone : <xsl:value-of
select="$typeZone"/></p>

				<xsl:if test="$typeSecteur != $typeZone">
					<p style="font-weight:bold">Secteur : <xsl:value-of
select="$typeSecteur"/></p>
				</xsl:if>

				<xsl:choose>
					<xsl:when test="$numArticle='0'">
						<p style="font-weight:bold">Description
générale</p>
					</xsl:when>
					<xsl:otherwise>
						<p style="font-weight:bold">Article <xsl:value-of
select="$numArticle"/> : <xsl:value-of
select="//zone[@type=$typeZone]/article[@numero=$numArticle]/@titre"/></p>
					</xsl:otherwise>
				</xsl:choose>

				<xsl:choose>
					<xsl:when test="$numArticle='0'">
						<xsl:apply-templates
select="//zone[@type=$typeZone]/description/paragraphe[secteur/@type=$typeSecteur or
not(secteur)]/texte"/>
					</xsl:when>
					<xsl:otherwise>
						<xsl:apply-templates
select="//zone[@type=$typeZone]/article[@numero=$numArticle]/paragraphe[secteur/@type=$typeSec
teur or not(secteur)]/texte"/>
					</xsl:otherwise>
				</xsl:choose>
			</body>
		</html>
	</xsl:template>

	<!-- Règle pour mettre en forme les éléments "texte" -->
```

```
        <xsl:template match="texte">
                <p><xsl:apply-templates/></p>
        </xsl:template>

</xsl:stylesheet>
```

5. GENERATION D'UNE ARBORESCENCE POUR NAVIGUER DANS LE REGLEMENT

5.1 LA SERVLET (SERVLETARBO.JAVA)

```
package maquettefinale;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

public class ServletArbo extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
  private StreamSource xmlSource;
  private Transformer xslTransformer;
  private String contextPath;

  /**Initialiser les variables globales*/
  public void init() throws ServletException {
    contextPath = this.getServletContext().getRealPath("");
    try {
      TransformerFactory tFactory = TransformerFactory.newInstance();
      xslTransformer = tFactory.newTransformer(new StreamSource(contextPath +
"/arborescence.xsl"));
      xmlSource = new StreamSource(contextPath + "/reglement.xml");
    } catch (Exception e) {e.printStackTrace();}
  }

  /**Traiter la requête HTTP Get*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String typeZone = request.getParameter("typeZone");
    String typeSecteur = request.getParameter("typeSecteur");
    String numArticle = request.getParameter("numArticle");

    // Fixe les paramètres
    xslTransformer.setParameter("typeZone", (typeZone==null)?"null":typeZone);
    xslTransformer.setParameter("typeSecteur", (typeSecteur==null)?"null":typeSecteur);
    xslTransformer.setParameter("numArticle", (numArticle==null)?"null":numArticle);

    // Effectue la transformation et renvoie la réponse
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    try {
      xslTransformer.transform(xmlSource, new StreamResult(out));
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**Nettoyer les ressources*/
  public void destroy() {
  }
}
```

5.2 LA TRANSFORMATION XSLT (ARBORESCENCE.XSL)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

	<!-- Déclaration des paramètres -->
	<xsl:param name="typeZone">UA</xsl:param>
	<xsl:param name="typeSecteur">null</xsl:param>
	<xsl:param name="numArticle">0</xsl:param>

	<!-- règle globale -->
	<xsl:template match="/">
		<html>
			<script>
				function arboPage (typeZone, typeSecteur, numArticle) {
					<![CDATA[parent.frames['menu'].location.href =
"maquettefinale.ServletArbo?typeZone=" + typeZone + "&typeSecteur=" + typeSecteur +
"&numArticle=" + numArticle;]]>
				}
				function infoPage (typeZone, typeSecteur, numArticle) {

	<![CDATA[infoWindow=open("maquettefinale.ServletArticle?typeZone=" + typeZone +
"&typeSecteur=" + typeSecteur + "&numArticle=" +
numArticle,"info","toolbar=0,directories=0,menu=0,scrollbars=1,location=0,resizable=1,status=0
,top=50,left=200,width=700,height=600,menubar=0,hotkeys=0");]]>
					infoWindow.focus();
				}
			</script>
			<body text="black" link="black" vlink="black" alink="black">
				<table border="0" cellpadding="0" cellspacing="0" width="100%">
					<tr valign="top"><td>
							<img src="../images/menu_new_root.gif"
align="left" border="0" vspace="0" hspace="0"/>
							Règlement
					</td></tr>
					<xsl:apply-templates select="//zone"/>
				</table>
			</body>
		</html>
	</xsl:template>

	<!-- règle concernant les éléments <zone> -->
	<xsl:template match="zone">
		<xsl:choose>
			<!-- Cas de la zone sélectionnée -->
			<xsl:when test="$typeZone=@type">
				<xsl:call-template name="afficheZone">
					<xsl:with-param name="zoneLien">null</xsl:with-param>
					<xsl:with-param
name="imageCorner">../images/menu_corner_minus.gif</xsl:with-param>
					<xsl:with-param
name="imageTee">../images/menu_tee_minus.gif</xsl:with-param>
					<xsl:with-param
name="imageFolder">../images/menu_folder_open.gif</xsl:with-param>
				</xsl:call-template>
				<xsl:apply-templates select="description"/>
				<xsl:apply-templates select="article"/>
			</xsl:when>
			<!-- Cas des autres zones -->
			<xsl:otherwise>
```

```
                                <xsl:call-template name="afficheZone">
                                        <xsl:with-param name="zoneLien">
                                                <xsl:value-of select="@type"/>
                                        </xsl:with-param>
                                        <xsl:with-param
name="imageCorner">../images/menu_corner_plus.gif</xsl:with-param>
                                        <xsl:with-param
name="imageTee">../images/menu_tee_plus.gif</xsl:with-param>
                                        <xsl:with-param
name="imageFolder">../images/menu_folder_closed.gif</xsl:with-param>
                                </xsl:call-template>
                        </xsl:otherwise>
                </xsl:choose>
        </xsl:template>

        <!-- Affichage de la ligne correspondant à une zone -->
        <xsl:template name="afficheZone">
                <xsl:param name="zoneLien"/>
                <xsl:param name="imageCorner"/>
                <xsl:param name="imageTee"/>
                <xsl:param name="imageFolder"/>

                <tr valign="top">
                        <td>
                                <a onmouseout="window.status='';return true;">
                                        <xsl:attribute
name="href">javascript:arboPage('<xsl:value-of select="$zoneLien"/>', 'null',
'null');</xsl:attribute>
                                        <xsl:attribute name="onmouseover">window.status="Zone
<xsl:value-of select="@type"/>";return true;</xsl:attribute>
                                        <img align="left" border="0" vspace="0" hspace="0"
width="18" height="18">
                                                <xsl:attribute name="src">
                                                        <xsl:choose>
                                                                <xsl:when test="position()=last()">
                                                                        <xsl:value-of
select="$imageCorner"/>
                                                                </xsl:when>
                                                                <xsl:otherwise>
                                                                        <xsl:value-of
select="$imageTee"/>
                                                                </xsl:otherwise>
                                                        </xsl:choose>
                                                </xsl:attribute>
                                        </img>
                                        <img align="left" border="0" vspace="0" hspace="0"
width="18" height="18">
                                                <xsl:attribute name="src">
                                                        <xsl:value-of select="$imageFolder"/>
                                                </xsl:attribute>
                                        </img>Zone <xsl:value-of select="@type"/>
                                </a>
                        </td>
                </tr>
        </xsl:template>

        <!-- règle concernant l'élément <description> -->
        <xsl:template match="description">
                <xsl:choose>
                        <xsl:when test="$numArticle='0'">
                                <xsl:call-template name="afficheDescription">
                                        <xsl:with-param name="numArticleLien">null</xsl:with-
param>
```

```
                                        <xsl:with-param
name="imageTee">../images/menu_tee_minus.gif</xsl:with-param>
                                        <xsl:with-param
name="imageFolder">../images/menu_folder_open.gif</xsl:with-param>
                                        <xsl:with-param name="texteLien">
                                                <b>Description</b>
                                        </xsl:with-param>
                                </xsl:call-template>

                                <xsl:if test="paragraphe/secteur">
                                        <xsl:call-template name="listeSecteurs"/>
                                </xsl:if>
                        </xsl:when>

                        <xsl:otherwise>
                                <xsl:call-template name="afficheDescription">
                                        <xsl:with-param name="numArticleLien">0</xsl:with-param>
                                        <xsl:with-param
name="imageTee">../images/menu_tee_plus.gif</xsl:with-param>
                                        <xsl:with-param
name="imageFolder">../images/menu_folder_closed.gif</xsl:with-param>
                                        <xsl:with-param name="texteLien">Description</xsl:with-
param>
                                </xsl:call-template>
                        </xsl:otherwise>
                </xsl:choose>
        </xsl:template>

        <!-- Affichage de la ligne correspondant à une description -->
        <xsl:template name="afficheDescription">
                <xsl:param name="numArticleLien"/>
                <xsl:param name="imageTee"/>
                <xsl:param name="imageFolder"/>
                <xsl:param name="texteLien"/>

                <tr valign="top">
                        <td>
                                <xsl:choose>
                                        <xsl:when test=".. = ../../zone[last()]">
                                                <img src="../images/menu_pixel.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <img src="../images/menu_bar.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                                        </xsl:otherwise>
                                </xsl:choose>
                                <xsl:choose>
                                        <xsl:when test="paragraphe/secteur">
                                                <a onmouseout="window.status='';return true;">
                                                        <xsl:attribute
name="href">javascript:arboPage("<xsl:value-of select="$typeZone"/>", "null", "<xsl:value-of
select="$numArticleLien"/>");</xsl:attribute>
                                                        <xsl:attribute
name="onmouseover">window.status="Description de la zone <xsl:value-of
select="$typeZone"/>";return true;</xsl:attribute>
                                                        <img align="left" border="0" vspace="0"
hspace="0" width="18" height="18">
                                                                <xsl:attribute name="src">
                                                                        <xsl:value-of
select="$imageTee"/>
                                                                </xsl:attribute>
                                                        </img>
```

```
                                                <img align="left" border="0" vspace="0"
hspace="0" width="18" height="18">
                                                        <xsl:attribute name="src">
                                                                <xsl:value-of
select="$imageFolder"/>
                                                        </xsl:attribute>
                                                </img>Description</a>
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <img src="../images/menu_tee.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                <a onmouseout="window.status='';return true;">
                                                        <xsl:attribute
name="href">javascript:arboPage("<xsl:value-of select="$typeZone"/>", "null",
"0");</xsl:attribute>
                                                        <xsl:attribute
name="onclick">javascript:infoPage("<xsl:value-of select="$typeZone"/>", "null",
"0");</xsl:attribute>
                                                        <xsl:attribute
name="onmouseover">window.status="Description de la zone <xsl:value-of
select="$typeZone"/>";return true;</xsl:attribute>
                                                        <img src="../images/menu_link_local.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                        <xsl:copy-of select="$texteLien"/>
                                                </a>
                                        </xsl:otherwise>
                                </xsl:choose>
                        </td>
                </tr>
        </xsl:template>

        <!-- Règle concernant l'élément <article> -->
        <xsl:template match="article">
                <xsl:choose>
                        <!-- Cas de l'article sélectionné -->
                        <xsl:when test="$numArticle=@numero">
                                <xsl:call-template name="afficheArticle">
                                        <xsl:with-param name="numArticleLien">null</xsl:with-
param>
                                        <xsl:with-param
name="imageCorner">../images/menu_corner_minus.gif</xsl:with-param>
                                        <xsl:with-param
name="imageTee">../images/menu_tee_minus.gif</xsl:with-param>
                                        <xsl:with-param
name="imageFolder">../images/menu_folder_open.gif</xsl:with-param>
                                        <xsl:with-param name="texteLien">
                                                <b>Article <xsl:value-of select="@numero"/></b>
                                        </xsl:with-param>
                                </xsl:call-template>

                                <xsl:if test="paragraphe/secteur">
                                        <xsl:choose>
                                                <xsl:when test="position()=last()">
                                                        <xsl:call-template name="listeSecteurs">
                                                                <xsl:with-param
name="lastArticle">true</xsl:with-param>
                                                        </xsl:call-template>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                        <xsl:call-template name="listeSecteurs">
                                                                <xsl:with-param
name="lastArticle">false</xsl:with-param>
                                                        </xsl:call-template>
                                                </xsl:otherwise>
                                        </xsl:choose>
```

```
                        </xsl:if>
                    </xsl:when>

                    <!-- Cas des autres articles -->
                    <xsl:otherwise>
                        <xsl:call-template name="afficheArticle">
                            <xsl:with-param name="numArticleLien">
                                <xsl:value-of select="@numero"/>
                            </xsl:with-param>
                            <xsl:with-param
name="imageCorner">../images/menu_corner_plus.gif</xsl:with-param>
                            <xsl:with-param
name="imageTee">../images/menu_tee_plus.gif</xsl:with-param>
                            <xsl:with-param
name="imageFolder">../images/menu_folder_closed.gif</xsl:with-param>
                            <xsl:with-param name="texteLien">Article <xsl:value-of
select="@numero"/></xsl:with-param>
                        </xsl:call-template>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:template>

            <!-- Affichage de la ligne correspondant à un article -->
            <xsl:template name="afficheArticle">
                <xsl:param name="numArticleLien"/>
                <xsl:param name="imageCorner"/>
                <xsl:param name="imageTee"/>
                <xsl:param name="imageFolder"/>
                <xsl:param name="texteLien"/>

                <tr valign="top">
                    <td>
                        <xsl:choose>
                            <xsl:when test=".. = ../../zone[last()]">
                                <img src="../images/menu_pixel.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                            </xsl:when>
                            <xsl:otherwise>
                                <img src="../images/menu_bar.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                            </xsl:otherwise>
                        </xsl:choose>
                        <xsl:choose>
                            <xsl:when test="paragraphe/secteur">
                                <a onmouseout="window.status='';return true;">
                                    <xsl:attribute
name="href">javascript:arboPage("<xsl:value-of select="$typeZone"/>", "null", "<xsl:value-of
select="$numArticleLien"/>");</xsl:attribute>
                                    <xsl:attribute
name="onmouseover">window.status="<xsl:value-of select="@titre"/> dans la zone <xsl:value-of
select="$typeZone"/>";return true;</xsl:attribute>
                                    <img align="left" border="0" vspace="0"
hspace="0" width="18" height="18">
                                        <xsl:attribute name="src">
                                            <xsl:choose>
                                                <xsl:when
test="position()=last()">
                                                    <xsl:value-of
select="$imageCorner"/>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                    <xsl:value-of
select="$imageTee"/>
                                                </xsl:otherwise>
```

```
                                                                        </xsl:choose>
                                                                </xsl:attribute>
                                                        </img>
                                                        <img align="left" border="0" vspace="0"
hspace="0" width="18" height="18">
                                                                <xsl:attribute name="src">
                                                                        <xsl:value-of
select="$imageFolder"/>
                                                                </xsl:attribute>
                                                        </img>Article <xsl:value-of
select="@numero"/>
                                                </a>
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <xsl:choose>
                                                        <xsl:when test="position()=last()">
                                                                <img
src="../images/menu_corner.gif" align="left" border="0" vspace="0" hspace="0" width="18"
height="18"/>
                                                        </xsl:when>
                                                        <xsl:otherwise>
                                                                <img src="../images/menu_tee.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                        </xsl:otherwise>
                                                </xsl:choose>
                                                <a onmouseout="window.status='';return true;">
                                                        <xsl:attribute
name="href">javascript:arboPage("<xsl:value-of select="$typeZone"/>", "null", "<xsl:value-of
select="@numero"/>");</xsl:attribute>
                                                        <xsl:attribute
name="onclick">javascript:infoPage("<xsl:value-of select="$typeZone"/>", "null", "<xsl:value-
of select="@numero"/>");</xsl:attribute>
                                                        <xsl:attribute
name="onmouseover">window.status="<xsl:value-of select="@titre"/> dans la zone <xsl:value-of
select="$typeZone"/>";return true;</xsl:attribute>
                                                        <img src="../images/menu_link_local.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                        <xsl:copy-of select="$texteLien"/>
                                                </a>
                                        </xsl:otherwise>
                                </xsl:choose>
                        </td>
                </tr>
        </xsl:template>

        <!-- Affichage de la liste des secteurs -->
        <xsl:template name="listeSecteurs">
                <xsl:param name="lastArticle">false</xsl:param>

                <tr valign="top">
                        <td>
                                <xsl:choose>
                                        <xsl:when test=".. = ../../zone[last()]">
                                                <img src="../images/menu_pixel.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <img src="../images/menu_bar.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                                        </xsl:otherwise>
                                </xsl:choose>
                                <xsl:choose>
                                        <xsl:when test="$lastArticle = 'true'">
                                                <img src="../images/menu_pixel.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
```

```
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <img src="../images/menu_bar.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                                        </xsl:otherwise>
                                </xsl:choose>
                                <img src="../images/menu_tee.gif" align="left" border="0"
vspace="0" hspace="0" width="18" height="18"/>
                                <a onmouseout="window.status='';return true;">
                                        <xsl:attribute
name="href">javascript:arboPage("<xsl:value-of select="$typeZone"/>", "<xsl:value-of
select="$typeZone"/>", "<xsl:value-of select="$numArticle"/>");</xsl:attribute>
                                        <xsl:attribute
name="onclick">javascript:infoPage("<xsl:value-of select="$typeZone"/>", "null", "<xsl:value-
of select="$numArticle"/>");</xsl:attribute>
                                        <xsl:attribute name="onmouseover">window.status="Partie
générale";return true;</xsl:attribute>
                                        <img src="../images/menu_link_local.gif" align="left"
border="0" vspace="0" hspace="0" width="18" height="18"/>
                                        <xsl:choose>
                                                <xsl:when test="$typeSecteur=$typeZone">
                                                        <b>Général</b>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                        Général
                                                </xsl:otherwise>
                                        </xsl:choose>
                                </a>
                        </td>
                </tr>

                <xsl:for-each select="../description//secteur">
                        <tr valign="top">
                                <td>
                                        <xsl:choose>
                                                <xsl:when test="../../.. =
../../../../zone[last()]">
                                                        <img src="../images/menu_pixel.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                        <img src="../images/menu_bar.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                </xsl:otherwise>
                                        </xsl:choose>
                                        <xsl:choose>
                                                <xsl:when test="$lastArticle = 'true'">
                                                        <img src="../images/menu_pixel.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                        <img src="../images/menu_bar.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                </xsl:otherwise>
                                        </xsl:choose>
                                        <xsl:choose>
                                                <xsl:when test="..=../../paragraphe[last()]">
                                                        <img src="../images/menu_corner.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                        <img src="../images/menu_tee.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                </xsl:otherwise>
```

```
                                        </xsl:choose>
                                        <a onmouseout="window.status='';return true;">
                                                <xsl:attribute
name="href">javascript:arboPage("<xsl:value-of select="$typeZone"/>", "<xsl:value-of
select="@type"/>", "<xsl:value-of select="$numArticle"/>");</xsl:attribute>
                                                <xsl:attribute
name="onclick">javascript:infoPage("<xsl:value-of select="$typeZone"/>", "<xsl:value-of
select="@type"/>", "<xsl:value-of select="$numArticle"/>");</xsl:attribute>
                                                <xsl:attribute
name="onmouseover">window.status="Secteur <xsl:value-of select="@type"/>";return
true;</xsl:attribute>
                                                <img src="../images/menu_link_local.gif"
align="left" border="0" vspace="0" hspace="0" width="18" height="18"/>
                                                <xsl:choose>
                                                        <xsl:when test="$typeSecteur=@type">
                                                                <b><xsl:value-of
select="@type"/></b>
                                                        </xsl:when>
                                                        <xsl:otherwise>
                                                                <xsl:value-of select="@type"/>
                                                        </xsl:otherwise>
                                                </xsl:choose>
                                        </a>
                                </td>
                        </tr>
                </xsl:for-each>
        </xsl:template>
</xsl:stylesheet>
```

6. GENERATION DE LA LEGENDE DE LA CARTE

6.1 LA SERVLET (SERVLETLEGENDE.JAVA)

```
package maquettefinale;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

public class ServletLegende extends HttpServlet {
  private static final String CONTENT_TYPE = "image/svg-xml";
  private StreamSource xmlSource;
  private Transformer xslTransformer;
  private String contextPath;

  /**Initialiser les variables globales*/
  public void init() throws ServletException {
    contextPath = this.getServletContext().getRealPath("");
    try {
      TransformerFactory tFactory = TransformerFactory.newInstance();
      xslTransformer = tFactory.newTransformer(new StreamSource(contextPath +
"/legende.xsl"));
      xmlSource = new StreamSource(contextPath + "/pos.svg");
    } catch (Exception e) {e.printStackTrace();}
  }

  /**Traiter la requête HTTP Get*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    // Effectue la transformation et renvoie la réponse
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();

    try {
      xslTransformer.transform(xmlSource, new StreamResult(out));
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**Nettoyer les ressources*/
  public void destroy() {
  }
}
```

6.2 LA TRANSFORMATION XSLT (LEGENDE.XSL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
       <xsl:output method="xml" encoding="ISO-8859-1"/>
```

```
	<!-- Règle principale -->
	<xsl:template match="/">
		<xsl:processing-instruction name="xml-stylesheet">href="../mapstyle.css"
type="text/css"</xsl:processing-instruction>
		<svg id="leg" width="200" height="300" viewBox="0 0 200 300"
zoomAndPan="disable" externalResourcesRequired="true"
xmlns:xlink="http://www.w3.org/1999/xlink">

			<text id="labelAffichage" x="10" y="20" style="font-family:Arial;font-
size:20;">Affichage</text>

			<text id="zones" x="25" y="45" style="font-family:Arial;font-
size:14;">Zones</text>
			<g id="zone_uncheck" style="fill:white;stroke:black;stroke-width:1"
visibility="hidden" onclick="showZones()">
				<rect x="10" y="35" width="10" height="10"/>
			</g>

			<g id="zone_check" style="fill:white;stroke:black;stroke-width:1"
onclick="hideZones()">
				<rect x="10" y="35" width="10" height="10"/>
				<polyline points="12,41 14,43 18,37"/>
			</g>

			<text id="parcelles" x="25" y="65" style="font-family:Arial;font-
size:14;">Parcelles</text>
			<g id="parcelle_uncheck" style="fill:white;stroke:black;stroke-
width:1;visibility:visible" onclick="showParcelles()">
				<rect x="10" y="55" width="10" height="10"/>
			</g>

			<g id="parcelle_check" style="fill:white;stroke:black;stroke-
width:1;visibility:hidden" onclick="hideParcelles()">
				<rect x="10" y="55" width="10" height="10"/>
				<polyline points="12,61 14,63 18,57"/>
			</g>

			<text id="labelControle" x="110" y="20" style="font-family:Arial;font-
size:20;">Contrôle</text>

			<text id="zones" x="125" y="45" style="font-family:Arial;font-
size:14;">Zones</text>
			<g id="zone_radio_uncheck" style="fill:white;stroke:black;stroke-
width:1" visibility="hidden" onclick="activeZones()">
				<circle cx="115" cy="40" r="5"/>
			</g>

			<g id="zone_radio_check" style="fill:white;stroke:black;stroke-width:1">
				<circle cx="115" cy="40" r="5"/>
				<circle cx="115" cy="40" r="3" style="fill:black"/>
			</g>

			<text id="parcelles" x="125" y="65" style="font-family:Arial;font-
size:14;">Parcelles</text>
			<g id="parcelle_radio_uncheck" style="fill:white;stroke:black;stroke-
width:1;visibility:visible" onclick="activeParcelles()">
				<circle cx="115" cy="60" r="5"/>
			</g>

			<g id="parcelle_radio_check" style="fill:white;stroke:black;stroke-
width:1;visibility:hidden">
				<circle cx="115" cy="60" r="5"/>
				<circle cx="115" cy="60" r="3" style="fill:black"/>
			</g>
```

```
                        <line id="separateur1" style="stroke:darkgray" x1="100" y1="5" x2="100"
y2="75"/>
                        <line id="separateur2" style="stroke:darkgray" x1="5" y1="75" x2="195"
y2="75"/>


                        <text id="labelLegende" x="60" y="95" style="font-family:Arial;font-
size:20;">Légende</text>

                        <g id="legende">
                                <xsl:apply-templates select="//g[@id='Zones']//g"/>
                        </g>

                        <line id="separateur3" style="stroke:darkgray" x1="5" y1="275" x2="195"
y2="275"/>

                        <a xlink:href="reglement.doc" target="docwindow">
                                <text id="liendoc" x="30" y="295" style="font-family:Arial;font-
size:14;">Règlement Word (.doc)</text>
                        </a>

                        <rect id="contour" x="0" y="0" width="200" height="300"
style="stroke:black;fill:none"/>
                </svg>
        </xsl:template>

        <!-- Règle permettant de créer les éléments de la légende -->
        <xsl:template match="g">
                <g>
                        <xsl:attribute name="id">
                                <xsl:value-of select="@id"/>
                        </xsl:attribute>

                        <!-- Création du rectangle de la couleur voulue -->
                        <rect width="20" height="10" style="stroke-width:1;stroke:black"
onclick="selectZone(evt.target)">
                                <xsl:attribute name="class">
                                        <xsl:value-of select="@class"/>
                                </xsl:attribute>
                                <xsl:attribute name="x">
                                        <xsl:choose>
                                                <xsl:when test="starts-with(@id,'Zones')">
                                                        <xsl:value-of select="5 + 65 *
floor((position()-1) div 11)"/>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                        <xsl:value-of select="10 + 65 *
floor((position()-1) div 11)"/>
                                                </xsl:otherwise>
                                        </xsl:choose>
                                </xsl:attribute>
                                <xsl:attribute name="y">
                                        <xsl:value-of select="105 + 15 * ((position()-1) mod
11)"/>
                                </xsl:attribute>
                        </rect>

                        <!-- Création du texte -->
                        <text style="font-family:Arial;font-size:14;">
                                <xsl:attribute name="x">
                                        <xsl:choose>
                                                <xsl:when test="starts-with(@id,'Zones')">
```

```
                                                <xsl:value-of select="30 + 65 *
floor((position()-1) div 11)"/>
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <xsl:value-of select="35 + 65 *
floor((position()-1) div 11)"/>
                                        </xsl:otherwise>
                                </xsl:choose>
                        </xsl:attribute>
                        <xsl:attribute name="y">
                                <xsl:value-of select="115 + 15 * ((position()-1) mod
11)"/>
                        </xsl:attribute>
                        <xsl:value-of select="@class"/>
                </text>
        </g>
    </xsl:template>
</xsl:stylesheet>
```

7. GENERATION D'UNE FICHE PARCELLAIRE

7.1 LA SERVLET (SERVLETFICHE.JAVA)

```
package maquettefinale;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.xml.sax.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import org.apache.fop.apps.Driver;

public class ServletFiche extends HttpServlet {
  private static final String CONTENT_TYPE = "application/pdf";
  private StreamSource xmlSource, svgSource;
  private Transformer xslTransformer, xslsvgTransformer, xslsvglegTransformer;
  private String contextPath;

  /**Initialiser les variables globales*/
  public void init() throws ServletException {
    contextPath = this.getServletContext().getRealPath("");
    try {
      TransformerFactory tFactory = TransformerFactory.newInstance();
      xslsvgTransformer = tFactory.newTransformer(new StreamSource(contextPath +
"/parcelle_svg-fo.xsl"));
      xslsvglegTransformer = tFactory.newTransformer(new StreamSource(contextPath + "/legende-
fo.xsl"));
      svgSource = new StreamSource("webapps/WebApp/pos.svg");
      xslTransformer = tFactory.newTransformer(new StreamSource(contextPath +
"/fiche_parcelle-fo.xsl"));
      xmlSource = new StreamSource(contextPath + "/reglement.xml");
    } catch (Exception e) {e.printStackTrace();}
  }

  /**Traiter la requête HTTP Get*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    if (request.getParameter("viewBox") != null)
      xslsvgTransformer.setParameter("viewBox", request.getParameter("viewBox"));

    if (request.getParameter("numParcelle") != null) {
      xslsvgTransformer.setParameter("numParcelle", request.getParameter("numParcelle"));
      xslTransformer.setParameter("numParcelle", request.getParameter("numParcelle"));
    }

    if (request.getParameter("typeZone") != null)
      xslTransformer.setParameter("typeZone", request.getParameter("typeZone"));

    if (request.getParameter("typeSecteur") != null)
      xslTransformer.setParameter("typeSecteur", request.getParameter("typeSecteur"));

    response.setContentType(CONTENT_TYPE);
    OutputStream out = response.getOutputStream();
    StringWriter bufWriter = new StringWriter();
```

```
    try {
         // Création de la carte à inclure dans le document
         File mapFile = File.createTempFile("map",".svg",new File(contextPath));
         FileWriter fw = new FileWriter(mapFile);

      xslsvgTransformer.transform(svgSource, new StreamResult(fw));
      fw.close();

         // Création de la légende à inclure dans le document
         File legFile = File.createTempFile("leg",".svg",new File(contextPath));
         fw = new FileWriter(legFile);

      xslsvglegTransformer.transform(svgSource, new StreamResult(fw));
      fw.close();

      // Fixe les paramètres
      xslTransformer.setParameter("svg", mapFile.getAbsolutePath());
      xslTransformer.setParameter("svgleg", legFile.getAbsolutePath());
      xslTransformer.transform(xmlSource, new StreamResult(bufWriter));
      bufWriter.flush();

      // Effectue la transformation et renvoie la réponse
      StringReader bufReader = new StringReader(bufWriter.toString());
      Driver driver = new Driver(new InputSource(bufReader), out);
      driver.setRenderer(Driver.RENDER_PDF);
      driver.run();
         mapFile.delete();
         legFile.delete();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**Nettoyer les ressources*/
  public void destroy() {
  }
}
```

7.2 LA TRANSFORMATION XSLT GENERANT LE DOCUMENT FO (FICHE_PARCELLE-FO.XSL)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
       <xsl:output method="xml"/>

       <!-- Déclaration des paramètres -->
       <xsl:param name="svg"/>
       <xsl:param name="svgleg"/>
       <xsl:param name="typeZone"/>
       <xsl:param name="typeSecteur"/>
       <xsl:param name="numParcelle"/>

       <!-- Déclaration des styles -->
       <xsl:attribute-set name="default">
```

```
                <xsl:attribute name="font-size">12pt</xsl:attribute>
                <xsl:attribute name="font-weight">normal</xsl:attribute>
                <xsl:attribute name="font-style">normal</xsl:attribute>
                <xsl:attribute name="text-align">justify</xsl:attribute>
                <xsl:attribute name="color">black</xsl:attribute>
                <xsl:attribute name="space-before">0.2cm</xsl:attribute>
        </xsl:attribute-set>
        <xsl:attribute-set name="titre">
                <xsl:attribute name="font-size">32pt</xsl:attribute>
                <xsl:attribute name="color">blue</xsl:attribute>
                <xsl:attribute name="font-weight">bold</xsl:attribute>
                <xsl:attribute name="text-align">center</xsl:attribute>
                <xsl:attribute name="space-before">2cm</xsl:attribute>
        </xsl:attribute-set>
        <xsl:attribute-set name="sousTitre">
                <xsl:attribute name="font-size">28pt</xsl:attribute>
                <xsl:attribute name="color">blue</xsl:attribute>
                <xsl:attribute name="font-weight">normal</xsl:attribute>
                <xsl:attribute name="text-align">center</xsl:attribute>
                <xsl:attribute name="space-before">1cm</xsl:attribute>
                <xsl:attribute name="space-after">0.5cm</xsl:attribute>
        </xsl:attribute-set>
        <xsl:attribute-set name="titreArticle">
                <xsl:attribute name="font-size">16pt</xsl:attribute>
                <xsl:attribute name="color">black</xsl:attribute>
                <xsl:attribute name="font-weight">bold</xsl:attribute>
                <xsl:attribute name="space-before">1cm</xsl:attribute>
        </xsl:attribute-set>

        <!-- Règle principale -->
        <xsl:template match="/">
                <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns="http://www.w3.org/2000/svg">

                        <!-- Mise en page générale -->
                        <fo:layout-master-set>
                                <fo:simple-page-master master-name="contents" page-
height="29.7cm" page-width="21cm">
                                        <fo:region-before extent="1cm"/>
                                        <fo:region-after extent="1cm"/>
                                        <fo:region-start extent="1cm"/>
                                        <fo:region-end extent="1cm"/>
                                        <fo:region-body margin-bottom="2cm" margin-top="2cm"
margin-left="2cm" margin-right="2cm"/>
                                </fo:simple-page-master>

                                <fo:page-sequence-master master-name="contents-Seq">
                                        <fo:repeatable-page-master-reference master-
reference="contents"/>
                                </fo:page-sequence-master>
                        </fo:layout-master-set>

                        <!-- Contenu du document -->
                        <fo:page-sequence master-reference="contents-Seq">
                                <fo:static-content flow-name="xsl-region-before">
                                </fo:static-content>
                                <fo:static-content flow-name="xsl-region-after">
                                        <fo:block font-size="10pt" font-family="Helvetica" text-
align="center">
                                                <fo:page-number/>
                                        </fo:block>
                                </fo:static-content>
```

```
                        <fo:flow flow-name="xsl-region-body">
                                <fo:block xsl:use-attribute-sets="titre">
                                        Fiche parcellaire
                                </fo:block>
                                <fo:block xsl:use-attribute-sets="sousTitre">
                                        Parcelle n°<xsl:value-of select="$numParcelle"/>,
Zone <xsl:value-of select="$typeZone"/>
                                        <xsl:if test="$typeSecteur!=$typeZone">
                                                , Secteur <xsl:value-of
select="$typeSecteur"/>
                                        </xsl:if>
                                </fo:block>
                                <fo:block space-before="1cm" space-after="1cm" text-
align="center">
                                        <fo:external-graphic>
                                                <xsl:attribute name="src"><xsl:value-of
select="$svg"/></xsl:attribute>
                                        </fo:external-graphic>
                                        <fo:external-graphic>
                                                <xsl:attribute name="src"><xsl:value-of
select="$svgleg"/></xsl:attribute>
                                        </fo:external-graphic>
                                </fo:block>
                                <fo:block>
                                        <xsl:apply-templates
select="//zone[@type=$typeZone]/description"/>
                                        <xsl:apply-templates
select="//zone[@type=$typeZone]/article"/>
                                </fo:block>
                        </fo:flow>
                </fo:page-sequence>
        </fo:root>
    </xsl:template>

    <!-- Règle concernant l'élément "description" -->
    <xsl:template match="description">
            <xsl:apply-templates select="paragraphe[secteur/@type=$typeSecteur or
not(secteur)]/texte"/>
    </xsl:template>

    <!-- Règle concernant les éléments "article" -->
    <xsl:template match="article">
            <fo:block xsl:use-attribute-sets="titreArticle">
                    Article <xsl:value-of select="@numero"/> : <xsl:value-of
select="@titre"/>
            </fo:block>
            <xsl:apply-templates select="paragraphe[secteur/@type=$typeSecteur or
not(secteur)]/texte"/>
    </xsl:template>

    <!-- Règle concernant les éléments "texte" -->
    <xsl:template match="texte">
            <fo:block xsl:use-attribute-sets="default">
                    <xsl:apply-templates/>
            </fo:block>
    </xsl:template>

</xsl:stylesheet>
```

7.3 GENERATION DE LA CARTE SVG A INCLURE DANS LE DOCUMENT (PARCELLE_SVG-FO.XSL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
	<xsl:output method="xml" encoding="ISO-8859-1"/>

	<!-- Déclaration des paramètres -->
	<xsl:param name="viewBox"/>
	<xsl:param name="numParcelle"/>

	<!-- Règle principale -->
	<xsl:template match="/">
		<xsl:processing-instruction name="xml-stylesheet">href="mapstyle.css"
type="text/css"</xsl:processing-instruction>
		<xsl:apply-templates select="/svg"/>
	</xsl:template>

	<!-- Règle traitant l'élément racine "svg" -->
	<xsl:template match="svg">
		<xsl:copy>
			<xsl:attribute name="width">8cm</xsl:attribute>
			<xsl:attribute name="height">6cm</xsl:attribute>
			<xsl:attribute name="viewBox"><xsl:value-of
select="$viewBox"/></xsl:attribute>
			<xsl:apply-templates select="//g[@id!='cosmetic']//path"/>

			<path style="stroke-width:0.1;stroke:blue;fill:none;stroke-
linejoin:round">
				<xsl:attribute name="d">
					<xsl:value-of select="//path[@id=$numParcelle]/@d"/>
				</xsl:attribute>
			</path>
		</xsl:copy>
	</xsl:template>

	<!-- Règle traitant les éléments "path" -->
	<xsl:template match="path">
		<xsl:copy>
			<xsl:copy-of select="@d"/>
			<xsl:choose>
				<xsl:when test="../@id='Parcelles'">
					<xsl:attribute name="style">
						stroke-width:0.1;stroke-
linejoin:round;stroke:rgb(102,102,102);fill:none
					</xsl:attribute>
				</xsl:when>
				<xsl:otherwise>
					<xsl:attribute name="class">
						<xsl:value-of select="../@class"/>
					</xsl:attribute>
					<xsl:attribute name="style">
						stroke-width:0.1;stroke-
linejoin:round;stroke:black
					</xsl:attribute>
				</xsl:otherwise>
			</xsl:choose>
		</xsl:copy>
	</xsl:template>

</xsl:stylesheet>
```

7.4 GENERATION DE LA LEGENDE A INCLURE DANS LA FICHE (LEGENDE-FO.XSL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
	<xsl:output method="xml" encoding="ISO-8859-1"/>

	<!-- Règle principale -->
	<xsl:template match="/">
		<xsl:processing-instruction name="xml-stylesheet">href="mapstyle.css"
type="text/css"</xsl:processing-instruction>
		<svg id="leg-fo" width="4cm" height="6cm" viewBox="0 0 200 300">

			<text id="labelLegende" x="60" y="30" style="font-family:Arial;font-
size:20">Légende</text>

			<g id="legende">
				<xsl:apply-templates select="//g[@id='Zones']//g"/>
			</g>

			<rect id="contour" x="0" y="0" width="200" height="300"
style="stroke:black;fill:none"/>
		</svg>
	</xsl:template>

	<!-- Règle permettant de créer les éléments de la légende -->
	<xsl:template match="g">
		<g>
			<xsl:attribute name="id">
				<xsl:value-of select="@id"/>
			</xsl:attribute>

			<!-- Création du rectangle -->
			<rect width="20" height="10" style="stroke-width:1;stroke:black">
				<xsl:attribute name="class">
					<xsl:value-of select="@class"/>
				</xsl:attribute>
				<xsl:attribute name="x">
					<xsl:choose>
						<xsl:when test="starts-with(@id,'Zones')">
							<xsl:value-of select="20 + 80 *
floor((position()-1) div 15)"/>
						</xsl:when>
						<xsl:otherwise>
							<xsl:value-of select="30 + 80 *
floor((position()-1) div 15)"/>
						</xsl:otherwise>
					</xsl:choose>
				</xsl:attribute>
				<xsl:attribute name="y">
					<xsl:value-of select="60 + 15 * ((position()-1) mod
15)"/>
				</xsl:attribute>
			</rect>

			<!-- Création du texte -->
			<text style="font-family:Arial;font-size:14;">
				<xsl:attribute name="x">
					<xsl:choose>
						<xsl:when test="starts-with(@id,'Zones')">
```

```
                                                <xsl:value-of select="45 + 80 *
floor((position()-1) div 15)"/>
                                        </xsl:when>
                                        <xsl:otherwise>
                                                <xsl:value-of select="55 + 80 *
floor((position()-1) div 15)"/>
                                        </xsl:otherwise>
                                </xsl:choose>
                        </xsl:attribute>
                        <xsl:attribute name="y">
                                <xsl:value-of select="70 + 15 * ((position()-1) mod
15)"/>
                        </xsl:attribute>
                        <xsl:value-of select="@class"/>
                </text>
        </g>
    </xsl:template>
</xsl:stylesheet>
```

8. GENERATION DU REGLEMENT EN PDF

8.1 LA SERVLET (SERVLETREGLEMENT.JAVA)

```
package maquettefinale;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.xml.sax.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import org.apache.fop.apps.Driver;

public class ServletReglement extends HttpServlet {
  private static final String CONTENT_TYPE = "application/pdf";
  private StreamSource xmlSource, svgSource;
  private Transformer xslTransformer, xslsvgTransformer, xslsvglegTransformer;
  private String contextPath;

  /**Initialiser les variables globales*/
  public void init() throws ServletException {
    contextPath = this.getServletContext().getRealPath("");
    try {
      TransformerFactory tFactory = TransformerFactory.newInstance();
      xslsvgTransformer = tFactory.newTransformer(new StreamSource(contextPath + "/carte_svg-
fo.xsl"));
      xslsvglegTransformer = tFactory.newTransformer(new StreamSource(contextPath + "/legende-
fo.xsl"));
      svgSource = new StreamSource(contextPath + "/pos.svg");
      xslTransformer = tFactory.newTransformer(new StreamSource(contextPath + "/reglement-
fo.xsl"));
      xmlSource = new StreamSource(contextPath + "/reglement.xml");
    } catch (Exception e) {e.printStackTrace();}
  }

  /**Traiter la requête HTTP Get*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    response.setContentType(CONTENT_TYPE);
    OutputStream out = response.getOutputStream();
    StringWriter bufWriter = new StringWriter();

    try {
         // Création de la carte à inclure dans le document
         File mapFile = File.createTempFile("map",".svg",new File(contextPath));
         FileWriter fw = new FileWriter(mapFile);

      xslsvgTransformer.transform(svgSource, new StreamResult(fw));
      fw.close();

         // Création de la légende à inclure dans le document
         File legFile = File.createTempFile("leg",".svg",new File(contextPath));
         fw = new FileWriter(legFile);

      xslsvglegTransformer.transform(svgSource, new StreamResult(fw));
      fw.close();
```

```
      // Fixe les paramètres
      xslTransformer.setParameter("svg", mapFile.getAbsolutePath());
      xslTransformer.setParameter("svgleg", legFile.getAbsolutePath());
      xslTransformer.transform(xmlSource, new StreamResult(bufWriter));
      bufWriter.flush();

      // Effectue la transformation et renvoie la réponse
      StringReader bufReader = new StringReader(bufWriter.toString());
      Driver driver = new Driver(new InputSource(bufReader), out);
      driver.setRenderer(Driver.RENDER_PDF);
      driver.run();
         mapFile.delete();
         legFile.delete();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**Nettoyer les ressources*/
  public void destroy() {
  }
}
```

8.2 GENERATION DU DOCUMENT FO (REGLEMENT-FO.XSL)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
       <xsl:output method="xml"/>

       <!-- Déclaration des paramètres -->
       <xsl:param name="svg"/>
       <xsl:param name="svgleg"/>

       <!-- Déclaration des styles -->
       <xsl:attribute-set name="default">
              <xsl:attribute name="font-size">12pt</xsl:attribute>
              <xsl:attribute name="font-weight">normal</xsl:attribute>
              <xsl:attribute name="font-style">normal</xsl:attribute>
              <xsl:attribute name="text-align">justify</xsl:attribute>
              <xsl:attribute name="color">black</xsl:attribute>
              <xsl:attribute name="space-before">0.2cm</xsl:attribute>
       </xsl:attribute-set>
       <xsl:attribute-set name="titre">
              <xsl:attribute name="font-size">32pt</xsl:attribute>
              <xsl:attribute name="color">blue</xsl:attribute>
              <xsl:attribute name="font-weight">bold</xsl:attribute>
              <xsl:attribute name="text-align">center</xsl:attribute>
              <xsl:attribute name="space-before">2cm</xsl:attribute>
              <xsl:attribute name="space-after">2cm</xsl:attribute>
       </xsl:attribute-set>
       <xsl:attribute-set name="sousTitre">
              <xsl:attribute name="font-size">28pt</xsl:attribute>
              <xsl:attribute name="color">blue</xsl:attribute>
              <xsl:attribute name="font-weight">normal</xsl:attribute>
              <xsl:attribute name="text-align">center</xsl:attribute>
```

```
		<xsl:attribute name="space-before">2cm</xsl:attribute>
		<xsl:attribute name="space-after">0.5cm</xsl:attribute>
	</xsl:attribute-set>
	<xsl:attribute-set name="titreArticle">
		<xsl:attribute name="font-size">16pt</xsl:attribute>
		<xsl:attribute name="color">black</xsl:attribute>
		<xsl:attribute name="font-weight">bold</xsl:attribute>
		<xsl:attribute name="space-before">1cm</xsl:attribute>
	</xsl:attribute-set>

	<!-- Règle principale -->
	<xsl:template match="/">
		<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns="http://www.w3.org/2000/svg">

			<!-- Mise en page générale -->
			<fo:layout-master-set>
				<fo:simple-page-master master-name="contents" page-
height="29.7cm" page-width="21cm">
					<fo:region-before extent="1cm"/>
					<fo:region-after extent="1cm"/>
					<fo:region-start extent="1cm"/>
					<fo:region-end extent="1cm"/>
					<fo:region-body margin-bottom="2cm" margin-top="2cm"
margin-left="2cm" margin-right="2cm"/>
				</fo:simple-page-master>

				<fo:page-sequence-master master-name="contents-Seq">
					<fo:repeatable-page-master-reference master-
reference="contents"/>
				</fo:page-sequence-master>
			</fo:layout-master-set>

			<!-- Contenu du document -->
			<fo:page-sequence master-reference="contents-Seq">
				<fo:static-content flow-name="xsl-region-before">
				</fo:static-content>
				<fo:static-content flow-name="xsl-region-after">
					<fo:block font-size="10pt" font-family="Helvetica" text-
align="center">
						<fo:page-number/>
					</fo:block>
				</fo:static-content>
				<fo:flow flow-name="xsl-region-body">
					<fo:block xsl:use-attribute-sets="titre">
						Règlement
					</fo:block>
					<fo:block text-align="center">
						<fo:external-graphic>
							<xsl:attribute name="src"><xsl:value-of
select="$svg"/></xsl:attribute>
						</fo:external-graphic>
						<fo:external-graphic>
							<xsl:attribute name="src"><xsl:value-of
select="$svgleg"/></xsl:attribute>
						</fo:external-graphic>
					</fo:block>
					<fo:block>
						<xsl:apply-templates select="//zone"/>
					</fo:block>
				</fo:flow>
			</fo:page-sequence>
```

```
		</fo:root>
	</xsl:template>

	<!-- Règle s'appliquant aux éléments "zone" -->
	<xsl:template match="zone">
		<fo:block xsl:use-attribute-sets="sousTitre">
			Zone <xsl:value-of select="@type"/>
		</fo:block>
		<xsl:apply-templates select="description"/>
		<xsl:apply-templates select="article"/>
	</xsl:template>

	<!-- Règle s'appliquant aux éléments "description" -->
	<xsl:template match="description">
		<xsl:apply-templates select="paragraphe/texte"/>
	</xsl:template>

	<!-- Règle s'appliquant aux éléments "article" -->
	<xsl:template match="article">
		<fo:block xsl:use-attribute-sets="titreArticle">
			Article <xsl:value-of select="@numero"/> : <xsl:value-of
select="@titre"/>
		</fo:block>
		<xsl:apply-templates select="paragraphe/texte"/>
	</xsl:template>

	<!-- Règle s'appliquant aux éléments "texte" -->
	<xsl:template match="texte">
		<fo:block xsl:use-attribute-sets="default">
			<xsl:apply-templates/>
		</fo:block>
	</xsl:template>

</xsl:stylesheet>
```

8.3 GENERATION DE LA CARTE A INCLURE DANS LE DOCUMENT (CARTE_SVG-FO.XSL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
	<xsl:output method="xml" encoding="ISO-8859-1"/>

	<!-- Règle principale -->
	<xsl:template match="/">
		<xsl:processing-instruction name="xml-stylesheet">href="mapstyle.css"
type="text/css"</xsl:processing-instruction>
		<xsl:apply-templates select="/svg"/>
	</xsl:template>

	<!-- Règle appliquée à l'élément racine "svg" -->
	<xsl:template match="svg">
		<xsl:copy>
			<xsl:attribute name="width">8cm</xsl:attribute>
			<xsl:attribute name="height">6cm</xsl:attribute>
			<xsl:attribute name="viewBox">2500 1000 2400 1800</xsl:attribute>
			<xsl:apply-templates select="//g[@id='Zones']//path"/>
			<rect id="contour" x="2500" y="1000" width="2400" height="1800"
style="stroke:black;fill:none"/>
		</xsl:copy>
	</xsl:template>
```

```
        <!-- Règle pour mettre en forme les éléments "path" -->
        <xsl:template match="path">
                <xsl:copy>
                        <xsl:copy-of select="@d"/>
                        <xsl:attribute name="class">
                                <xsl:value-of select="../@class"/>
                        </xsl:attribute>
                        <xsl:attribute name="style">
                                stroke-width:0.1;stroke-linejoin:round;stroke:black
                        </xsl:attribute>
                </xsl:copy>
        </xsl:template>

</xsl:stylesheet>
```

8.4 GENERATION DE LA LEGENDE A INCLURE DANS LE DOCUMENT (LEGENDE-FO.XSL)

Voir 6.4.

9. GENERATION DE LA CARTE DU P.O.S. EN PDF AU FORMAT A3

9.1 LA SERVLET (SERVLETCARTEA3.JAVA)

```
package maquettefinale;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.xml.sax.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import org.apache.fop.apps.Driver;

public class ServletCarteA3 extends HttpServlet {
  private static final String CONTENT_TYPE = "application/pdf";
  private StreamSource svgSource, xmlSource;
  private Transformer xslTransformer, xslsvgTransformer, xslsvglegTransformer;
  private String contextPath;

  /**Initialiser les variables globales*/
  public void init() throws ServletException {
    contextPath = this.getServletContext().getRealPath("");
    try {
      TransformerFactory tFactory = TransformerFactory.newInstance();
      xslsvgTransformer = tFactory.newTransformer(new StreamSource(contextPath +
"/carteA3_svg-fo.xsl"));
      xslsvglegTransformer = tFactory.newTransformer(new StreamSource(contextPath +
"/legendeA3_svg-fo.xsl"));
      svgSource = new StreamSource(contextPath + "/pos.svg");
      xslTransformer = tFactory.newTransformer(new StreamSource(contextPath + "/carteA3-
fo.xsl"));
      xmlSource = new StreamSource(contextPath + "/reglement.xml");
    } catch (Exception e) {
        e.printStackTrace();
       }
  }

  /**Traiter la requête HTTP Get*/
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    response.setContentType(CONTENT_TYPE);
    OutputStream out = response.getOutputStream();
    StringWriter bufWriter = new StringWriter();

    try {
         // Création de la carte à inclure dans le document
         File mapFile = File.createTempFile("map",".svg",new File(contextPath));
         FileWriter fw = new FileWriter(mapFile);

      xslsvgTransformer.transform(svgSource, new StreamResult(fw));
      fw.close();

         // Création de la légende à inclure dans le document
         File legFile = File.createTempFile("leg",".svg",new File(contextPath));
         fw = new FileWriter(legFile);
```

```
      xslsvglegTransformer.transform(svgSource, new StreamResult(fw));
      fw.close();

      // Fixe les paramètres
      xslTransformer.setParameter("svg", mapFile.getAbsolutePath());
      xslTransformer.setParameter("svgleg", legFile.getAbsolutePath());
      xslTransformer.transform(xmlSource, new StreamResult(bufWriter));
      bufWriter.flush();

      // Effectue la transformation et renvoie la réponse
      StringReader bufReader = new StringReader(bufWriter.toString());
      Driver driver = new Driver(new InputSource(bufReader), out);
      driver.setRenderer(Driver.RENDER_PDF);
      driver.run();
         mapFile.delete();
         legFile.delete();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**Nettoyer les ressources*/
  public void destroy() {
  }
}
```

9.2 GENERATION DU DOCUMENT FO (CARTEA3-FO.XSL)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
       <xsl:output method="xml"/>

       <!-- Déclaration des paramètres -->
       <xsl:param name="svg"/>
       <xsl:param name="svgleg"/>

       <!-- Règle principale -->
       <xsl:template match="/">
              <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns="http://www.w3.org/2000/svg">

                     <!-- Mise en page générale -->
                     <fo:layout-master-set>
                            <fo:simple-page-master master-name="contents" page-
height="29.7cm" page-width="42cm">
                                   <fo:region-before extent="1cm"/>
                                   <fo:region-after extent="1cm"/>
                                   <fo:region-start extent="1cm"/>
                                   <fo:region-end extent="1cm"/>
                                   <fo:region-body margin-bottom="2cm" margin-top="2cm"
margin-left="2cm" margin-right="2cm"/>
                            </fo:simple-page-master>

                            <fo:page-sequence-master master-name="contents-Seq">
                                   <fo:repeatable-page-master-reference master-
reference="contents"/>
                            </fo:page-sequence-master>
                     </fo:layout-master-set>
```

```
                        <!-- Contenu du document : la carte et la légende -->
                        <fo:page-sequence master-reference="contents-Seq">
                                <fo:flow flow-name="xsl-region-body">
                                        <fo:block text-align="center">
                                                <fo:external-graphic>
                                                        <xsl:attribute name="src"><xsl:value-of
select="$svg"/></xsl:attribute>
                                                </fo:external-graphic>
                                                <fo:external-graphic>
                                                        <xsl:attribute name="src"><xsl:value-of
select="$svgleg"/></xsl:attribute>
                                                </fo:external-graphic>
                                        </fo:block>
                                </fo:flow>
                        </fo:page-sequence>
                </fo:root>
        </xsl:template>

</xsl:stylesheet>
```

9.3 GENERATION DE LA CARTE SVG INCLUSE DANS LE DOCUMENT (CARTEA3_SVG-FO.XSL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
        <xsl:output method="xml" encoding="ISO-8859-1"/>

        <!-- Règle principale -->
        <xsl:template match="/">
                <xsl:processing-instruction name="xml-stylesheet">href="mapstyle.css"
type="text/css"</xsl:processing-instruction>
                <xsl:apply-templates select="/svg"/>
        </xsl:template>

        <!-- Règle appliquée à l'élément racine "svg" -->
        <xsl:template match="svg">
                <xsl:copy>
                        <xsl:attribute name="width">24cm</xsl:attribute>
                        <xsl:attribute name="height">18cm</xsl:attribute>
                        <xsl:attribute name="viewBox">-100 0 8000 6000</xsl:attribute>
                        <xsl:apply-templates select="//g[@id='Zones']//path"/>
                        <rect id="contour" x="-100" y="0" width="8000" height="6000"
style="stroke:black;fill:none"/>
                </xsl:copy>
        </xsl:template>

        <!-- Règle appliquée aux éléments "path" -->
        <xsl:template match="path">
                <xsl:copy>
                        <xsl:copy-of select="@d"/>
                        <xsl:attribute name="class">
                                <xsl:value-of select="../@class"/>
                        </xsl:attribute>
                        <xsl:attribute name="style">
                                stroke-width:0.1;stroke-linejoin:round;stroke:black
                        </xsl:attribute>
                </xsl:copy>
        </xsl:template>
```

```
</xsl:stylesheet>
```

9.4 GENERATION DE LA LEGENDE INCLUSE DANS LE DOCUMENT (LEGENDEA3_SVG-FO.XSL)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
	<xsl:output method="xml" encoding="ISO-8859-1"/>

	<!-- Règle principale -->
	<xsl:template match="/">
		<xsl:processing-instruction name="xml-stylesheet">href="mapstyle.css"
type="text/css"</xsl:processing-instruction>
		<svg id="leg-fo" width="12cm" height="18cm" viewBox="0 0 200 300">

			<text id="labelLegende" x="60" y="30" style="font-family:Arial;font-
size:20;">Légende</text>

			<g id="legende">
				<xsl:apply-templates select="//g[@id='Zones']//g"/>
			</g>

			<rect id="contour" x="0" y="0" width="200" height="300"
style="stroke:black;fill:none"/>
		</svg>
	</xsl:template>

	<!-- Règle permettant de créer les éléments de la légende -->
	<xsl:template match="g">
		<g>
			<xsl:attribute name="id">
				<xsl:value-of select="@id"/>
			</xsl:attribute>

			<!-- Création du rectangle -->
			<rect width="20" height="10" style="stroke-width:1;stroke:black">
				<xsl:attribute name="class">
					<xsl:value-of select="@class"/>
				</xsl:attribute>
				<xsl:attribute name="x">
					<xsl:choose>
						<xsl:when test="starts-with(@id,'Zones')">
							<xsl:value-of select="20 + 80 *
floor((position()-1) div 15)"/>
						</xsl:when>
						<xsl:otherwise>
							<xsl:value-of select="30 + 80 *
floor((position()-1) div 15)"/>
						</xsl:otherwise>
					</xsl:choose>
				</xsl:attribute>
				<xsl:attribute name="y">
					<xsl:value-of select="60 + 15 * ((position()-1) mod
15)"/>
				</xsl:attribute>
			</rect>

			<!-- Création du texte -->
			<text style="font-family:Arial;font-size:14;">
				<xsl:attribute name="x">
```

```
                                        <xsl:choose>
                                                <xsl:when test="starts-with(@id,'Zones')">
                                                        <xsl:value-of select="45 + 80 *
floor((position()-1) div 15)"/>
                                                </xsl:when>
                                                <xsl:otherwise>
                                                        <xsl:value-of select="55 + 80 *
floor((position()-1) div 15)"/>
                                                </xsl:otherwise>
                                        </xsl:choose>
                                </xsl:attribute>
                                <xsl:attribute name="y">
                                        <xsl:value-of select="70 + 15 * ((position()-1) mod
15)"/>
                                </xsl:attribute>
                                <xsl:value-of select="@class"/>
                        </text>
                </g>
        </xsl:template>
</xsl:stylesheet>
```

Reprographie : Cete de Lyon ✆ (+33) (0) 4 72 14 30 30 (septembre 2002)
Dépôt légal : 3e trimestre 2002
ISSN : 1263-2570
ISRN : Certu/RE -- 02 - 16-- FR

Certu
9, rue Juliette-Récamier
69456 Lyon cedex 06
✆ (+33) (0) 4 72 74 59 59
Internet http://www.certu.fr