

Institut d'Informatique et Mathématiques Appliquées de Grenoble



**LSR IMAG**



**Equipe SIGMA**

# **L'amélioration du prototype KIWIS**

Mémoire de stage de **Magistère 1ère année**  
**Année universitaire 2001-2002**

Présenté par M. Ioan Marius Bilasco

Encadré par M. Jérôme Gensel

Réalisé au sein de l'équipe SIGMA

Laboratoire LSR IMAG

## **Remerciements**

Je remercie Monsieur Jérôme GENSEL, Maître de Conférence à l'Université Pierre-Mendès France de Grenoble, pour ses conseils lors de l'élaboration de ce travail.

Je remercie Madame Marlène VILLANOVA-OLIVER, doctorante, et Monsieur Hervé MARTIN, Professeur à l'Université Joseph Fourier de Grenoble, pour leur encadrement, leur aide et leur amitié tout au long de l'année.

Je tiens à remercier également toute l'équipe SIGMA du LSR pour cette année passée ensemble.

## Table des matières

<b>Table des matières.....</b>	<b>3</b>
<b>Table des figures.....</b>	<b>6</b>
<b>1. Introduction.....</b>	<b>8</b>
Structure d'accueil.....	8
Equipe d'accueil.....	8
Travail à réaliser.....	9
Plan du mémoire.....	9
<b>2. Le prototype KIWIS (état des lieux).....</b>	<b>11</b>
2.1 Contexte.....	11
2.2 Les Modèles.....	12
2.2.1 Le modèle des données.....	12
2.2.2 Le modèle utilisateur.....	15
2.2.3 Le modèle hypermédia.....	16
2.3 Exploitation des modèles dans KIWIS.....	18
2.4 Implémentation.....	20
2.4.1 Choix techniques.....	20
2.4.2 Plate-forme KIWIS retenue.....	20
2.4.3 La sécurité.....	22
2.4.4 Conclusion.....	22
<b>3. L'étude des technologies mises en œuvre.....</b>	<b>24</b>
3.1 XML.....	24
3.1.2 La norme XML.....	24
3.1.3 XML Schéma.....	25

3.1.4 La spécification XSL.....	26
3.1.7 Conclusion sur XML.....	27
3.2 Les servlets Java.....	27
3.2.4 Le suivi de session .....	29
3.2.4 Les API DOM et SAX .....	31
3.2.5 Conclusion sur Java.....	32
3.3 Conclusion sur Java et XML.....	33
3.4 Les technologies support : le moteur des servlets-Tomcat et l’outil de publication COCOON.....	34
3.4.1 Le moteur des servlets Tomcat .....	34
3.4.2 L’outil de publication Cocoon .....	37
<b>4. Les résultats .....</b>	<b>41</b>
4.1 Réflexions sur les différents modèles proposés dans le prototype KIWIS .....	41
4.1.1 Le modèle des données .....	41
4.1.2 Modèle d’utilisateurs.....	42
4.1.3 Le modèle des fonctionnalités.....	42
4.1.4 Le modèle hypermédia .....	45
4.2 Nouvelle démarche de conception .....	45
4.3 Les aspects de sécurité dans le nouveau prototype .....	46
4.3.1 Modalités d’implantation .....	47
4.4.1 Schéma fonctionnel.....	48
4.4.2 Définition de l’architecture .....	50
4.4.3 Conception du prototype avec WAE.....	51
4.5 L’interface concepteur de KIWIS .....	56
4.5.1 La connexion.....	56

4.5.2 La page du concepteur .....	57
4.5.3 Création d'un nouveau système d'information .....	57
4.5.4 Les pages de définition de modèle de données .....	59
4.5.5 Interface de définition des groupes .....	61
4.5.6 Interface de définition du modèle de fonctionnalités .....	61
4.5.7 Interface de modélisation de la carte de navigation .....	63
4.5.8 La création d'un utilisateur du système .....	64
4.6 L'adaptabilité de l'interface utilisateur .....	65
<b>5. Conclusions et perspectives .....</b>	<b>66</b>
<b>Bibliographie .....</b>	<b>67</b>

## Table des figures

Figure 2.1 : Représentation d'une association de classe en AROM [AROM2000] .....	13
Figure 2.2 : Modèle utilisateur .....	15
Figure 2.3 : Les différents composants d'une page Web .....	18
Figure 2.4 : Traitement d'une requête envoyée à un SI généré par KIWIS .....	19
Figure 2.5 : Architecture logicielle du prototype .....	21
Tableau 2.6 : La sécurité dans KIWIS .....	22
Figure 2.7 : Technologies et outils utilisés dans le prototype .....	23
Figure 3.1 : principes de fonctionnement d'XSLT .....	26
Tableau 3.2 : Points forts des technologies XML présentées dans ce chapitre .....	27
Figure 3.3 : Fonctionnement d'une application Web .....	28
Figure 3.4 : Points remarquables des technologies Java utilisées dans KIWIS .....	33
Figure 3.5 : Interconnexion XML et Java .....	33
Figure 3.6 Implantation de serveur Apache-Tomcat .....	35
Figure 3.7 structure minimale imposée d'une application Web .....	35
Figure 3.8 Diagramme de séquence du traitement d'une requête .....	39
Figure 3.9 le processus de génération de la réponse .....	39
Figure 4.1a Stratification de la fonctionnalité <i>Liste des Crues</i> pour le groupe des <i>géographes</i> .....	43
Figure 4.1b Stratification de la fonctionnalité <i>Liste des Crues</i> pour le groupe <i>public</i> .....	43
Figure 4.2 Navigation au sein d'une fonctionnalité <i>composée</i> (exemple2) .....	44
Figure 4.3 Navigation au sein d'une fonctionnalité <i>composée</i> (exemple2) .....	44
Figure 4.5 : Vue logique de l'architecture de KIWIS-APP .....	51
Figure 4.6 : Diagramme des cas d'utilisations pour l'acteur CONCEPTEUR .....	52
Figure 4.7 : Diagramme des cas d'utilisations pour l'acteur UTILISATEUR .....	53
Figure 4.8 : Diagramme WAE « connexion » du Concepteur .....	54
Figure 4.9 : Diagramme WAE « Définition d'une fonctionnalité » (Concepteur) .....	55
Figure 4.10 Page d'authentification de KIWIS .....	56
Figure 4.11 La page d'accueil du concepteur dans KIWIS .....	57
Figure 4.12 La page de création d'un nouveau SI .....	58
Figure 4.13 Structure d'un SI en KIWIS .....	58
Figure 4.14 Présentation de la démarche de conception .....	59

Figure 4.15 Création d'une nouvelle classe .....	59
Figure 4.16 Création d'une nouvelle classe .....	60
Figure 4.17 La page réponse de création d'une association .....	60
Figure 4.18 La création d'une groupe d'utilisateurs .....	61
Figure 4.19 La première phase de création d'une fonctionnalité.....	62
Figure 4.20 La deuxième étape dans la conception d'une fonctionnalité.....	62
Figure 4.21 Définition de la carte de navigation 1/2 .....	63
Figure 4.22 Définition de la carte de navigation 2/2 .....	63
Figure 4.23 Création d'un nouveau <i>login</i> .....	64

# 1. Introduction

## Structure d'accueil

Créé le 1 janvier 1995, le Laboratoire Logiciels Systèmes Réseaux (LSR) a le statut d'Unité Mixte de Recherche (UMR 5526). Ses tutelles sont le CNRS, l'Institut National Polytechnique de Grenoble et l'Université Joseph Fourier. Le LSR est l'un des sept laboratoires de la fédération de recherche en Informatique et Mathématiques Appliquées de Grenoble (IMAG : FR 0071).

Le LSR organise ses activités autour des axes suivants :

- § Ingénierie des grands logiciels
- § Conception et validation des logiciels
- § Applications de la programmation logique avec contraintes
- § Services base de données sur le réseau
- § Modélisation des systèmes d'information
- § Ingénierie des réseaux et du multimédia

## Equipe d'accueil

L'équipe d'accueil au sein du LSR est l'équipe SIGMA qui réalise des recherches appliquées sur l'ingénierie des systèmes d'information. Ces recherches sont centrées sur la formalisation, la conception et l'organisation des Systèmes d'Information. Les résultats attendus de ces recherches sont des modèles, des langages, des outils et des démarches afin d'appuyer des raisonnements à différents niveaux qui favorisent principalement quatre aspects : présentation multimédia adapté, flexibilité, traçabilité et réutilisation.

Les principaux thèmes de recherche de l'équipe sont :

1. Ingénierie des systèmes d'information,
2. Formalisation, multimodélisation et métamodélisation,
3. Réutilisation et traçabilité,
4. Systèmes d'information multimédias distribués et Web,

## 5. Applications industrielles et systèmes de gestion de données multimédias.

L'équipe SIGMA est organisée autour de 2 axes :

1. axe composant et réutilisation qui explore une approche génie logiciel à base de patrons dans la conception des systèmes d'information
2. axe multimédia qui explore la conception des systèmes d'information multimédia basés sur le Web

Le sujet de ce stage s'inscrit dans ce deuxième axe.

### **Travail à réaliser**

Le but de ce stage est d'améliorer le prototype KIWIS de façon à le faire évoluer vers un outil portable et modulaire, disposant d'une interface conviviale permettant à un concepteur de SIMW, utilisateur de KIWIS, d'appréhender facilement la démarche méthodologique de conception. En d'autres mots, il s'agit accroître l'*utilisabilité* de KIWIS.

Cette amélioration de l'interface nécessite :

- une compréhension du cadre méthodologique (modèles, communication entre ces modèles) et des différentes étapes de conception d'un SIMW implantée dans KIWIS.
- l'étude et la maîtrise des technologies Web mises en œuvre dans KIWIS : serveur Web (configurations des serveurs Apache et Tomcat), Servlets, XML et Cocoon (module de publication des pages Web sur la base d'une transformation de fichiers XML en fichier HTML).

Outre l'aspect technique, une des contributions attendues était une réflexion sur l'organisation des différentes étapes de conception et sur les modèles sous-jacents.

### **Plan du mémoire**

Le mémoire commence par la présentation du contexte de ce stage, le prototype KIWIS, ses modèles, la démarche de conception et différents aspects liés à l'implantation physique de ceux-ci.

Ensuite, un chapitre entier est consacré à l'étude des principales technologies utilisées dans KIWIS (Java et XML), ainsi qu'à une présentation des technologies support utilisées

pour le développement de ce premier prototype : le serveur Apache-Tomcat et l'environnement de publication Cocoon sont décrits.

Le quatrième chapitre présente les résultats obtenus. Dans un premier temps, une réflexion autour des modèles proposés dans KIWIS est menée. Cette étape de réflexion est suivie de la description de la nouvelle démarche de conception proposée, et d'une partie plus technique qui discute les détails liés à la sécurité dans l'ancien et le nouveau prototype. La dernière partie de ce chapitre présente les nouvelles interfaces de conception et les problèmes liés à la construction de celles-ci.

En conclusion, nous proposons un bilan du travail effectué et des résultats obtenus en fonction des objectifs initiaux. Nous présentons également un ensemble de perspectives venant s'inscrire dans la continuité de ce travail.

## 2. Le prototype KIWIS (état des lieux)

### 2.1 Contexte

Avec le développement de systèmes de plus en plus rapides et de plus en plus performants, les informations traitées qui, au départ, n'étaient que textuelles et linéaires, sont maintenant représentées sous forme d'ensembles constitués d'entités atomiques liées entre elles. On parle d'hypertexte. Ce concept fournit une méthode d'accès non séquentielle à l'information, contrairement aux approches traditionnelles. La nature des informations a également évolué. Le concept de multimédia permet de regrouper désormais des données jusqu'alors exploitées séparément : du texte, du son, de l'image, de la vidéo, etc.

Le Web permet d'accéder de façon simple à une multitude d'informations stockées dans des ordinateurs parfois dispersés géographiquement et reliés entre eux par un réseau mondial. Un tel environnement est aussi utilisé dans le cadre d'accès intranet au sein d'un réseau local d'ordinateurs. Le principal avantage d'une telle approche consiste en l'accès simple et standardisé à de nombreuses informations de sources différentes, grâce à des outils qui facilitent ainsi la communication entre les personnes.

Dans ce contexte, et face à l'intérêt grandissant des utilisateurs pour l'Internet, il est apparu nécessaire pour l'équipe SIGMA, de disposer d'une architecture et d'une méthode de conception de systèmes d'information (SI), d'une part, basés sur le Web et, d'autre part, prenant en compte la dimension *adaptabilité*.

L'adaptabilité d'un SI se définit par sa capacité à adapter les informations (contenu), leur forme (présentation, navigation, etc.), ainsi que les services (consultation, modification, etc.) qu'il propose aux besoins et aux caractéristiques individuelles et collectives des utilisateurs.

Le travail de l'axe multimédia de l'équipe SIGMA consiste à proposer à la fois différents modèles d'aide à la conception de Systèmes d'Information Multimédia basés sur le Web (SIMW), et à proposer un outil implantant la méthode. Notamment, l'accent est mis sur l'adaptabilité du SIMW à ses utilisateurs. L'idée est que le SIMW doit fournir non seulement une information pertinente mais que l'accès à celle-ci soit progressif afin d'éviter que l'utilisateur ne se perde dans un espace d'information hypermédia trop dense. C'est pourquoi l'équipe SIGMA propose que la conception d'un SIMW repose sur quatre modèles distincts mais liés : un *modèle de données* chargé de la description du domaine d'application, un *modèle de l'utilisateur* chargé de la description des droits d'accès et des préférences en matière de présentation de l'information, un *modèle pour l'accès progressif* et personnalisé à l'information, et un *modèle de présentation* permettant la personnalisation graphique des pages Web qui sont dynamiquement générées par le SIMW en réponse aux requêtes de l'utilisateur.

Ces spécifications ont servi à l'élaboration du cahier des charges d'un générateur automatique de systèmes d'information multimédia basés sur le Web, appelé KIWIS. KIWIS se présente comme un serveur Web permettant de mettre en œuvre des modèles présentés ci-dessus. Une fois les modèles instanciés, KIWIS se charge du déploiement du serveur Web sur lequel sera accessible le SI ainsi conçu.

Le projet KIWIS (Knowledge for Improving Web Information System) a débuté en septembre 1999 au sein de l'équipe SIGMA<sup>1</sup> du laboratoire LSR (Logiciels, Systèmes et Réseaux) de l'IMAG, avec le démarrage de la thèse de Marlène Villanova. Son objectif est de fournir un ensemble de modèles et une architecture générique pour la conception de systèmes d'information (SI) multimédias adaptables et basés sur le Web. Dans le cadre de sa thèse, entreprise depuis 1999, Marlène Villanova, a proposé une démarche méthodologique de conception de SIMW reposant sur ces quatre modèles. Un premier prototype opérationnel de ce système a été développé par Christelle Erb durant son stage d'ingénieur CNAM (réalisé d'octobre 2000 à octobre 2001) et par Marlène Villanova.

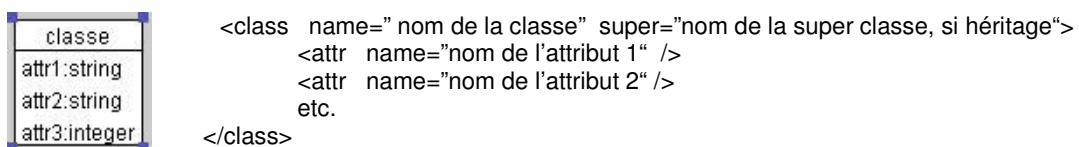
## 2.2 Les Modèles

### 2.2.1 Le modèle des données

Ce modèle correspond au contenu informationnel du SI. La modélisation des données dans KIWIS est basée sur deux concepts essentiels : les *classes*, qui regroupent des entités dont la structure, le comportement et la sémantique sont communs, et des *associations*, qui regroupent des liens entre classes. Classes et associations ont vocation à être instanciées : pour les classes, on parlera d'*instances* et pour les associations, on parlera de *tuples*.

#### § Classes

Une *classe* décrit un ensemble d'objets ayant des propriétés communes, des contraintes communes, et interagissant de la même manière avec les autres objets. Chaque classe est caractérisée par un ensemble d'attributs.



La notation XML [XML] (à droite) indique qu'une classe dans KIWIS a un nom, éventuellement une super-classe, et possède un ensemble d'attributs, lesquels sont identifiés par un nom, un type et un niveau de détail.

Dans KIWIS un objet représente une entité distinguable du domaine considéré. Tout objet est instance d'une seule classe et de ces ancêtres.

```

<object name="nom de l'objet instance" class="nom de la classe concernée">
  <attr1> valeur de l'attribut 1 </attr1>
  <attr2> valeur de l'attribut 2 </attr2>
  etc.
</object>

```

Un *objet* est donc défini par un nom d'instance (deux instances ne pouvant pas porter le même nom) et le nom de la classe à laquelle est attaché l'instance.

## § Associations

Les associations permettent de représenter une liaison entre deux instances ou plus, et peuvent comporter des attributs spécifiques, appelés attributs d'association.

Le rôle d'une association correspond à la connexion entre l'association et la classe à laquelle elle est connectée. Ainsi dans le cas d'une association *assoc* entre les classes *classe1* et *classe2*, le rôle *role1* correspond à la connexion entre *assoc* et *classe1*, et le rôle *role2* correspond à la connexion entre *assoc* et *classe2*.

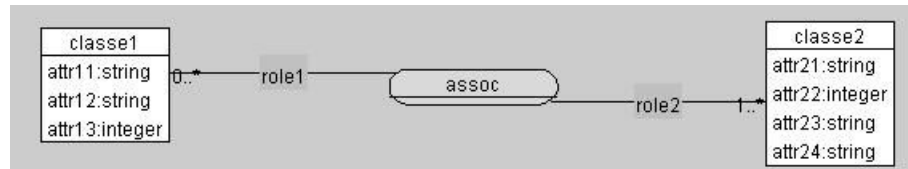


Figure 2.1 : Représentation d'une association de classe en AROM [AROM2000]

```

<assoc name="nom de l'association" class1="nom de la classe1" class2="nom de la classe 2">
  <role1 name="nom du rôle" multiplicity="multiplicité" />
  <role2 name="nom du rôle" multiplicity="multiplicité"/>
</assoc>

```

Un rôle est défini par un nom et une multiplicité. La sémantique de la multiplicité du rôle d'une association dans notre architecture est similaire à celle utilisée dans UML. La multiplicité du rôle *role1* de l'association *assoc* est un intervalle d'entiers [min...max] correspondant respectivement au nombre minimum et maximum d'instances de *classe1* qui peuvent apparaître dans *assoc* pour le rôle *role1* pour tout n-1 uplet d'instances pour les autres rôles.

Un *tuple* est une instance d'association. Chaque tuple est attaché à une seule association et ses ancêtres.

```

<tuple name="nom de l'association">
  <role1> nom de l'objet instance de la classe 1</role1>
  <role2> nom de l'objet instance de la classe 2 </role2>
</tuple>

```

Un tuple est identifié par le nom de l'association à laquelle il est attaché et la valeur des rôles du tuple (obligatoirement le nom d'une instance de la classe associée au rôle).

Pour concevoir un tel modèle, le concepteur de SI peut utiliser les méthodes ou langages traditionnels (OMT, UML, etc.). Il lui faudra toutefois inclure les 2 notions suivantes : les niveaux de détail et les attributs à formats multiples [VILL2001].

#### ¶ Les niveaux de détail

Le *niveau de détail* permet de sélectionner les informations en terme de liste d'attributs de classe qui seront visibles à un niveau donné. Ainsi, pour chaque attribut de la classe le concepteur définira un niveau de détail à partir duquel l'information sera présentée à l'utilisateur du SI.

Cette notion sera utile au modèle utilisateur, pour ne présenter que les informations de tel ou tel niveau selon les droits d'accès définis pour l'utilisateur connecté.

On fixe que le niveau 1 contient l'ensemble des attributs qui identifie la classe de manière significative. Par exemple, le concepteur d'un SI constitué, entre autres, d'une classe *Personne*, elle-même constituée des attributs suivants {nom, prénom, adresse, email, age, etc.}, pourrait affecter "1" comme niveau de détail aux attributs Nom et Prénom, "2" comme niveau de détail à l'attribut Adresse, etc. Comme il est toujours possible que 2 personnes portent le même nom et même prénom, à partir du niveau "1", l'utilisateur sélectionnera le niveau de détail suivant pour retrouver l'adresse de la personne.

#### ¶ Les formats multiples

La seconde notion de notre modèle est la possibilité de définir des *formats multiples* sur une donnée multimédia. Une telle donnée est un attribut particulier pour lequel on dispose de plusieurs contenus informationnels qui sont porteurs chacun d'un peu plus d'informations.

Pour chaque donnée multimédia à formats multiples, on stocke dans le SI plusieurs « formes » pour l'attribut en question.

Par exemple, dans un site de présentation de villes françaises, on pourra modéliser la classe *Ville* de la façon suivante :

```
Classe Ville {  
    nom_ville : nom de la ville à présenter,  
    département : département dans lequel se situe la ville,  
    nb_hab : nombre d'habitants de la ville,  
    présentation_ville : vidéo de présentation de la ville,  
    ...  
}
```

L'exemple d'un attribut 'vidéo' illustre le concept de formats multiples. On pourra disposer d'une photo issue de la vidéo (forme 1) représentant une vue aérienne de la ville, d'un résumé composé des key-frames (forme 2), présentant les principaux monuments de la ville, et enfin de la vidéo complète (forme 3) débutant par la vue aérienne et au cours d'une balade dans la ville, nous présentant les principaux sites touristiques (parmi lesquels se trouvent les principaux monuments). Ce qui sera traduit par notre prototype de la façon suivante (les attributs *level\_detail* correspond aux niveaux des détails associés aux entités):

```
<attribut name="presentation_ville" level_detail="4" format="multiple">  
    <g1 name="photo" level_detail="1" />  
    <g2 name="résumé" level_detail="2" />  
    <g3 name="vidéo" level_detail="3" />  
</attribut>
```

Les différentes valeurs de la donnée multimédia doivent correspondre à un contenu informationnel du plus succinct au plus riche. L'objectif est que l'utilisateur puisse choisir le format qui lui sera présenté pour les données multimédias à formats multiples du SI et naviguer d'un format à l'autre ponctuellement sur une donnée.

Ces concepts permettent de proposer à l'utilisateur un accès progressif à l'information, c'est-à-dire d'accéder à une version plus ou moins détaillée de la donnée, la donnée étant une classe (avec les niveaux de détail) ou un attribut (avec les formats multiples).

Grâce aux niveaux de détail, les utilisateurs naviguent graduellement entre différents niveaux de représentation de données structurées, selon leurs droits, leurs profils et/ou leurs préférences, alors que les formats multiples adaptent la présentation d'une donnée multimédia selon la configuration matérielle, l'intérêt ou encore les contraintes temporelles des utilisateurs.

L'approche à base de niveaux de détail et de formats multiples permet de disposer d'un plus grand nombre de possibilités de consultation des données et d'un accès progressif à l'information. Les différentes options proposées constituent autant de critères de paramétrage du système d'information.

### 2.2.2 Le modèle utilisateur

Pour prendre en compte différents niveaux d'adaptabilité, 2 concepts qui permettent d'organiser et de maintenir les droits et préférences des utilisateurs ont été proposés. Au niveau du *concept de groupe* sont définies toutes les informations pertinentes relatives au groupe d'utilisateurs comme les droits, et au niveau du *concept d'utilisateur* sont spécifiées les préférences des utilisateurs en terme de navigation et de présentation des données.

La figure 2.2 décrit le modèle utilisateur. Les classes 'Utilisateur' et 'Groupe' permettent respectivement de maintenir l'information relative à un utilisateur en tant qu'individu et à un groupe d'utilisateurs. L'appartenance à un ou plusieurs groupes est modélisée par l'association 'appartient'. La variable 'principal' de cette association permet de spécifier le groupe d'appartenance à considérer par défaut pour un individu.

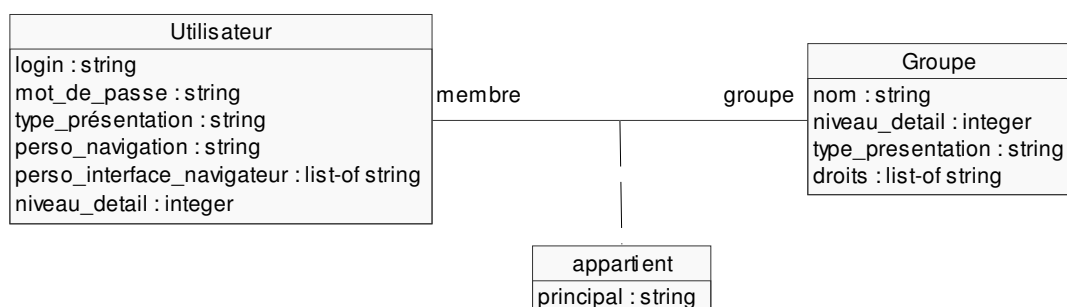


Figure 2.2 : Modèle utilisateur

### **Notion de groupe**

La notion de groupe renferme des informations partagées par plusieurs utilisateurs.

Traditionnellement, les groupes sont définis selon un ensemble de droits. Ces droits dépendent eux-mêmes du statut des utilisateurs ; l'administrateur aura un maximum de droits alors que l'invité en aura un minimum.

De plus, dans KIWIS, un niveau de détail et un type de présentation par défaut sont définis pour un groupe. Le niveau de détail correspond au niveau de visibilité par défaut du contenu informationnel du SI, pour les utilisateurs appartenant à ce groupe. Le type de présentation est un modèle de présentation par défaut choisi pour les utilisateurs membres du groupe. Lorsque ces informations ne sont pas définies au niveau de l'individu, ce sont celles du groupe qui sont prises en compte.

### **Notion de profil**

La notion de profil est complémentaire à celle du groupe. Pour chaque utilisateur, en plus de son groupe d'appartenance, on définit un ensemble de caractéristiques ayant trait aux préférences de celui-ci, qui sont intéressantes dans le cadre de SI multimédias :

- type de présentation
- personnalisation de la navigation
- personnalisation de l'interface du navigateur

Tout d'abord, un utilisateur est identifié par un *login* de connexion et un *mot de passe*, pour gérer l'authentification. L'utilisateur peut personnaliser son profil en désignant un modèle de présentation comme présentation par défaut. Il peut spécifier en terme de navigation, le mode d'activation des liens (dans la même fenêtre ou dans une nouvelle) et le chargement automatique ou non des images, des vidéos. Au niveau de l'interface du navigateur, il peut personnaliser l'apparence des fenêtres (présence/absence des barres d'outil, de menu, de défilement). Enfin, comme au niveau d'un groupe, l'utilisateur peut définir un niveau de détail par défaut (masque sur le contenu informationnel du SI), à partir duquel l'information lui sera présentée.

Le concepteur d'un SIW peut définir un ensemble de groupes et d'utilisateurs, et leur attribuer un certain nombre de droits. L'utilisateur peut ensuite personnaliser son profil, au fil de sa consultation du SIW.

### **2.2.3 Le modèle hypermédia**

Un modèle hypermédia permet de décrire les éléments hypermédia qui seront publiés sur le site. Une page de site Web est considérée comme un élément hypermédia dont la

description consiste en l'utilisation de deux sous-modèles : un modèle de navigation et un modèle de présentation.

### Modèle de navigation

L'étape principale de la définition d'un tel modèle pour un SIW est la définition des « unités de navigation » : c'est-à-dire la définition des fonctionnalités du système (il s'agit des fonctionnalités 'consulter la liste des objets du domaine d'information', 'ajouter un nouvel objet dans le domaine d'information', etc.)

Pour chacune des fonctionnalités, il s'agit pour chaque groupe d'utilisateurs, de définir le contenu informationnel de la page Web associée à cette fonctionnalité (liste des attributs de classe visibles), les liens de navigation vers d'autres pages (fonctionnalités du système accessibles comme l'activation de filtres, la mise à jour du profil, etc..) et de raccrocher cette unité de navigation à la structure du site (depuis quelle page, cette unité sera-t-elle appelée).

### Modèle de présentation

Le modèle de présentation permet d'exprimer la composition et l'apparence physique des pages Web. Dans KIWIS, chaque unité de navigation (du modèle de navigation) correspond à une page.

Deux dimensions relatives à la présentation sont traitées :

#### □ Charte graphique

Une charte graphique est un document qui contient les règles de création des pages d'un site afin d'en assurer la cohérence. Elle doit permettre aux visiteurs, grâce à son homogénéité, de reconnaître, sans même avoir à lire le contenu des pages, qu'ils sont toujours sur le même site.

En règle générale, on définit un modèle de page. C'est-à-dire une grille standard dans laquelle toutes les pages s'inscrivent. Cela consiste à identifier plusieurs zones qui rempliront différentes fonctions et qui auront une apparence homogène sur toutes les pages du site.

Pour une page Web, les 4 régions suivantes ont été identifiées (voir figure 2.3): une entête, une barre horizontale de navigation, une barre verticale d'index et un corps. Pour chacune d'elles le concepteur peut définir une couleur, une hauteur, une largeur et une position absolue, etc. Le concepteur peut également personnaliser les composants atomiques identifiés que sont les titres, les sous-titres, les cellules de tableau et qui seront contenus dans les différentes régions.

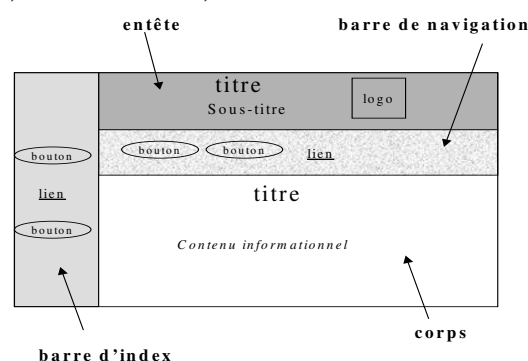


Figure 2.3 : Les différents composants d'une page Web

L'objectif de la charte graphique est de définir l'apparence des 4 régions et de tous les composants pouvant être présents dans une page Web. Ces informations sont codées dans une feuille de style Cascading Style Sheet [CSS].

#### □ Modèle de présentation

Dans un deuxième temps, le concepteur peut définir des présentations.

Une présentation est une page Web composée au minimum d'une région « corps », elle-même composée au moins d'une zone texte (titre, zone textuelle libre ou contenu informationnel du SIW sous forme tableau ou cascade). La figure 4.5 montre les différents éléments de composition d'une page Web :

- une zone d'entête peut contenir au plus un titre, un sous-titre et un logo, et au minimum un titre si cette zone existe,
- une barre de navigation (horizontale) est composée de boutons et/ou de liens,
- une barre d'index (verticale) est composée de boutons et/ou de liens,
- un corps contient le contenu informationnel principal sous forme « tableau » ou « page » et pouvant contenir un titre, une zone de texte libre ou encore une *applet*, et au minimum un élément de type texte ou une *applet*.

Lors de la création d'une nouvelle présentation, le concepteur sélectionne pour chaque région choisie, un ou plusieurs composants, en sachant qu'une page est au moins constituée d'un corps. Concernant l'affichage du contenu informationnel, 2 modèles ont été proposés: l'un en tableau (toutes les instances d'une classe sont présentées dans un tableau) et l'autre en cascade (une instance de classe par page). Le concepteur associe également une charte graphique à cette présentation, soit définie par lui-même, soit celle proposée par défaut.

## 2.3 Exploitation des modèles dans KIWIS

Cette section montre les relations entre les différents modèles, lors du traitement d'une requête utilisateur (description du rôle des différents « Manager » dans le chapitre 6).

Le système KIWIS renvoie une page XML (eXtensible Markup Language) et une page XSL (eXtensible Stylesheet Language) [XSLT], en réponse à une requête. Du point de vue de l'utilisateur il s'agit d'une traduction de la représentation d'une requête en une information hypermédia. (voir figure 2.4)

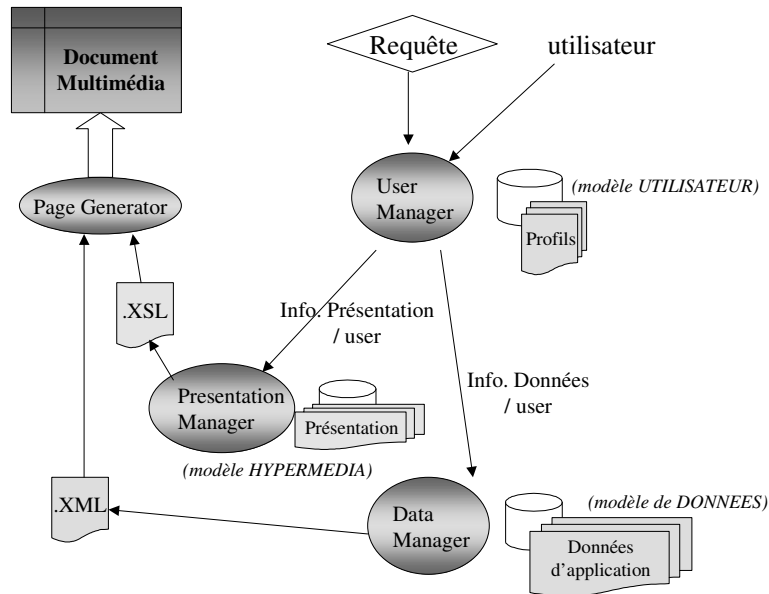


Figure 2.4 : Traitement d'une requête envoyée à un SI généré par KIWIS

Une requête est soumise au système par un utilisateur ( ). Le « User Manager » recherche les informations relatives à l'individu en terme de présentation et en terme d'accès aux données dans la base de données des utilisateurs ( ) et les transmet respectivement aux modules « Presentation Manager » et « DataManager » ( ).

Le module « PresentationManager » recherche dans sa base de données des présentations ( ) les informations correspondant aux caractéristiques de l'utilisateur et génère le fichier de présentation (format XSL) ( ).

Le module « Data Manager » sélectionne, dans l'ensemble des données d'application du SI, les données correspondant à la requête formulée et aux droits de l'utilisateur ( ). Il génère le fichier de données (format XML) correspondant ( ).

Ces 2 fichiers (le fichier de présentation et le fichier de données) sont envoyés par le système au « Page Generator » qui génère le document multimédia final qui est renvoyé à l'utilisateur ( ).

On remarque ici l'intérêt d'utiliser des technologies comme XML qui séparent contenu et présentation dans 2 fichiers différents. Il est alors possible d'associer un même fichier de présentation à plusieurs fichiers de données et inversement, ce qui correspond parfaitement à l'idée d'adaptabilité des SI défendue par KIWIS.

Au niveau implémentation, pour réaliser ce prototype, les technologies XML et Java ont été utilisées. XML, établi comme le standard d'échanges de données, favorise la mise en œuvre de l'adaptabilité en permettant de séparer contenu et présentation. Java fournit un ensemble d'outils permettant la mise en œuvre de solutions XML et offre des mécanismes de gestion de session intéressants. Ces 2 technologies sont présentées dans le chapitre suivant.

## 2.4 Implémentation

### 2.4.1 Choix techniques

En partant des besoins identifiés pour un SI qui se présente comme un ensemble de moyens et d'organisations mis en œuvre pour collecter, structurer et stocker, traiter et diffuser de l'information les choix techniques suivants ont été faits :

- Collecte d'informations :

- Formulaires HTML

- Conversion AROM

- Structuration et stockage des informations :

- Structuration et stockage du modèle des données : XMLSchéma, une définition de la structure des objets d'une classe par fichier .xsd

- Stockage des instances : XML, les objets d'une classe par fichier .xml

- Stockage des utilisateurs/groupes : XML, un fichier par SI

- Traitement de l'information par le biais de requêtes, selon la syntaxe XQuery, encapsulées dans des servlets [SERVL].
- Diffusion de l'information : bibliothèques de feuilles de style (XSL) correspondant aux différents profils possibles, sélectionnées par traitement dans les servlets.

### 2.4.2 Plate-forme KIWIS retenue

L'architecture logicielle de KIWIS est décrite dans la figure 2.5

- Serveur WEB : Apache 1.3 (version compatible Windows NT et Windows 95)

- Apache est un serveur HTTP (compatible version 1.1 du protocole) gratuit, fonctionnant sur de nombreux systèmes (Windows NT/9x, Netware 5.x, OS/2, et la plupart des versions d'Unix), modulaire, efficace.

- Il a été choisi notamment parce qu'il est le plus utilisé aujourd'hui et, qu'à ce titre, il fait l'objet de nombreux développements.

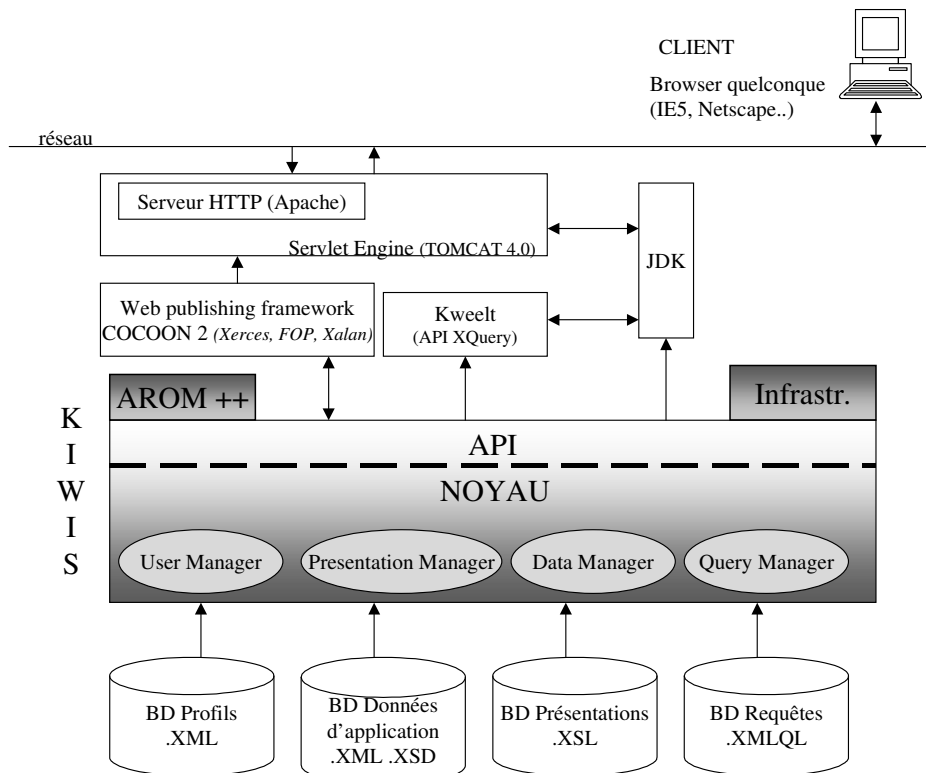


Figure 2.5 : Architecture logicielle du prototype

□ Moteur de servlets : Tomcat 4.0

Le rôle d'un moteur de servlets est de gérer l'activité des servlets tout au long de leur vie. Le moteur prend en charge les détails des connexions Internet, récupérant les requêtes http, et les transformant en un objet facile à manipuler par les servlets. Cette technologie est choisie pour sa robustesse. Il existe plusieurs moteurs de servlets, mais Tomcat [TOMC] a retenu l'attention car il s'agit de l'implémentation de référence des spécifications Java Servlet 2.3 API [TOMCAT], faisant partie du projet open-source Jarkarta de l'équipe de développement Apache

□ Publication Web : Cocoon 2

Cocoon [COCO] est une plate-forme de publication de contenu XML sur le web, écrite en Java (il s'agit en fait d'une servlet). Pour fonctionner, Cocoon doit être installé au sein d'un moteur de servlets comme par exemple Tomcat. Lors de l'installation, le parseur XML Xerces, le transformateur XSLT Xalan et le formateur XSL :FO FOP sont installés dans l'environnement Tomcat (moteur de servlets choisi).

□ Interrogation de fichiers XML : Kweelt

Kweelt [KWEELT] est une implémentation Java de la proposition de langage de requêtes pour XML, Quilt. La version 2 de cet outil intègre la spécification W3C XQuery. Cette implémentation satisfait complètement les pré-requis de la norme W3C pour les langages de requêtes. Il s'agit d'un outil Open-source, écrit en Java 1.2.

#### ¶ AROM

AROM (Allier Relations et Objets pour Modéliser) est un système de représentation de connaissances [AROM2000]. Réalisé en Java, AROM constitue une plate-forme de développement de bases de connaissances offrant une représentation graphique à la UML, une interface de programmation Java et des outils de consultation et d'édition de ces bases sur le Web. Ce qui est visé en priorité, dans le cadre de l'utilisation d'AROM, est la possibilité d'offrir au concepteur un outil graphique de définition du contenu informationnel de son SI.

#### ¶ JDK 1.3

Il s'agit de l'environnement de développement Java fourni par Sun Microsystems, qui inclut un compilateur, une machine virtuelle, la bibliothèque de classes Java (API) et divers utilitaires.

### 2.4.3 La sécurité

La sécurité informatique peut être définie comme étant la situation dans laquelle un système informatique, connecté ou non à un réseau externe de télécommunications, est protégé des dangers internes ou externes. Dans le cadre de systèmes d'information basés sur le Web, les différentes facettes de la sécurité sont [DONS1998] :

- *l'authentification* : détermination de l'identité de l'utilisateur,
- la *confidentialité* : contrôle de la divulgation des informations,
- *l'intégrité* : assurance que l'information stockée ou transmise est inaltérée,
- la *non-répudiation* : protection contre la négation d'une action accomplie.

Dans cette première version du prototype KIWIS, le sujet n'a pas été traité complètement. Certains aspects ont cependant été mis en place (voir tableau 2.6).

sécurité dans KIWIS	
Authentification	login et mot de passe
Intégrité	
Confidentialité	profils
Non-répudiation	log ?

Tableau 2.6 : La sécurité dans KIWIS

### 2.4.4 Conclusion

Le tableau 2.7 récapitule pour les différentes fonctions d'un système d'information, les technologies et outils utilisés dans la première version du prototype KIWIS.

	<b>technologies et outils utilisés</b>
<b>Collecte</b>	formulaires HTML,XSLT AROM
<b>Structuration</b>	XML Schema (éventuellement après conversion AROM-KIWIS)
<b>Stockage</b>	XML (pour les données) XML Schema (pour la structure)
<b>Traitement</b>	servlets java (traitement feuilles de style XSL, requêtes Xquery) et un peu de code javascript
<b>Diffusion</b>	COCOON (XML, XSL --> HTML)

Figure 2.7 : Technologies et outils utilisés dans le prototype

### 3. L'étude des technologies mises en œuvre

Les technologies XML (Extensible Markup Language) et Java ont de nombreuses similitudes [BERN2000] :

- indépendance par rapport à la plate-forme et à un constructeur,
- conçues pour le Web (ou tout du moins, dont les caractéristiques ont été facilement adaptées au Web),
- support pour UNICODE : codes caractères conçus dès le départ avec une idée d'internationalisation (codes pouvant couvrir tous les caractères de n'importe quel langage humain).

De plus, la plate-forme Java propose un ensemble impressionnant d'outils permettant la mise en œuvre de la technologie XML pour l'ensemble des besoins des applications Internet :

- traduction sur mesure de XML vers HTML (pour les navigateurs commerciaux) grâce aux API Java,
- visualisation côté clients selon les besoins des utilisateurs (par application ou applet),
- échange de données (passerelle avec les bases de données et différents types d'application).

La première partie de ce chapitre retrace l'historique d'XML, présente rapidement la norme et les langages associés : XML schéma, XSL et XQuery. La seconde partie argumente le choix des servlets Java.

#### 3.1 XML

##### 3.1.2 La norme XML

Extensible Markup Language (XML) [XML1998] est un nouveau "langage" de description et d'échanges de documents structurés, centré sur les données. XML ne doit pas être vu comme un langage de programmation mais comme un métalangage. Autrement dit, XML est un langage capable d'en décrire d'autres. En fait, il est au centre d'un ensemble de technologies (à l'état de recommandations W3C déjà parues ou à paraître) qui se complètent mutuellement.

L'un des principaux avantages d'utiliser XML comme source de données est que la présentation de celles-ci (comme dans une page Web par exemple) est séparée de leur structure. Les données XML définissent la structure et le contexte, alors qu'une feuille de

style est appliquée pour définir la présentation. Ces données XML peuvent être présentées de nombreuses façons, en leur appliquant simplement différentes feuilles de styles. Par exemple, différentes interfaces peuvent être présentées aux utilisateurs suivant leur profil, leur type de navigateur web, ou encore d'autres critères en définissant une feuille de style différente pour chacune des configurations.

Les apports décisifs de XML qui peuvent motiver le choix de cette norme, sont [MICH2000]:

- extensibilité et structure
- modularité et réutilisation des structures types
- contrôle de validité
- accès à des sources de données hétérogènes

### **3.1.3 XML Schéma**

Un document est « bien-formé » s'il respecte la syntaxe XML. Définir la structure d'un document XML est une étape indispensable lorsque l'application qui utilise le document doit vérifier la conformité des données XML à un modèle prédéfini. On parle alors de document « valide ». L'outil traditionnel pour valider un document XML s'appelle la DTD (Document Type Définition). Toutefois, les auteurs d'applications se sont rendu compte dans les années qui ont suivi l'adoption de la recommandation XML (dont est issu le mécanisme de DTD), que ce formalisme pourrait aussi servir à l'échange de données structurées entre les applications les plus diverses (sans orientation "document") comme dans les applications de traitement de transactions commerciales. C'est dans cette optique, que les schémas XML ont vu le jour. Les schémas offrent l'avantage par rapport aux DTD de :

- pouvoir contraindre les données et les structures (fort typage),
- être exprimés dans une syntaxe XML bien formé,

XMLSchema [XSCH] est une norme au statut de recommandation W3C depuis Mai 2001, qui utilise la syntaxe XML pour toutes les descriptions de données.

Pour présenter les données stockées dans des documents XML, il faut appliquer à celui-ci une feuille de style. Les eXtensible Stylesheet Language (XSL) permettent de spécifier des réalisations physiques d'un document pour divers supports, et aussi de modifier la structure ou le contenu du document.

### 3.1.4 La spécification XSL

Extensible Stylesheet Language (XSL) [XSLT], le langage de feuilles de style associé à XML, est l'un des outils les plus puissants de la technologie XML. En effet, l'un des apports majeur de XML est de permettre de séparer l'information de sa présentation. Cela signifie principalement qu'une même information, à l'origine conçue ou extraite pour être générique et indépendante d'un quelconque contexte d'utilisation, pourra être utilisée pour servir différents utilisateurs sur différents supports de présentation. Plus précisément :

- servir différents utilisateurs signifie être en mesure de personnaliser l'information et de l'adapter à une cible identifiée pour présenter un contenu pertinent et cohérent.
- publier vers différents supports de présentation signifie être en mesure de délivrer une information dans divers formats supportés par les navigateurs Internet, les téléphones WAP (Wireless Access Protocole), les assistants de gestion personnels, les imprimantes, etc.

XSL est l'outil qui tient les promesses d'XML en permettant tout d'abord de transformer un document XML pour l'adapter à son contexte d'utilisation et, ensuite, de le présenter dans le format attendu par l'utilisateur ou l'application qui souhaite accéder à l'information.

Processus de transformation d'XSL :

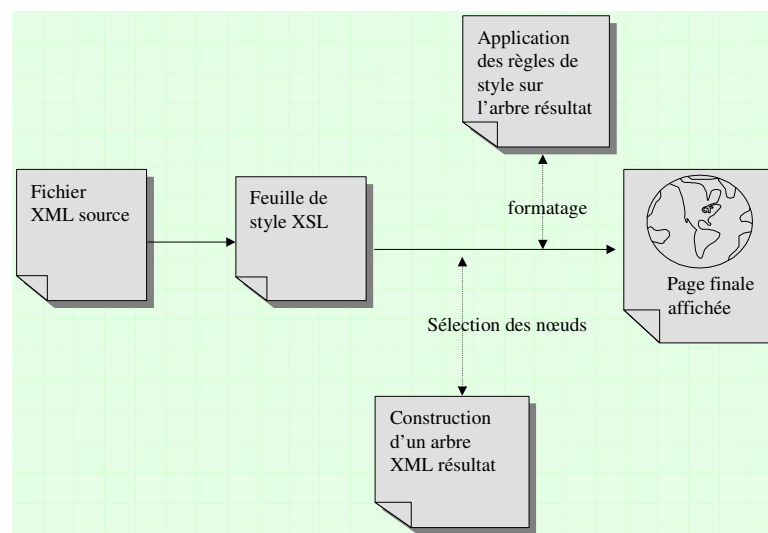


Figure 3.1 : principes de fonctionnement d'XSLT

XSL Transformations (XSLT) a été initialement conçu comme un langage permettant de spécifier des transformations d'arbres, à l'intérieur d'un mécanisme général de feuille de style (XSL). La figure 3.1 illustre les principes de fonctionnement de XSLT : un document XML source est associé à une feuille de style, ce qui entraîne tout d'abord une sélection ( ) dans l'arbre source des nœuds et leur contenu qui composeront l'arbre résultat, puis une opération de formatage ( ) par application des règles de style définies dans la feuille XSL, pour aboutir généralement à un arbre HTML final ( ) formaté à partir de données XML.

Le langage d'expressions utilisé dans XSLT est le même que celui défini pour Xpath [XPATH], un autre langage de la famille XML qui permet l'adressage dans des arbres XML.

### 3.1.7 Conclusion sur XML

Les technologies du KIWIS dans ce chapitre sont celles mises en oeuvre pour la réalisation de notre prototype. Cela nous a permis d'évaluer leurs possibilités dans le cadre de systèmes d'informations multimédias adaptables.

Le tableau 3.2 récapitule les points remarquables de chacune des technologies abordées.

	statut	points remarquables
<b>XML</b>	"eXtensible Markup Language" Recomm. W3C 1999	- extensibilité - contrôle de validité - séparation structure et présentation des données - accès à des sources de données hétérogènes
<b>XML Schema</b>	"eXtensible Markup Language Schema" Recomm. W3C Mai 2001	- prise en compte des espaces de noms - fort typage des données - syntaxe XML
<b>XSL</b>	"eXtensible Stylesheet Language" Recomm. W3C nov. 1999	- réorganisation de documents XML (XSLT) - formatage selon support de sortie

Tableau 3.2 : Points forts des technologies XML présentées dans ce chapitre

La section suivante propose une présentation des technologies serveur utilisées pour le prototype : les servlets Java, le suivi de session, les interfaces de programmation DOM et SAX.

## 3.2 Les servlets Java

Les servlets (on dit généralement *une servlet*) sont au serveur Web ce que les applets sont au navigateur pour le client. Les servlets [Hunter,1999] sont donc des applications Java fonctionnant du côté serveur.

- Utilité et technologies existantes

La figure 3.3 expose le fonctionnement d'une application Web.

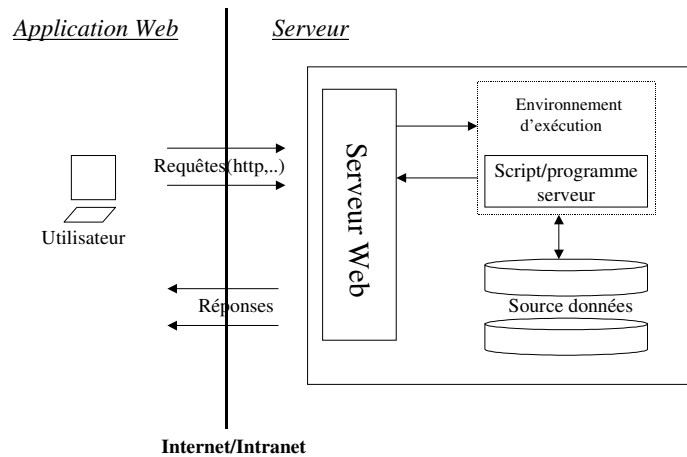


Figure 3.3 : Fonctionnement d'une application Web

L'application Web recueille les données de l'utilisateur et envoie une requête au serveur Web (généralement par l'intermédiaire d'un formulaire HTML contenant du code Javascript, des applets, etc.)

Le serveur Web identifie l'environnement d'exécution, le charge, et exécute le script ou programme serveur (connexion à une base de données, personnalisation de pages, etc.) correspondant.

Le script ou programme serveur précise le contenu et intègre la réponse dans le flot de sortie à destination du navigateur. Celui-ci affiche ensuite les données pour l'utilisateur.

L'unité principale de traitement est donc le script ou programme serveur. Il existe plusieurs technologies dans ce domaine : les scripts CGI (Common Gateway Interface) écrits en Perl, PHP (PHP : Hypertext Pre-processor), les ASP (Active Server Pages de Microsoft) et les servlets/JSP (Java ServerPages) de Java.

Les servlets présentent plusieurs avantages sur les autres technologies [BERN2000]:

- une servlet n'est pas exécutée dans un processus séparé,
- une servlet reste en mémoire entre les différents processus,
- il n'y a qu'une seule instance, qui répond à toutes les requêtes,
- une servlet peut être exécutée par un moteur de servlets, ce qui permet une utilisation sécurisée.

De plus, cette technologie hérite de tous les points forts du langage Java (portabilité, puissance, ..etc.).

#### □ Principes de fonctionnement

Le cycle de vie d'une servlet Web est le suivant :

- la servlet est créée puis initialisée (méthode *init()* de *HttpSession*),
- le service du client est implémenté (méthode *service()*),
- la servlet est détruite (méthode *destroy()*).

Les servlets fournissent également des méthodes (*getParameter()* par exemple) pour récupérer les valeurs des différents objets d'un formulaire HTML (case cochée, valeur sélectionnée dans une liste de sélection, etc.). Il suffit pour cela, d'indiquer dans l'attribut 'action' de la balise HTML <form> le nom de la servlet à invoker.

Enfin, les servlets permettent aussi de gérer des sessions (*HttpSession*) [HUNT1999]. C'est le mécanisme choisi dans KIWI pour identifier les utilisateurs connectés, récupérer leur profil et gérer leur session.

### 3.2.4 Le suivi de session

Le protocole HTTP est un protocole non connecté (on parle aussi de *protocole sans états*, en anglais *stateless protocol*), cela signifie que chaque requête est traitée indépendamment des autres et qu'aucun historique des différentes requêtes n'est conservé. Ainsi le serveur web ne peut pas se "souvenir" de la requête précédente, ce qui est dommageable dans des utilisations telles que le e-commerce, pour lequel le serveur doit mémoriser les achats de l'utilisateur sur les différentes pages.

Il s'agit donc de maintenir la cohésion entre l'utilisateur et la requête, c'est-à-dire :

- reconnaître les requêtes provenant du même utilisateur
- associer un profil à l'utilisateur
- connaître les paramètres de l'application (nombre de produits vendus, ...)

On appelle ce mécanisme de gestion des états le **suivi de session** (en anglais *session tracking*).

Avant l'apparition des servlets, il y avait trois méthodes qui permettaient de réaliser des suivis des sessions.

- utilisation de *cookies* (des chaînes de caractères stockées côté client et renvoyées au serveur à chaque demande effectuée par le client)
- réécriture d'URL (le rajout après l'URL de la requête une succession de paramètres/valeurs qui permettent d'identifier précisément la requête)
- utilisation de champs cachés dans un formulaire HTML (des éléments <input type="hidden" name="paramI" value="valueI"> où les paires *paramI*, *valueI* permettent au moment du traitement du formulaire, l'identification de la requête comme appartenant à une session)

Les servlets supportent toutes ces trois méthodes, et en plus, elles mettent à la disposition des programmeurs un nouvel outil, plus souple et plus sûr, l'objet *HttpSession*.

L'objet *HttpSession* permet de mémoriser les données de l'utilisateur grâce à une structure similaire à une table de hachage, permettant de relier chaque identifiant de session (*JSESSIONID*) à l'ensemble des informations relatives à l'utilisateur. Cet ensemble d'informations concernant l'utilisateur est stocké côté serveur, et seulement l'identifiant de la session est renvoyé au client, en utilisant selon les contraintes, soit les cookies, soit la réécriture d'URL. Cet envoi d'informations est transparent pour le programmeur, qui ainsi, n'a pas à se préoccuper du fait que le navigateur client accepte ou non les cookies.

Parmi les avantages du stockage des informations côté serveur s'inscrivent les suivants :

- la possibilité de stocker des objets complexes, chose impossible à réaliser par le biais des cookies,
- un trafic moins important entre le client et le serveur, puisque à chaque demande on n'envoie que l'identifiant de la session et non pas toutes les informations concernant celle-ci,
- une sécurité plus élevée des informations qui étant stockées côté serveur ne sont pas susceptibles d'être interceptées.

Un scénario typique de la réalisation du suivi d'une session est le suivant :

a) **obtenir ou créer l'objet session.** La création d'un nouveau objet session doit se faire impérativement avant que la réponse ne soit envoyée au client. L'objet session est obtenu suite à l'appel `getSession(boolean create)`. Si l'argument *create* vaut *true* alors un nouvel objet session est généré, et si l'argument vaut *false*, alors selon les cas on obtient un objet session associé à la session, si une existe ou *null* sinon.

b) **la récupération et la modification des données associées à la session.** Une fois qu'on a obtenu l'objet session, celui-ci peut être utilisé comme un bulletin d'informations en ligne. Les données d'une session sont accessibles à tous les servlets de l'application qui peuvent les modifier si nécessaire. Les deux méthodes principales de l'objet session sont les suivantes : *Object* `getAttribute(String name)` qui permet de récupérer les informations stockées dans la session, et *void* `setAttribute(String name, Object value)` qui permet d'ajouter ou de modifier les informations de la session.

c) **la désactivation/la destruction d'un objet session.** Les objets session sont automatiquement détruits par le système dans la mesure où entre deux requêtes successives les concernant un intervalle de temps est dépassé. Cet intervalle sur la plupart des conteneurs de servlets est de 30 minutes, mais à l'aide de la méthode `setMaxInactiveInterval()` de l'objet session, on peut modifier cette valeur. Dans certains cas, il est souhaitable que la désactivation

d'une session soit réalisée dès qu'une série d'opérations a abouti. Ce cas a été aussi prévu et il suffit d'appeler la méthode `invalidate()` d'objet session qui détruit la session invalide.

Voici un petit exemple de gestion d'un objet session. L'exemple illustre une possibilité de gérer l'authentification des clients d'une application web.

```
public void doPost(HttpServletRequest req, HttpServletResponse res) {
    HttpSession ses=req.getSession(false); //essayer d'obtenir de la session en cours
    if (ses!=null) {
        ses.invalidate(); //détruire la session existante
    }
    String login=req.getParameter("login");
    if (authenticationReussie(login, req.getParameter("passwd"))) {
        ses=req.getSession(true); //création d'une nouvelle session propre à ce nouveau client
        ses.setAttribute("login", login); //sauver dans l'objet session l'information concernant le
client
        ses.setMaxInactiveInterval(360); //si le client ne se manifeste pas à nouveau dans 6
minutes la session sera détruite.
    }
    ...
}

public void doGet(HttpServletRequest req, HttpServletResponse res) { // une requête de type Get
dans cet exemple est considère comme une demande de déconnexion
    HttpSession ses=req.getSession(false);
    if (ses !=null) {
        ses.invalidate();
        res.getWriter().println("déconnexion effectuée avec succès");
    }
    ...
}
```

### 3.2.4 Les API DOM et SAX

Les premiers paragraphes de ce chapitre ont montré que la technologie XML permet de définir la structure du document uniquement, ce qui permet, d'une part de pouvoir définir séparément la présentation de ce document, d'autre part d'être capable de récupérer les données présentes dans le document pour les utiliser.

Toutefois, la récupération des données encapsulées dans le document nécessite un outil appelé « analyseur » (en anglais *parser*), permettant de parcourir le document et d'en extraire les informations qu'il contient.

Si on distingue 2 types de parseurs (*validants* permettant de vérifier qu'un document est conforme à sa DTD, *non validants* se contentant de vérifier que le document XML est bien formé), on distingue surtout 2 types d'approches [MICH2000] :

- les API utilisant une approche « hiérarchique » : les analyseurs utilisant cette technique construisent une structure hiérarchique contenant des objets représentant les éléments du document, et dont les méthodes permettent d'accéder aux propriétés. La principale API utilisant cette approche est DOM (Document Object Model).
- les API basés sur un mode « évènementiel » permettant de réagir à des évènements (comme le début d'un élément, la fin d'un élément) et de renvoyer le résultat à

l'application utilisant cette API. SAX (Simple API for XML) est la principale interface utilisant l'aspect événementiel.

#### ¶ DOM

Le "modèle objet de document" [DOM] fournit un moyen de gérer des documents XML dans un programme. Le DOM offre la possibilité de créer des documents et des fragments de documents, de naviguer dans le document, de copier et de supprimer des fragments de documents, d'ajouter ou de modifier des attributs.

Le DOM constitue l'un des moyens les plus simples et les plus courants de respecter les recommandations du W3C en matière de XML. Il crée tous les objets en mémoire associant chacun d'eux à un nœud dans le document XML, ce qui explique pourquoi les implémentations du DOM peuvent être volumineuses et le traitement des documents XML monopoliser la mémoire vive. Il existe une autre façon d'accéder aux informations des documents : l'API simplifiée de XML (SAX).

#### ¶ SAX

Cet outil s'avère efficace dans les opérations où le DOM présente des faiblesses. Les deux approches sont complémentaires et indispensables au développeur XML.

Simple API for XML [SAX] a été développé pour optimiser l'analyse de documents XML volumineux. Le traitement de document est orienté évènement. Au lieu d'analyser le document dans le DOM et d'utiliser celui-ci pour la navigation, le parseur génère des évènements quand il rencontre quelque chose.

Cette API est simple (peu d'interfaces), peu exigeante en ressources et rapide, car le document n'est pas chargé en totalité pour que son analyse commence. Cependant, SAX possède aussi quelques défauts : les données sont extraites séquentiellement, SAX ne fournit pas d'informations lexicales (n'extrait pas les commentaires et n'indique pas l'ordre des attributs), SAX est en lecture seule (impossible de modifier les éléments du document).

### **3.2.5 Conclusion sur Java**

Le tableau de la figure 3.4 liste les points remarquables des technologies Java présentées dans ce chapitre et qui paraissaient intéressantes pour le développement d'un prototype autour d'XML.

	fonctionnalités	points remarquables
<b>les servlets</b>	- génération de pages dynamiques - suivi de session	- méthodes Java - gestion des formulaires HTML - permet d'identifier les utilisateurs et de gérer leurs préférences
<b>DOM et SAX</b>	- parcours et mise à jour de documents XML	- utilisables depuis les servlets et les pages HTML,XSL - construction arborescente et syntaxe relativement simple qui facilite la mise à jour (extraction, navigation, ajout et suppression) par traitement de documents conformes à la syntaxe XML

Figure 3.4 : Points remarquables des technologies Java utilisées dans KIWIS

### 3.3 Conclusion sur Java et XML

Les technologies XML et Java partagent la même philosophie : une utilisation dans le cadre de systèmes distribués et des simplifications de langages complexes (SGML pour XML et C++ pour Java). Ces deux points rendent naturelle l'utilisation commune de ces deux langages. Le schéma de la figure 3.5 a pour objectif de montrer l'interconnexion des deux technologies.

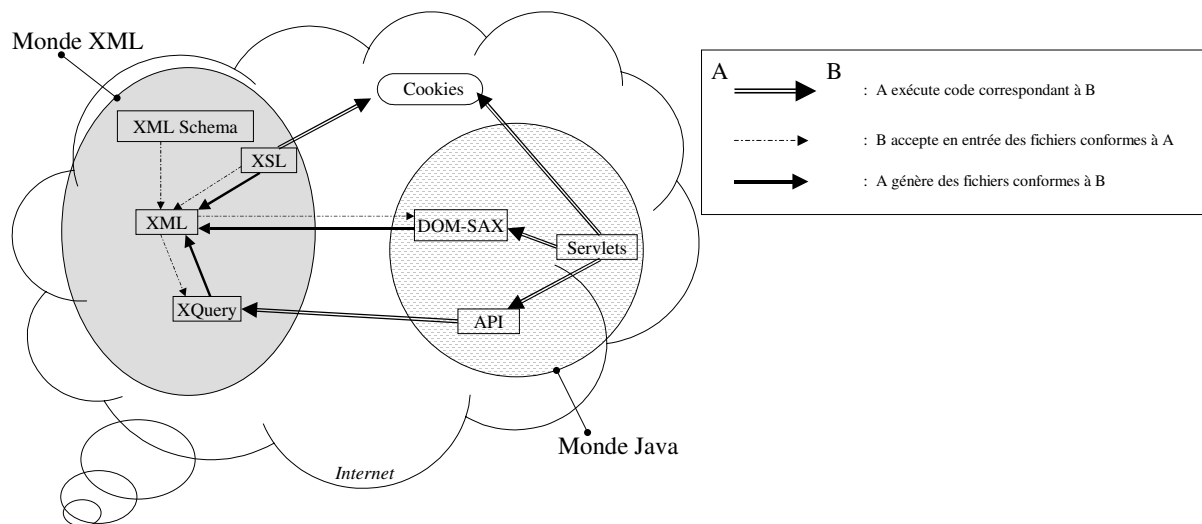


Figure 3.5 : Interconnexion XML et Java

Les servlets ont des méthodes leur permettant d'assurer le suivi des sessions, d'utiliser les interfaces DOM et SAX, et d'exécuter des API concernant les langages de requêtes (comme Kweelt, une API du langage XQuery [XQUERY], langage standardisé

d'interrogation des bases de données XML). La passerelle vers le monde XML est ainsi établie.

L'interconnexion au sein de la famille de langages XML n'est plus à démontrer : XQuery est un langage de requêtes pour les documents XML, qui génère un fichier XML résultat ; la validation de documents XML se fait actuellement grâce à XML Schema ; la mise en forme de documents XML est réalisée par les feuilles de style XSL, dans lesquelles peut être encapsulé du code JavaScript pour la gestion des cookies. Enfin, la mise à jour de fichiers XML (ajout, suppression de nœuds), est aujourd'hui très souvent géré par DOM et SAX. Et la passerelle vers le monde Java, cette fois, est établie.

## **3.4 Les technologies support : le moteur des servlets-Tomcat et l'outil de publication COCOON**

### **3.4.1 Le moteur des servlets Tomcat**

Le moteur de servlets Tomcat est une composante du serveur Web Apache. En général un moteur des servlets peut être une partie intégrante de celui-ci, mais on peut l'installer comme une application à part entière, comme un module de serveur Web. L'avantage d'utiliser le serveur Web Apache-Tomcat, réside dans le fait que la communication entre le serveur et le moteur se réalise plus facilement et qu'elle est moins coûteuse du point de vue des ressources utilisées.

Le moteur des servlets gère les servlets des applications installées sur le serveur Web où il se trouve. Son but est d'aider les développeurs, de leur permettre de se focaliser sur les aspects importants de l'application et d'ignorer les aspects techniques liés à :

- La réalisation de la connexion Internet,
- La récupération des données incluses dans les requêtes http,
- La façon dont les réponses sont envoyées et leur éventuel formatage.

Le moteur transforme le protocole utilisé pour l'envoi des requêtes en un objet accessible aux servlets. En même temps, il met à la disposition des servlets, un objet qui permet d'envoyer la réponse correspondante à la requête.

Le conteneur ou le moteur des servlets gère aussi le cycle de vie des servlets :

- la création d'une instance de la servlet
- l'appel de la méthode `init()` qui leur permet de s'installer et se configurer correctement

- appeler la méthode `service()` de celle-ci, si une requête la concernant arrive
- l'appel de la méthode `destroy()` avant la destruction de la servlet pour libérer les ressources utilisées
- la destruction de l'instance

Il est possible que ce cycle soit exécuté à chaque requête, mais cela ralentirait le processus de renvoi de la réponse. Aussi, en pratique le moteur crée une première instance, soit après le démarrage du moteur, soit à la première requête reçue, selon les configurations faites par les concepteurs. La destruction de l'instance peut arriver à n'importe quel moment, mais en général la destruction aura lieu seulement si un délai est dépassé sans qu'aucune requête ne soit arrivée pour la servlet en cause.

L'implantation de serveur Apache-Tomcat et des applications Web de celui ci suit la schéma suivant :

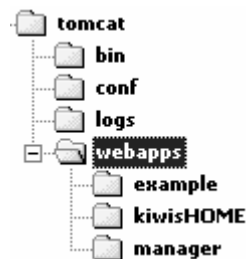


Figure 3.6 Implantation de serveur Apache-Tomcat

Le répertoire `/bin` contient les classes de Apache-Tomcat, le répertoire `/conf` contient les fichiers de configuration de Tomcat (le principal fichier est `server.xml` qui contient la configuration de moteur). Le répertoire `/webapps` contient les applications web. Chaque sous-répertoire contient une application web.

La structure d'un répertoire associé à une application Web installée au sein de Tomcat est créée suivant ce schéma :



Figure 3.7 structure minimale imposée d'une application Web

Les répertoires `classes` et `lib` contiennent les classes java qui sont nécessaires au bon fonctionnement de l'application Web. Ces répertoires sont créés pour simplifier le travail des développeurs, qui ne sont pas censés changer la variable système `CLASSPATH` chaque fois qu'ils installent une nouvelle application Web. Ce mécanisme, qui charge pour chaque application ses propres bibliothèques (les classes dans `classes` et `lib`), accroît l'indépendance des

applications Web installées au sein de Tomcat, ce qui permet d'avoir sur un même serveur Apache-Tomcat plusieurs applications qui fonctionnent sans aucune interférence.

Le répertoire *WEB-INF* contient aussi un fichier *web.xml* qui permet de configurer d'avantage l'application est aussi de designer les correspondances entre les requêtes concernant cette application et ses servlets.

Par défaut, les requêtes correspondent à l'application installée dans le sous-répertoire (de webapps) dont le nom correspond à la première partie d'URL. Ainsi une requête <http://server:port/example/index.html> sera renvoyée à l'application installée dans le répertoire *example*. Mais par le biais d'un fichier de configuration *server.xml*, qui permet de décrire les configurations de l'application et de spécifier certains attributs de celle-ci, on peut faire correspondre les requêtes à telle ou telle application. Voici un extrait qui permet, au moment de la réception d'une requête par le moteur des servlets, de trouver l'application Web à laquelle la requête est adressée.

```
...
<Context path="/kiwis" docBase="kiwisHome" .../>
<Context path="/manager" docBase="manager"/>
<Context path="/examples" docBase="examples">
...
```

Dans cet extrait toutes les requêtes qui commencent par « *kiwis* » (<http://server:port/kiwis/...>) seront traitées par l'application installée dans le répertoire *kiwisHome*, celles commençant par *manager* seront traitées par l'application installée dans le répertoire *manager*, etc.

Outre ces aspects d'organisation physique d'une application, le moteur associe à chaque application Web un objet *Context* qui spécifie les différents aspects de l'application. Cet objet est accessible aux servlets au moment de l'initialisation de celles ci, mais aussi au moment du traitement d'une requête. Cela permet de réaliser la communication entre les différentes entités de l'application Web (en principe les pages JSP et les servlets), qui peuvent utiliser l'objet *Context* comme un porteur des messages en lui associant des attributs qui peuvent être lus par toutes les entités de l'application.

Un autre aspect assez important pour nous est le fait que les objets sessions *HttpSession*, dont nous avons parlé dans le paragraphe précédent sont propres à un client et à une application/contexte précis. Ainsi chaque transaction (durée de vie d'une session) concernant le client et l'application Web est isolée, ce qui élimine les risques de perte ou vol d'informations soit par un autre client, soit par une autre application. Cependant, cet aspect révèle quelques limitations, par exemple, dans le cas où la transaction que le client est en train d'effectuer porte sur plusieurs applications.

Tomcat constitue également le support d'un puissant environnement de publication de documents XML, appelé Cocoon, qui est présenté dans le paragraphe qui suit.

### 3.4.2 L'outil de publication Cocoon

Apache Cocoon est un outil de publication des documents XML. Il est centré sur l'utilisation de XML et de XSLT en tant que technologies côté serveur. Conçu pour atteindre un maximum de performance, Cocoon offre un environnement flexible basé sur la séparation des trois principaux aspects d'une application : le contenu, la logique, et le style.

Cocoon interagit avec la plupart des types de base des données incluant : les système de fichiers, le base de données XML, et des basées de données réparties sur le Web. Il adapte le contenu de la réponse en tenant compte des capacités de réception du client : HTML, WML, PDF, SVG, ou RTF, pour en énumérer quelques uns. En général, Cocoon est utilisé comme un servlet, mais on peut aussi interagir à l'aide d'une puissante interface de commande en ligne. Il a été conçu comme un environnement abstrait, qui donne la possibilité d'étendre ses fonctionnalités selon les besoins des développeurs de façon aisée et modulaire.

Le concept de base en Cocoon est la carte du site (*sitemap*). Le *sitemap* permet la construction des sites web à partir des documents XML et des composantes qui contiennent la logique métier.

Le *sitemap* a été conçu en partant des principes suivants :

- la clarté est d'une importance majeure,
- le schéma doit être suffisamment expressif pour permettre l'apprentissage en lisant les exemples
- il ne faut pas qu'il impose des limitations, ni sur l'extension éventuelle ni sur les possibilités d'administration du site
- il doit contenir toutes les informations nécessaires à Cocoon pour traiter toutes les requêtes qu'il reçoit.
- le mécanisme qui fait la correspondance entre les URI et les ressources doit être assez puissant afin d'assurer toutes les correspondances nécessaires,
- les fonctionnalités de base de service Web (la re-direction, la génération des pages d'erreurs, l'autorisation d'accéder à une ressource) doivent être supportées,
- les noms des ressources peuvent être comparés avec toutes les variables d'état, et pas seulement avec l'URI (des paramètres http, des variables d'environnement, des paramètres du serveur, le temps, etc.).
- Le *sitemap* doit être suffisamment flexible pour permettre la construction entière d'un site

La structure générale d'un *sitemap* est la suivante :

```
<?xml version="1.0"?>
<map:sitemap
  xmlns:map="http://apache.org/cocoon/sitemap/1.0">
  <map:components/>
  <map:views/>
  <map:resources/>
  <map:action-sets/>
  <map:pipelines/>
</map:sitemap>
```

L'élément `<map:components>` décrit les composantes du *sitemap* : les *générateurs*, les *transformers*, les *sérialisateurs*, les *matchers* etc. Les générateurs sont des *parseurs* qui génèrent des événements SAX qui ensuite seront traités par les *transformers*. Puis, le résultat de ces transformations sera sérialisé dans un format compatible avec le dispositif cible (navigateur Web, navigateur WAP, etc.). Les *matchers* rendent possible la correspondance entre des URIs et des ressources du site.

Le mécanisme de publication de Cocoon est construit autour des pipelines: un document XML est envoyé dans un pipeline existant et, subissant une suite des transformations, il devient prêt à être retourné en réponse. Chaque pipeline commence par un générateur, continue avec zéro ou plusieurs *transformers*, puis finit avec une sérialisation. Derrière ce processus que l'on retrouve dans chaque pipeline, on peut aussi définir un mécanisme propre de traitement d'erreurs.

Pour définir un pipeline il suffit de mettre un élément `map:pipeline` à l'intérieur de l'élément `map:pipelines`. Ensuite, on peut inclure dans cet élément un ou plusieurs éléments parmi les suivants :

- `map:match`-choisit le traitement à effectuer selon l'URI de la requête
- `map:mount`-inclut un sous-sitemap
- `map:redirect-to`-Associe à la requête un autre URI, selon laquelle elle sera traitée
- `map:parameter`-définit des paramètres supplémentaires pour les composantes d'un sitemap
- `map:generate`-définit l'étape de création de document XML qui rentre dans le pipeline
- `map:transform`-définit zéro ou plusieurs étapes de transformations
- `map:serialize`-définit la dernière étape, celle de sérialisation
- `map:handle-errors`-traite les erreurs qui apparaissent lors du traitement

Ces éléments décrivent les mécanismes de base pour les différentes étapes du traitement des documents XML. Ces étapes sont :

- la sélection de la suite des transformations à subir basée sur les *Matchers*.
- la génération des documents XML (en partant des objets, des bases des données relationnels, etc.) par le biais des *Generators*
- la transformation (dans un autre document XML, ou différents types d'objets) des documents par le biais des *Transformers*
- la publication des documents XML par le biais des *Serializers*

Une requête reçue par Cocoon est traitée suivant le scénario décrit dans la figure 3.8.

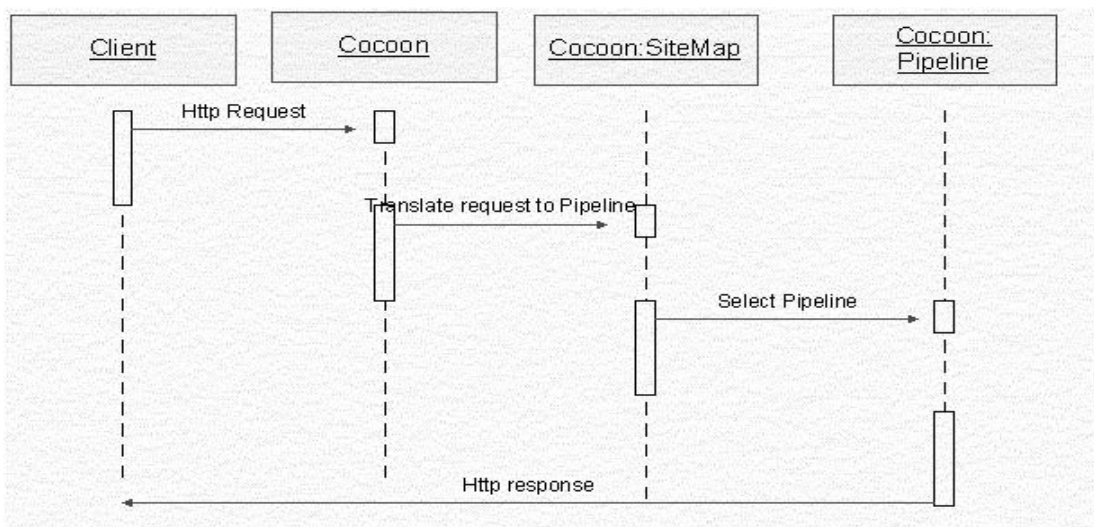


Figure 3.8 Diagramme de séquence du traitement d'une requête

A la réception d'une requête, Cocoon la transforme et l'envoie au *sitemap*, le sitemap en utilisant les *matchers* trouve le bon pipeline et le met en marche.

La figure 3.9 illustre le processus qui se déclenche pour une requête qui porte sur la page *hello.html*.

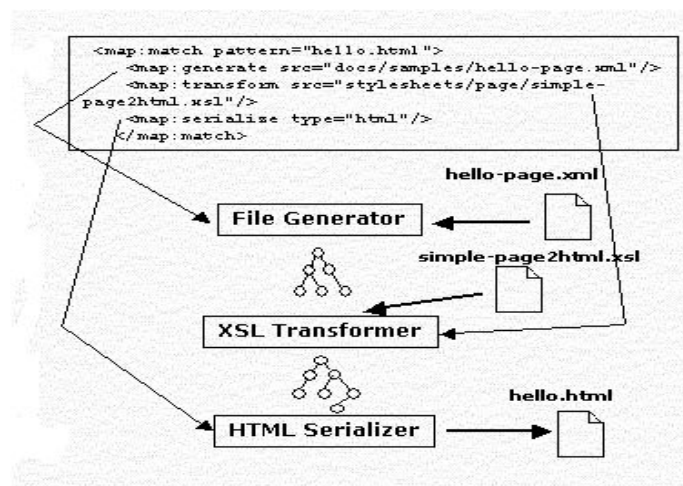


Figure 3.9 le processus de génération de la réponse

Parmi les atouts de Cocoon, on peut citer aussi la possibilité d'intégrer dans un sitemap, d'autres sitemaps, obtenant une structure arborescente du site. Cette structure correspond aux sous-applications installées au sein de Cocoon. Les fichiers des configurations des sitemaps (sitemap.xmap) deviennent plus lisibles et faciles à manipuler dans le cas où des modifications sur ceux-ci doivent se faire. Les sous-sitemaps sont indépendants et il n'y a pas des interactions entre ceux-ci.

```
<map:match pattern="kiwis/*">
  <map:mount uri-prefix="kiwis" check-reload="no"
    src="kiwis/sitemap.xmap"/>
</map:match>
```

L'attribut *src* précise où se trouve le sitemap. L'attribut *check-reload* indique si la modification d'un sitemap implique une régénération automatique de celui-ci. L'*uri-prefix* indique la partie à enlever de l'URI de la requête. Si, par exemple, la page "kiwis/welcome" est demandée, "kiwis/" est enlevé de l'URI et "welcome" est transmis au sous-sitemap qui est chargé depuis "kiwis/sitemap.xmap".

Cocoon se présente comme un outil capable de satisfaire tous les besoins d'un site web, qui en plus de la séparation des trois aspects : contenu, logique métier et style, il permet la construction des sites Web avec un degré élevé de modularité.

## 4. Les résultats

### 4.1 Réflexions sur les différents modèles proposés dans le prototype KIWIS

#### 4.1.1 Le modèle des données

Le modèle de données de l'ancien prototype est un modèle de type Entité-Relation enrichi avec les notions de niveau de détail et de format multiple. Ces deux notions ont été introduites pour créer une stratification de l'espace informationnel qui permet ensuite de définir des espaces informationnels spécifiques à chaque groupe d'utilisateurs.

Au premier abord, l'idée de stratifier le domaine informationnel en utilisant la notion de niveau de détail associée aux entités de l'espace informationnel est intéressante, celle-ci permettant d'assurer à la fois des modalités d'accès progressif à l'information et aussi d'en cacher certains aspects pour des raisons de sécurité. Malheureusement la manière dont elle a été mise en œuvre est contraignante et peu efficace.

En associant directement chaque attribut d'une entité (classe ou association) à un niveau de détail, on crée une stratification du contenu informationnel valable pour tous les groupes d'utilisateurs du système. Dans le cas où on n'a que peu de groupes et un domaine informationnel qui n'est pas très complexe cela peut fonctionner assez bien.

Mais en associant, dès la phase de conception du modèle des données, ces niveaux aux entités (attributs, classes, associations ou autres constructions) on admet que celles-ci ont a priori la même pertinence pour tous les utilisateurs du système. Cet aspect met en évidence des limitations du point de vue de l'accès progressif. Pour illustrer cela, prenons l'exemple d'un SIG (Système d'Information Géographique) sur les crues de rivières. Considérons qu'au premier niveau de la stratification d'une classe, par exemple, on ait les attributs les plus pertinents. Pour des spécialistes, le plus important est de connaître le débit de la crue, alors que pour des utilisateurs courants, l'essentiel est de connaître les dégâts occasionnés par la crue. Donc, dans l'approche proposée, qui associe directement les niveaux de détail aux attributs, il est impossible d'obtenir une stratification pouvant satisfaire à la fois les géographes et le grand public.

La stratification d'un système informationnel doit donc se faire après que le modèle de données ait été conçu, et que les classes d'utilisateurs (regroupés par leurs intérêts) soient bien définies. En fait, il faudrait disposer de plusieurs stratifications du système informationnel, une pour chaque groupe. Cela offrirait ainsi la possibilité de stratifier le système suivant les besoins et les intérêts des utilisateurs.

En principe, dans le modèle des données, les mêmes défauts peuvent être imputés à l'association des formats multiple aux données. Ainsi, les différents formats qu'on peut associer à une information seront les mêmes pour tout le monde, ce qui ne rend pas possible la distinction entre les différents utilisateurs qui n'ont accès qu'en partie à l'information.

Un troisième aspect que nous avons éliminé du modèle antérieur est l'association créée au moment de la définition d'un attribut entre celui-ci et une classe hypermédia (link, image, video, texte, etc). Cette classe hypermédia est ensuite utilisée au moment de la génération des pages web, pour établir la façon dont l'attribut doit être présenté. Cette liaison établie au moment de définition oblige à chaque fois que l'on affiche cet attribut de l'afficher par rapport à sa classe hypermédia. Ainsi, nous proposons d'établir la classe hypermédia de l'attribut dans le modèle hypermédia, dans la phase de personnalisation des pages web où il sera décidé, en fonction des besoins, la classe hypermédia qu'on va lui associer.

Suite à ces constats, nous avons décidé d'enlever les notions de niveau de détail, format multiple et classe hypermédia associée à un attribut de modèle de données, qui redevient un modèle classique de type Entité-Relation, les stratifications de celui-ci reposant sur les modèles des fonctionnalités.

#### **4.1.2 Modèle d'utilisateurs**

Le modèle des utilisateurs n'a pas été changé. Les deux notions : la notion de *group*, et celle de *profil* d'utilisateurs sont suffisantes pour permettre la mise en œuvre des mécanismes d'adaptabilité et de personnalisation du contenu informationnel mis à la disposition des utilisateurs.

Le groupe ou la classe d'utilisateurs regroupe les utilisateurs en fonction des besoins en contenu informationnel. Le profil utilisateurs permet l'adaptabilité de la présentation du contenu conformément aux attentes de l'utilisateur.

Le modèle reste figé, implicite, et le concepteur n'a pas la possibilité de personnaliser le modèle. L'approche de KIWIS étant centrée sur l'utilisateur, tous les attributs des groupes, ainsi que ceux des profils utilisateurs, ont une sémantique bien précise et une logique métier associée. Ainsi, en ajoutant de nouveaux attributs de façon dynamique lors de la conception, le système n'aura pas les moyens de préciser leur sémantique, ni les actions associées.

On peut éventuellement imaginer de mettre à la disposition du concepteur une série d'attributs prédéfinis parmi lesquels il va pouvoir choisir ceux qui lui conviennent, lui laissant ainsi le choix du degré de personnalisation du système.

Nous avons adapté les droits associés aux groupes en les transformant de manière plus précise : le droit de consultation de système, le droit de mise à jour de son profil utilisateur, le droit d'instancier les classes, et les associations du système, et enfin le droit de conception du système

#### **4.1.3 Le modèle des fonctionnalités**

Initialement ce modèle était seulement utilisé pour décrire les différentes pages web en tant que contenu informationnel. On procédait à une sélection des données qu'on désirait présenter dans telle ou telle page. Puis, on accédait progressivement à l'information, via les

niveaux de détails associées aux informations contenues dans la page ainsi que la navigation entre ces niveaux (par la technique de zoom in/out).

Etant donné que dans ce nouveau prototype nous avons supprimé les niveaux de détail des attributs, pour permettre encore un accès progressif aux informations, nous associons cette information au moment de la sélection des attributs qui seront inclus dans cette page web. Nous obtenons ainsi une stratification de la fonctionnalité. Et comme chaque fonctionnalité est propre à tel ou tel groupe, les concepteurs peuvent obtenir des stratifications personnalisées au niveau du groupe.

Revenons au problème de stratification du SIG présente dans le premier paragraphe. Pour obtenir les stratifications souhaitées, il suffit de définir une fonctionnalité pour chacun des groupes. On crée une fonctionnalité *Liste des crues* pour le groupe des *géographes* (voir la figure 4.1a) et une autre fonctionnalité avec le même nom (pourquoi pas) pour le groupe *public* (voir la figure 4.1b).

<b>Liste des Crues</b>	
<i>basée sur la Classe Crue</i>	
Débit	<b>1</b>
Morts	<b>3</b>
Disparus	<b>4</b>
Localisation	<b>1</b>
côte maximale	<b>2</b>

<b>Liste des Crues</b>	
<i>basée sur la Classe Crue</i>	
débit	<b>-</b>
morts	<b>2</b>
disparus	<b>2</b>
localisation	<b>1</b>
côte maximale	<b>3</b>

Figure 4.1a Stratification de la fonctionnalité *Liste des Crues* pour le groupe des *géographes*

Figure 4.1b Stratification de la fonctionnalité *Liste des Crues* pour le groupe *public*

Sur la droite des tableaux, on voit les niveaux de détail associée aux attributs dans le cadre de la fonctionnalité pour un certain groupe. Le concepteur choisit et inclut dans la fonctionnalité seulement les attributs qu'il veut rendre visibles (dans la figure 4.1b, l'attribut débit n'est pas visible pour les membres du group *public*). En partant de la même classe on obtient en effet deux stratifications de celle-ci suivant les besoins des groupes *géographes* et *public*.

En fait, les fonctionnalités sont des *fenêtres* sur les éléments du système informationnel, et c'est par leur biais que l'utilisateur a accès aux informations. En définissant plusieurs niveaux pour une fonctionnalité, et en associant à ces niveaux les différents attributs qu'on veut rendre visibles par cette fonctionnalité, on peut établir le mode d'accès progressif aux informations et traiter certains aspects concernant la sécurité. Il suffit de ne pas inclure dans une fonctionnalité les attributs qu'on désire garder secrets, et l'utilisateur n'aura ainsi jamais la possibilité d'y accéder.

Si dans l'ancien prototype, une fonctionnalité ne portait que sur une classe ou sur une association, dans cette deuxième implémentation on a introduit les fonctionnalités « composées » qui permettent de présenter sur une même page les informations concernant plusieurs classes du domaine informationnel. Cela s'effectue par le biais des fonctionnalités

définies pour ces classes. Ces fonctionnalités composées admettent elles-aussi une stratification des fonctionnalités élémentaires qu'elle contient. Dans ce cas, il y aura plusieurs niveaux pour le mécanisme de zoom in/out de passage entre les différents niveaux de détail, à la fois associés à la fonctionnalité « composée » et aux fonctionnalités élémentaires contenues dans celle-ci.

Restons toujours dans le contexte d'un SIG et considérons qu'il existe encore deux classes Orages et MoyensDéployés. Dans la mesure où le concepteur le souhaite il peut définir une fonctionnalité « composée » pour regrouper sur une même page web la *Liste des Crues*, la *Liste des Orages* et les *Moyens déployés pour secourir les habitants des sites sinistrés*. Selon sa volonté, il peut aussi stratifier cette fonctionnalité composée en présentant par exemple au premier niveau les Orages et les Crues, puis, au deuxième niveau les moyens déployés (voir figure 4.2).



Figure 4.2 Navigation au sein d'une fonctionnalité *composée* (exemple2)

Comme on peut le voir sur les images présentées la stratification de la fonctionnalité simple reste visible, le mécanisme de zoom permet une navigation à la fois entre les niveaux de détails de la fonctionnalité *composée* mais aussi entre les niveaux de détails définies pour les fonctionnalités incluses.

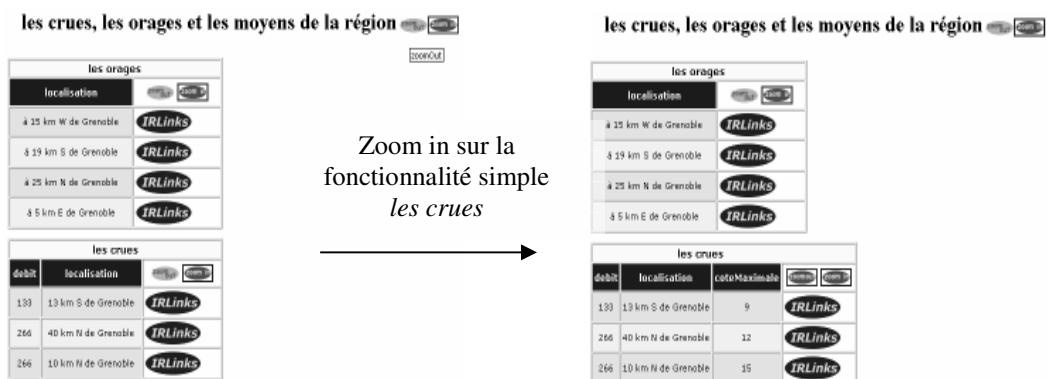


Figure 4.3 Navigation au sein d'une fonctionnalité *composée* (exemple2)

Les fonctionnalités élémentaires associées à un groupe d'utilisateurs délimitent l'espace informationnel associé à ce groupe, constituant une *fenêtre* sur le système propre à la classe d'utilisateurs pour laquelle elles ont été définies.

La stratification sur plusieurs niveaux des informations présentées dans les différentes fonctionnalités associées à un certain groupe, correspond à une stratification personnalisée de l'espace informationnel accessible à celui-ci.

Ainsi, nous avons un modèle qui permet à la fois de créer des stratifications personnalisées de l'espace informationnel mettant les bases de l'accès progressif et aussi de limiter ces espace en fonction des besoins des sécurité du système.

#### **4.1.4 Le modèle hypermédia**

Dans le cadre de ce modèle seuls les aspects liés à la navigation au sein du système ont été modifiés.

Nous considérons que chaque fonctionnalité définie dans le système pour un groupe constitue un nœud de navigation. Ensuite, nous pouvons définir des liens entre le nœud associé à une fonctionnalité et d'autres nœuds de la carte de navigation. Ainsi, la carte de navigation au sein du système est composée de plusieurs cartes de navigation, chacune étant propre à une certaine classe d'utilisateurs. Cette approche permet de construire des cartes de navigation personnalisées au niveau de groupes d'utilisateurs, rendant ainsi le système plus adapté à ses utilisateurs.

Nous avons aussi ajouté des liens au niveau des instances présentées dans le cadre d'une fonctionnalité. Ceux-ci permettent l'accès direct aux fonctionnalités qui ne portent que sur les instances des différentes classes ou associations liées à celle-ci, paramétrant ainsi dans un certain degré les fonctionnalités de consultation.

## **4.2 Nouvelle démarche de conception**

Les étapes de conception du système sont restées quasiment les mêmes, avec néanmoins de petites modifications dictées par les changements apportés aux modèles.

Dans un premier temps, le concepteur doit décrire le modèle de données. Cela s'effectue à l'aide de l'interface mise en œuvre dans cette nouvelle version du prototype. Il n'est pas possible pour l'instant d'effectuer des imports de modèles déjà décrits dans des systèmes de modélisation, comme AROM par exemple. En effet, dans l'ancien prototype il y avait la possibilité d'importer des modèles déjà conçu dans le système KIWIS, mais pour l'instant il n'est pas disponible, car des modifications sont nécessaires pour le réaliser conformément au nouveau modèle de données proposé.

Un fois le modèle des données décrit, la deuxième étape consiste à définir les groupes des utilisateurs, pour pouvoir ensuite instancier de manière personnalisée les deux modèles qui restent.

Le troisième pas dans la conception d'un système d'information revient à définir, pour chaque groupe d'utilisateurs, les fonctionnalités qui les concernent. Cette troisième étape

permet de préciser l'espace informationnel associé à chacun des groupes en fonction des intérêts, et en même temps de stratifier cet espace de façon convenable.

Une fois que les fonctionnalités sont bien définies pour tout le monde, l'étape de définition de la carte de navigation pour chacun des groupes s'enchaîne naturellement.

Pour l'instant au niveau de la présentation nous n'avons pas encore intégré la possibilité de personnaliser les aspects des pages web générées. On dispose malgré tout des deux types de présentation : toutes les instances en tableau, et une instance à la fois.

Une dernière étape consiste à définir les utilisateurs du système, en leur associant un groupe dont ils héritent le droits d'accès et l'espace informationnel accessible. Il faut aussi spécifier différents aspects liés à leurs préférences en terme de présentation des informations (mode tableau/page, le nombre d'items présentées à la fois), ainsi que des préférences liées au navigateur dont il dispose (présence/absence des différents barres, ouverture des liens dans de nouvelles fenêtres, etc.).

Ensuite, il faut instancier les différentes classes et associations. Pour l'instant nous prévoyons le chargement sur le web des fichiers XML conformes aux différents schémas XSD créés suite à la définition du modèle des données. Les informations contenues dans ceux-ci seront ajoutées à celles existants sur le serveur.

Après ces étapes le nouveau système d'information est déployé et accessible à ses utilisateurs.

### **4.3 Les aspects de sécurité dans le nouveau prototype**

La sécurité dans l'ancien prototype se limite à un mécanisme d'authentification des utilisateurs. En fonction de leur *login* on leur fournit soit la page de concepteur (si le *login* était *admin*), soit les pages de consultation du SI choisi. Mais si un des utilisateurs ordinaires de KIWis connaît les URLs des pages concepteur de système, il peut facilement y accéder simplement en saisissant simplement l'URL dans la barre de navigation de son navigateur. Ensuite, il peut modifier le système à son plein gré car même au niveau des servlets qui traitent les formulaires, aucune vérification sur les droits dont l'utilisateur dispose n'est effectuée.

Cet aspect étant essentiel pour un système d'information, nous avons décidé de créer un mécanisme pour empêcher les actions des utilisateurs non autorisés. Dans cette catégorie nous avons inclus :

- les demandes des pages auxquelles l'utilisateur n'a pas le droit d'accéder.
- l'accès aux servlets doit être limité en fonction des droits de l'utilisateur, car même si l'utilisateur habituel n'a pas d'accès aux pages des conceptions, il peut quand même envoyer des requêtes pour les servlets de conception, et avec les bons paramètres parvenir à modifier le système.

Pour satisfaire ces deux points, un suivi de session doit être mis en place. Au moment de la connexion, les droits de l'utilisateur qui vient de s'identifier seront chargés dans la session courante, et lors du traitement des requêtes, celui-ci va s'effectuer en fonction des droits qui sont associés à la session dont la requête fait partie.

L'utilisation des cookies pour réaliser le suivi des session peut satisfaire les besoins de sécurité décrits plus haut. Mais les avantages que l'utilisation des objets HttpSession apporte pour la réalisation d'un suivi de session (la possibilité de stocker un objet complexe dans le cadre d'une session, la limitation du trafic réseau, et une sécurité accrue car l'utilisateur n'a aucun moyen d'accès aux informations), nous ont convaincu de mettre en place le suivi à base de ces objets. L'utilisation des sessions va également permettre plus tard la mise en place des différents processus, dans le but d'accroître l'adaptabilité dynamique des systèmes. Pour cela on peut en effet imaginer que des objets complexes seront utilisés, objets qui en utilisant les cookies seraient assez difficiles à gérer.

#### 4.3.1 Modalités d'implantation

Dans un premier temps, nous avons essayé d'inclure le mécanisme de suivi de session dans l'architecture initiale, où le prototype fonctionnait comme application par défaut de Cocoon. Les objets sessions sont propres à chaque application déployée dans Tomcat, donc pour tous les servlets qui se trouvent à l'intérieur de Cocoon, on n'a le droit qu'à une seule session par client. Ainsi, toutes les sous-applications de Cocoon partagent le même objet session, fait qui peut produire des enchevêtrements assez importants. Une deuxième possibilité étudiée est l'utilisation des sessions internes, propres au Cocoon, mais là aussi le problème était qu'à partir des servlets on ne peut pas accéder à celles-ci.

Il nous restait deux possibilités: soit, nous réalisons l'application dans sa totalité en n'utilisant que des servlets, concevant ainsi notre propre outil de publication, soit nous essayons d'implémenter en Cocoon tous les traitements qui sont effectués au niveau des servlets. Compte tenu du fait que le prototype n'utilise Cocoon que pour la publication des différentes pages web du système, nous avons décidé de concevoir notre propre servlet de publication. Nous pourrions facilement intégrer dans ce servlet le modèle de sécurité souhaité, et il permettra aussi l'application des transformations sur des documents XML, générés dynamiquement suite aux interrogations effectuées sur la base de données XML du système.

Nous avons gardé la notion de *sitemap* présente dans Cocoon, en l'enrichissant avec une notion liée à la sécurité. Pour une entrée, on a normalement : le pattern des requêtes pour lesquelles elle va s'appliquer, le nom de fichier XML et le nom de la feuille de style qui s'y applique :

```
<map:match pattern="*/*_users.html"> <!--creation, suppression des utilisateurs-->
  <map:generate src="{1}/xml/profil_users.xml"/>
  <map:transform src="kiwis/xsl/{2}_users.xsl"/>
  <map:serialize/>
</map:match>
```

Nous avons rajouté encore un élément qui contient les droits dont la session de client doit disposer. Ainsi, nous obtenons l'entrée suivante dans notre propre *sitemap* :

```
<entry match="*/*_users.html"> <!--creation, suppression des utilisateurs-->
  <generate src="{1}/xml/profil_users.xml"/>
```

```
<transform src="kiwis/xsl/{2}_users.xml"/>
<security rights="conception | instantiation"/>
</entry>
```

L'élément *security* contient la liste des droits permettant d'accéder à cette page. Dans le cas présent, seuls les utilisateurs ayant, soit le droit de *conception* du système, soit le droit *d'instanciation* sur le système, peuvent accéder aux pages qui correspondent au pattern *\*/\*\_users.html*. Il est parfois nécessaire que certaines pages ou ressources, comme des images, soient accessibles pour le client qui vient de se connecter et sur lequel on n'a aucune information. Dans ce cas, nous avons prévu des entrées qui ne comportent pas de balise *security* et donc qui ne supposent pas de contrôle au niveau des droits d'accès, qui sont délivrés immédiatement.

La possibilité de paramétrer les feuilles des styles nous permet de ne pas surcharger les pages web de scripts JavaScripts qui, en récupérant les cookies, vont adapter la présentation et les modalités de navigation. Puisqu'en stockant dans les objets session toutes ces informations au moment de publication d'une page web, le système va transmettre à la feuille de style tous les paramètres de la présentation. Un document html va ainsi être généré, conforme aux préférences des utilisateurs, et ne nécessitant plus de traitement au moment du chargement.

En ce qui concerne l'accès aux servlets de l'application et les traitements des différents formulaires html à traiter par celles-ci, une stratégie assez simple a été mise en place. Avant de commencer à traiter effectivement la requête, la servlet récupère les droits de celui qui l'envoie en accédant à l'objet session correspondant, et puis selon ces droits il la traite ou renvoie un message indiquant que l'opération n'est pas autorisée.

## **4.4 Le nouveau prototype KIWIS**

Cette section présente les principales étapes du développement du ce nouveau prototype.

### **4.4.1 Schéma fonctionnel**

En fonction des besoins recensés lors du chapitre 3 décrivant les fonctionnalités de KIWIS, notre prototype est constitué de 6 modules (figure 4.4) : quatre modules constituant le noyau du système (« User Manager », « Presentation Manager », « Data Manager » et « Query Manager ») et un module externe (le module Infrastructure).

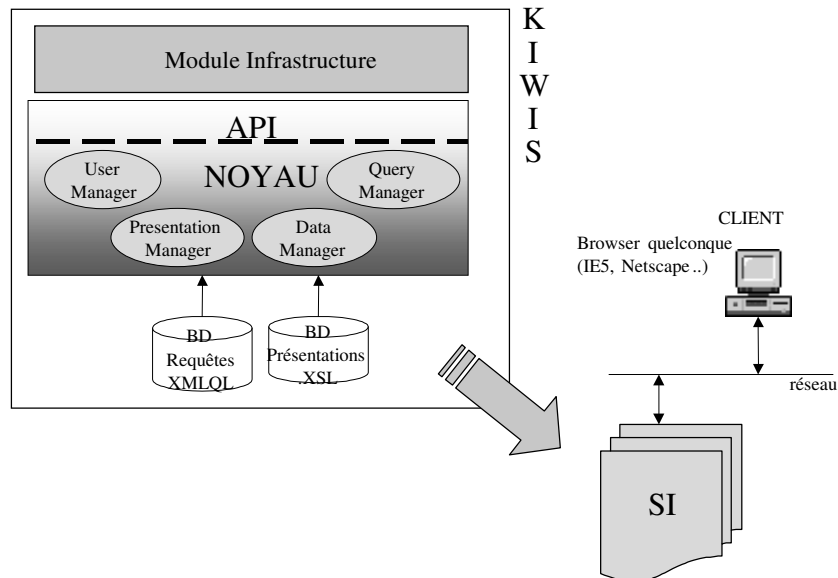


Figure 4.4 : Architecture fonctionnelle du système KIWIS

Le module « User Manager » est chargé de l'authentification des utilisateurs. Il gère les accès aux données du SI en fonction des droits de l'utilisateur. Enfin, il maintient les informations relatives au modèle utilisateur.

Le module « Présentation Manager » maintient la bibliothèque de présentations et sélectionne les informations nécessaires à la présentation du contenu informationnel du SI selon les caractéristiques de l'utilisateur.

Le module « Data Manager » gère le contenu informationnel du SI (format XML).

Le module « Query Manager » permet d'accéder aux données d'application au travers de requêtes sur les données.

Le module « Infrastructure » contient l'ensemble des composants logiciels nécessaires à l'installation de notre générateur de SI. Une fois ces composants installés, le concepteur peut définir des SI qui seront accessibles via le web par les différents postes clients. Ce module contient, entre autres, l'outil de publication de pages XML à travers de web.

Une fois les composants logiciels déployés sur le poste du concepteur (par l'intermédiaire du module « Infrastructure »), le site Web <http://hostname:8080/kiwis> est accessible. Lorsqu'un client se connecte via son navigateur préféré au système, le module « Session manager » ordonne au « Web page generator » de renvoyer à l'utilisateur un formulaire de connexion. Les informations saisies sont renvoyées au « User Manager » qui :

- vérifie l'existence de l'utilisateur dans le système
- identifie le groupe auquel l'utilisateur appartient

- crée une session basée sur ses informations et génère une interface personnalisée à l'utilisateur correspondant.

#### 4.4.2 Définition de l'architecture

Un pattern architectural est l'expression d'un schéma fondamental d'organisation pour des systèmes logiciels. Jim Conallen identifie dans les applications Web d'aujourd'hui 3 patterns architecturaux couramment utilisés [CONA2000] :

- le pattern du *client Web léger* (Thin Web Client), est employé surtout pour les applications destinées à l'Internet pour lesquelles la configuration du client n'est pas contrôlée. La logique métier est exécutée sur le serveur.

Le client ne nécessite qu'un navigateur web standard compatible avec les formulaires.

- le pattern du *client Web lourd* (Thick Web Client), permet à une partie significative de la logique métier d'être exécuté sur le poste client. Les communications avec le serveur passent toujours par http.

En général le client met en œuvre HTML dynamique (scripts clients tels que JavaScript ou VBScript, des applets Java ou des contrôles ActiveX) pour exécuter la logique métier (le plus souvent on y retrouve la validation des données saisies).

- le pattern *livraison Web* (Web Delivery), permet en plus de l'emploi de HTTP pour la communication client-serveur, la mise en œuvre d'autres protocoles, tels que DCOM, IIOP, RMI, dans les systèmes d'objets distribués.

Ce type de pattern est plus approprié quand un contrôle suffisant sur les configurations client et réseau existe. Il n'est donc pas particulièrement adapté aux applications destinées à l'Internet, pour lesquelles il n'y a pas ou peu de contrôle sur les configurations client.

Le prototype KIWIS met en œuvre, le pattern client Web lourd aussi pour la partie conception d'un nouveau SI (l'utilisation du DHTML pour la rendre plus conviviale et effectuer la validation des données saisies), que pour la partie utilisation du SI côté client et notamment la prise en compte de l'adaptabilité.

Les composants majeurs de notre architecture sont les suivants (cf figure 4.5) :

- le *navigateur client* est un navigateur HTML standard compatible avec les formulaires, acceptant l'utilisation de cookies et de JavaScript,
- le *serveur Web* est le point d'accès principal pour tous les navigateurs clients, et en fonction des requêtes des pages HTML, déclenche des traitements côté serveur (par

exemple, la publication de données XML en HTML via l’outil de publication que nous avons conçu),

- le protocole *http* est le protocole utilisé pour réaliser la communication entre les navigateurs client et le serveur Web,
- les *pages HTML* sont des page Web qui comprend une interface utilisateur et du contenu (texte, des formulaires de saisie, etc)
- le *serveur application* est le principal exécuteur de la logique métier et se trouve sur la même machine que le serveur Web,
- les *scripts* sont des scripts de type JavaScript inclus dans des pages HTML et interprété par le navigateur,
- les *documents XML* contiennent des données brutes sans indication aucune pour la mise en page.

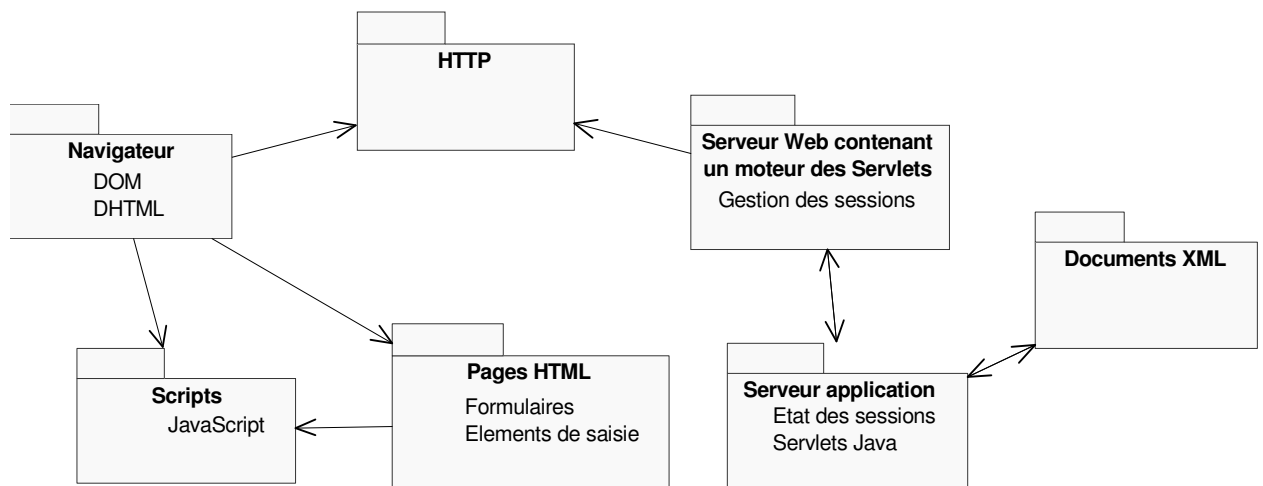


Figure 4.5 : Vue logique de l’architecture de KIWIS-APP

#### 4.4.3 Conception du prototype avec WAE

Nous l’avons vu dans le paragraphe précédent, les patterns architecturaux de base des applications Web impliquent tous l’usage de pages Web. Celles-ci font le lien entre le navigateur et le reste du système. Il est déterminant de consigner les pages Web en tant qu’éléments de première importance dans le modèle et de les représenter au côté des classes. Pour réaliser cela, J. Conallen propose une extension d’UML [UML] pour les applications Web : WAE (Web Application Extension) [CONA2000].

Cette section comprend 2 parties : tout d’abord une description des besoins fonctionnels du prototype selon la syntaxe UML, et ensuite une modélisation de l’application à créer à l’aide de l’extension WAE.

#### 4.4.3.1 Description UML des besoins fonctionnels

Notre prototype s’adresse à 2 acteurs principalement. : tout d’abord le concepteur pour définir et concevoir un nouveau SIW, ensuite l’utilisateur final pour accéder au nouveau SIW.

- Acteur « concepteur » : personne qui spécifie le SI en exécutant les actions suivantes :
  - Construction du contenu informationnel du SI (modèle et instances)
  - Déclaration de groupes et d’utilisateurs
  - Conception/Personnalisation des pages Web

Le cas d’utilisation principal est l’entrée dans le système, après identification du concepteur du système. Ce cas d’utilisation est étendu par les cas d’utilisations de la figure 4.6.

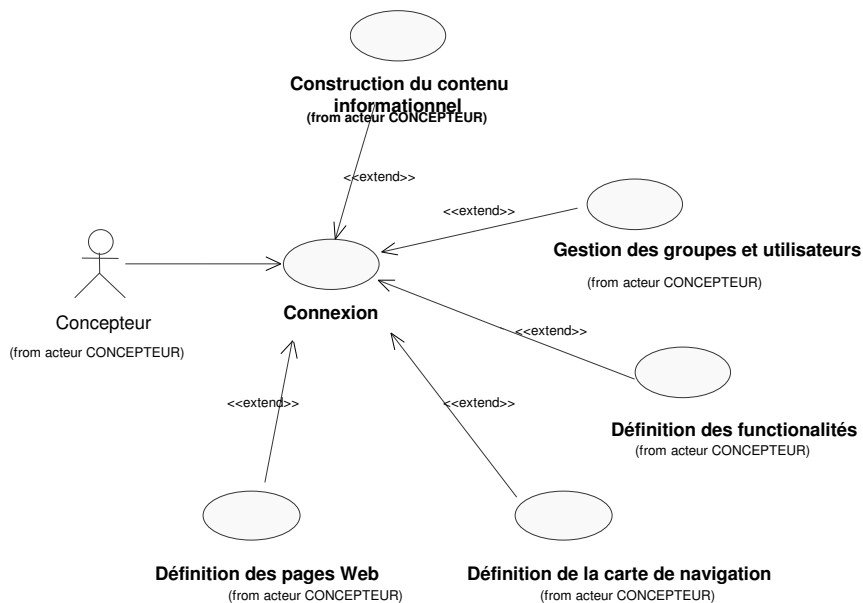


Figure 4.6 : Diagramme des cas d’utilisations pour l’acteur CONCEPTEUR

- Acteur « utilisateur » : personne qui utilise le système pour les objectifs suivants :
  - Consulter les données du système d’information
  - Mettre à jour le système, selon ses droits
  - Mettre à jour son profil

Comme pour le concepteur, le cas d'utilisation principal est l'entrée dans le système. Ce cas d'utilisation est étendu par les cas d'utilisations de la figure 4.7.

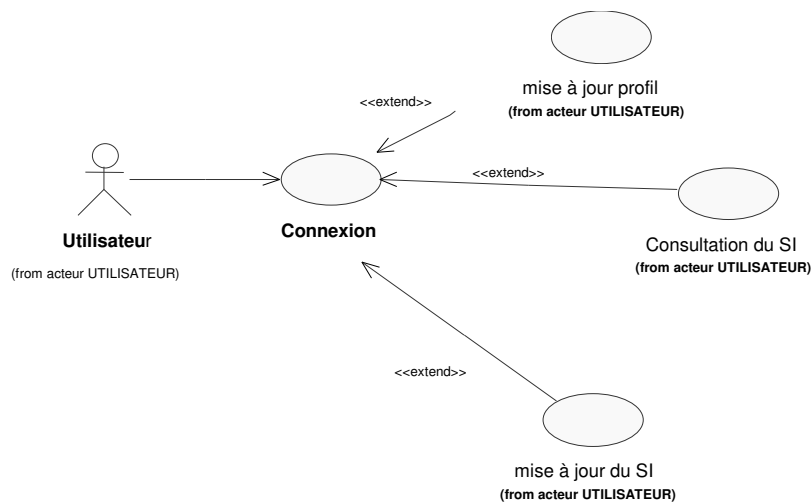


Figure 4.7 : Diagramme des cas d'utilisations pour l'acteur UTILISATEUR

La section suivante décrit les diagrammes de séquence de quelques cas d'utilisations, selon une syntaxe UML enrichie de stéréotypes pour application Web (WAE).

#### 4.4.3.2 Modélisation WAE

WAE (Web Application Extension) proposé par J. Conallen, complète la notation UML avec une sémantique et des contraintes permettant d'intégrer des éléments d'architecture Web dans le reste du modèle système. Il s'agit de consigner l'exécution de la logique métier des pages Web, des scripts et des composants côté client.

La phase d'analyse en UML se termine par un diagramme de séquence du modèle d'analyse qui sera convertit pendant la phase de conception en un diagramme de séquence WAE, dans lequel les classes Interface du modèle d'analyse seront converties en pages client et les classes Contrôle en pages serveur.

L'objectif de ce paragraphe est de montrer l'enchaînement des différents composants Web de notre prototype sur 2 cas d'utilisations.

- cas d'utilisation « Connexion » pour l'acteur Concepteur

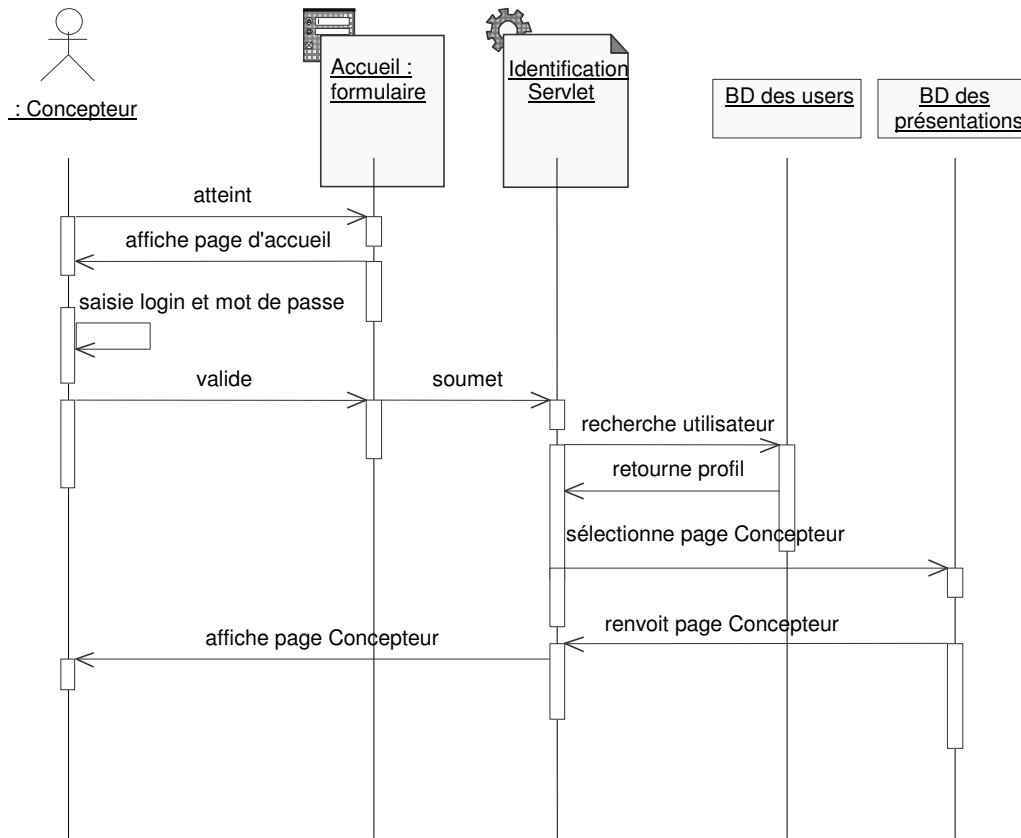


Figure 4.8 : Diagramme WAE « connexion » du Concepteur

Le concepteur saisit l'URL du site dans son navigateur. La page d'accueil contenant un formulaire de connexion lui est renvoyée. Après saisie de son *login* et de son *mot de passe*, il valide le formulaire, ce qui a pour effet de déclencher la servlet «IdentificationServlet». Le premier traitement de la servlet est la recherche de l'existence de l'utilisateur connecté dans la base des profils. Si l'utilisateur n'est pas authentifié, la page d'accueil lui est renvoyée avec un message d'erreur (non modélisé dans le diagramme de la figure 4.8). Sinon, le profil récupéré est chargé dans un objet Session (qui reste de côté server) et la page de conception d'un SIW est renvoyée au navigateur, par la servlet.

- cas d'utilisation « Définition d'une fonctionnalité » pour l'acteur Concepteur

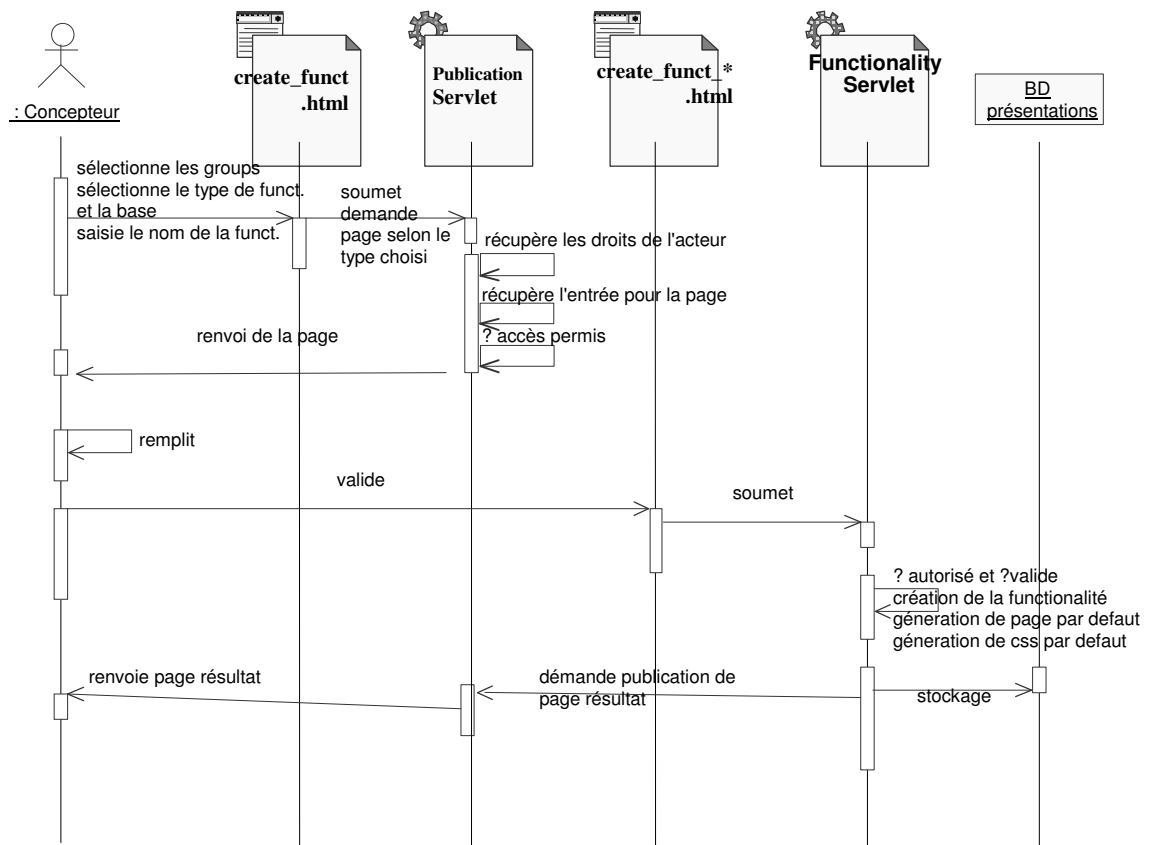


Figure 4.9 : Diagramme WAE « Définition d'une fonctionnalité » (Concepteur)

Ce cas d'utilisation correspond à une des étapes de conception d'un SIW.

Le concepteur sélectionne dans la page de conception, l'option correspondant à la définition d'une nouvelle fonctionnalité du système. Cette action a pour effet de déclencher l'exécution de la servlet « `PublicationServlet` », qui après la vérification des droits de client, renvoie la page de définition d'une fonctionnalité. Le concepteur la remplit et la valide. La servlet « `PublicationServlet` » est déclenchée. Elle récupère de la 1<sup>ère</sup> page de définition le nom, le type, la classe ou l'association selon le type et les groupes pour lesquels on veut définir la fonctionnalité. Tous ces paramètres sont alors utilisés pour générer la 2<sup>ème</sup> page pour finir le processus de définition d'une fonctionnalité. Le concepteur remplit la page renvoyée et en validant ses choix, un appel vers la servlet « `FunctionalityServlet` » s'effectue. Celle-ci tout d'abord se prononce sur la validité de la demande et ensuite récupère les données envoyées générant par la suite la fonctionnalité qui sera stockée dans la BD des présentations. En même temps une page web, et une page CSS correspondant à cette fonctionnalité seront générées. Une fois ces traitements réalisés, la servlet renvoie au navigateur une page qui contient le résumé des opérations effectuées.

## 4.5 L'interface concepteur de KIWIS

### 4.5.1 La connexion

Le concepteur doit s'authentifier en saisissant le login admin et le mot de passe. Afin de créer un nouveau système d'information aucun système déjà présent ne doit être sélectionné, cela indiquerait que le concepteur veut modifier ce système.

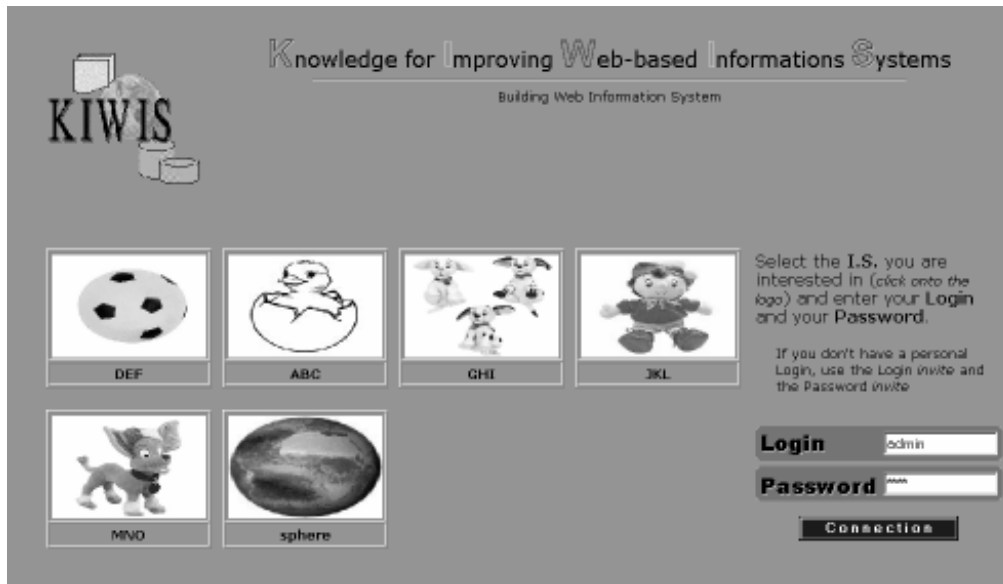


Figure 4.10 Page d'authentification de KIWIS

La connexion en tant de concepteur, c'est-à-dire en tant qu'utilisateur du SI qui peut modifier la structure du SI peut se faire aussi par des utilisateurs habituels de ce système s'ils ont le droit de conception. Mais pour concevoir des nouveaux systèmes d'information il est nécessaire que le login de concepteur (*admin*) soit authentifié.

Au moment de la soumission de formulaire d'authentification la servlet qui se charge de l'authentification des utilisateurs exécute les actions suivantes :

1. invalidation de la session en cours (si une existe)
2. récupération du *login*, du *mot de passe* et le nom de système
3. création d'une requête adresse au module Query Manager qui interroge la base de données contenant les informations sur l'utilisateur du système.
4. récupération du profil utilisateur (ses droits et ses préférences)
5. création d'une nouvelle session à laquelle on associe le profil utilisateur obtenu
6. renvoie d'une page d'accueil, selon les droits de l'utilisateur

Dans le cas où l'authentification n'aboutit pas (lors de l'étape numéro 3, le login ou le mot de passe ne sont pas valides) alors la page d'authentification est renvoyée à l'utilisateur.

Lors de la sixième étape, dont la mesure ou l'utilisateur qui vient de s'authentifier est le concepteur alors une page spécialement conçue lui sera renvoyée.

#### 4.5.2 La page du concepteur

Cette page permet au concepteur de gérer tous les systèmes d'informations installés au sein du KIWIS.

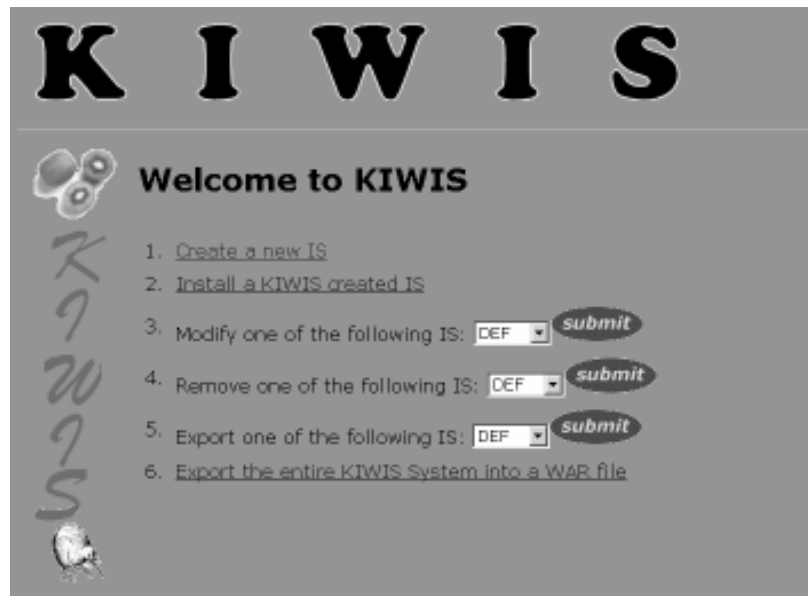


Figure 4.11 La page d'accueil du concepteur dans KIWIS

Maintenant, nous allons nous concentrer sur la démarche de création d'un nouveau SI, et présenter la démarche à suivre pour concevoir un tel système avec KIWIS.

En cliquant sur l'option « Create a new IS », le concepteur est dirigé vers la page de création d'un nouveau système.

#### 4.5.3 Création d'un nouveau système d'information

Le concepteur saisit les champs *IS Name*, *Short Description* et *Detailed Description*, choisit un logo pour ce système et valide ses choix en cliquant sur le bouton *Create*. Le lien *show/hide* permet au concepteur de visualiser la liste des SI installés sur KIWIS.

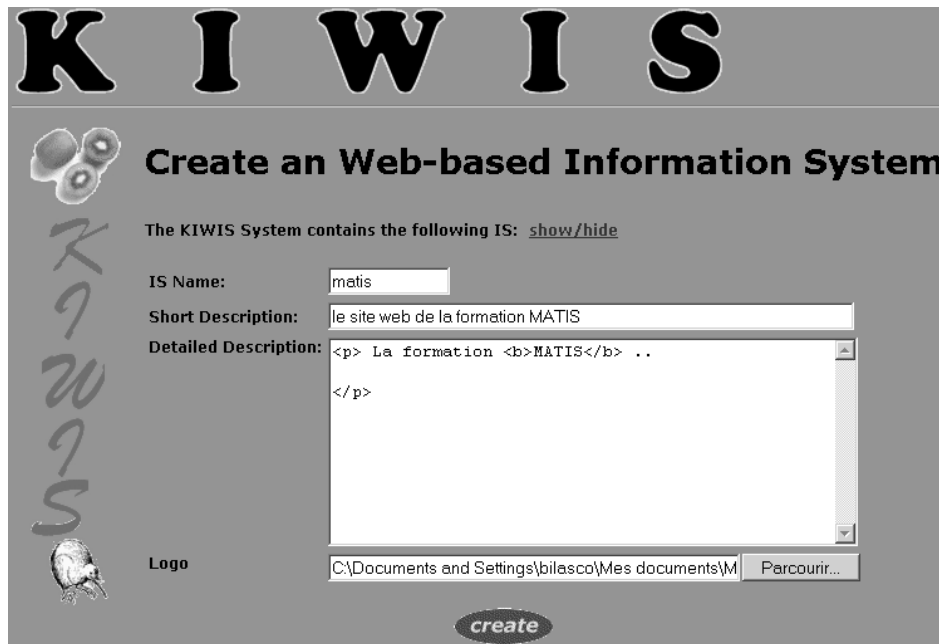


Figure 4.12 La page de création d'un nouveau SI

Comme dans toutes les pages qui composent l'interface de conception d'un nouveau SI, la validation des données saisies se fait à l'aide des scripts Javascript, mais aussi au niveau des servlets qui sont derrière ces pages, une validation des données pour empêcher le traitement des requêtes qui ne sont plus valides, ou qui ne sont pas générées par les pages de conception KIWIS, est opérée.

Suite au traitement du formulaire par la servlet « ISServlet » qui se charge de la gestion globale des SI installés au sein du KIWIS, un répertoire portant le nom du SI est créé. Dans ce répertoire nous retrouvons la structure de SI :



Figure 4.13 Structure d'un SI en KIWIS

Le répertoire *xml*, contient les bases de données du SI, le *xsd* contenant les définitions de celles-ci, selon la syntaxe Xschema. Les répertoires *xsl* et *css* correspondent à la bibliothèque des présentations. Les deux autres répertoire : *images* qui stocke les images liée à ce SI (par exemple le logo) et *work* qui stocke les données intermédiaires obtenues suite aux requêtes effectuées sur la basée de données.

Si toutes ces opérations aboutissent une page « wizard » est retournée au concepteur afin de le guider tout au long de la démarche de conception.

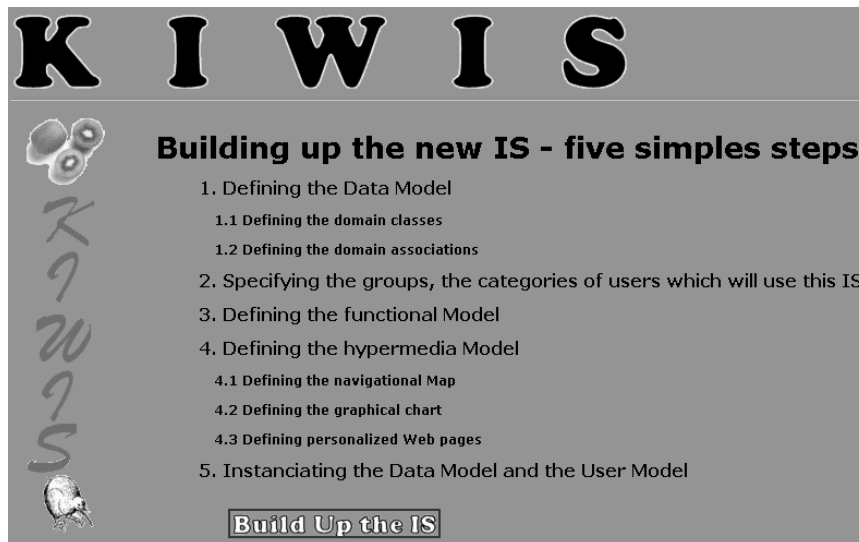


Figure 4.14 Présentation de la démarche de conception

En cliquant sur l'image *Build Up the IS* le concepteur accède à la page de définition du modèle de données et commence son travail par définir les classes du domaine.

#### 4.5.4 Les pages de définition de modèle de données

En premier lieu, le concepteur va décrire les classes du système :

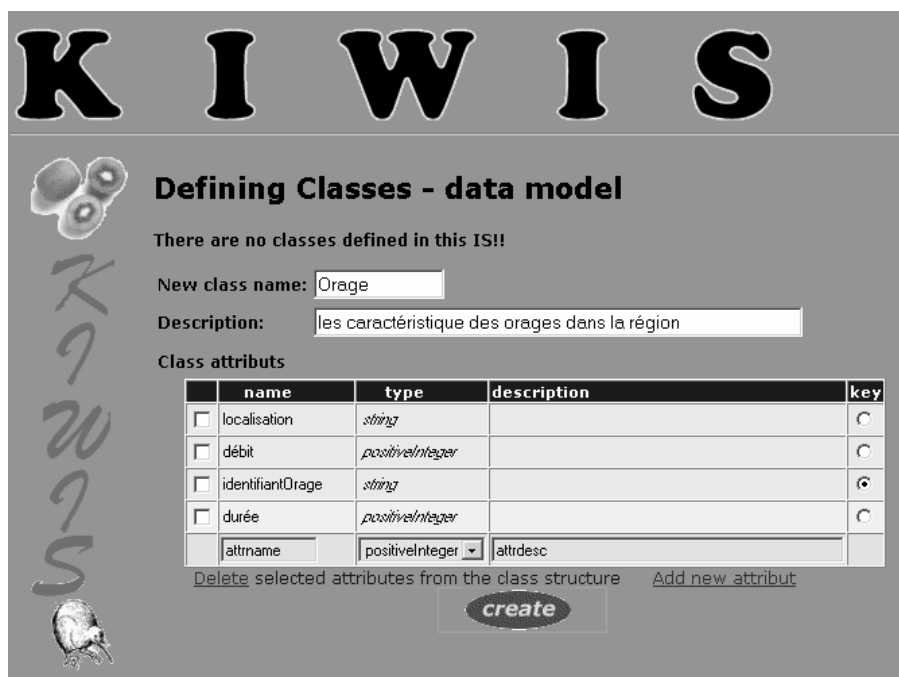


Figure 4.15 Création d'une nouvelle classe

Il saisit le nom de la classe, éventuellement il indique aussi une courte description du rôle de celle-ci au sein du système. Ensuite, il ajoute les attributs de la classe en saisissant leur nom, leur type, et leur description si nécessaire. La validation d'un attribut se fait en cliquant sur le lien *Add new attribut*. A tout moment, le concepteur peut effacer des

attributs, les sélectionnant et ensuite cliquant sur *Delete*. Après avoir introduit tous les attributs, le concepteur doit choisir parmi ceux-ci, un attribut qui identifie uniquement une instance de cette classe.

Après avoir défini tous les classes du SI, le concepteur va passer à la définition des associations du SI :

**K I W I S**

**Defining Association - data model**

There are no associations defined in this IS!!

New assoc name:

Description:

Select the roles for this association

role name	role description
<input type="checkbox"/> Crue	les crues dans la région
<input type="checkbox"/> Moyen	les moyens déployable pour secourir les habitants
<input type="text" value="Moyen"/>	

[Remove selected roles from this association](#) [Add new Role](#)

Additional attributes for this association

name	type	description
<input type="checkbox"/> nombre	integer	le nombre de moyens de ce type employés
<input type="text" value="nombre"/>	<input type="text" value="integer"/>	le nombre de moyens de ce type employés

[Delete selected attributes from the assoc structure](#) [Add new attribut](#)

**create**

Figure 4.16 Création d'une nouvelle classe

Dans cette nouvelle version de KIWIS nous permettons au concepteur de définir autant des rôles qu'il souhaite pour ses associations. L'ajout d'un rôle se fait comme dans le cas des attributs d'une classe. Après la création de l'association une réponse de ce type est renvoyée au concepteur

**K I W I S**

The MoyensDeployesCrues association was succesfully registred in the system.

If you want to create another class [click here](#).

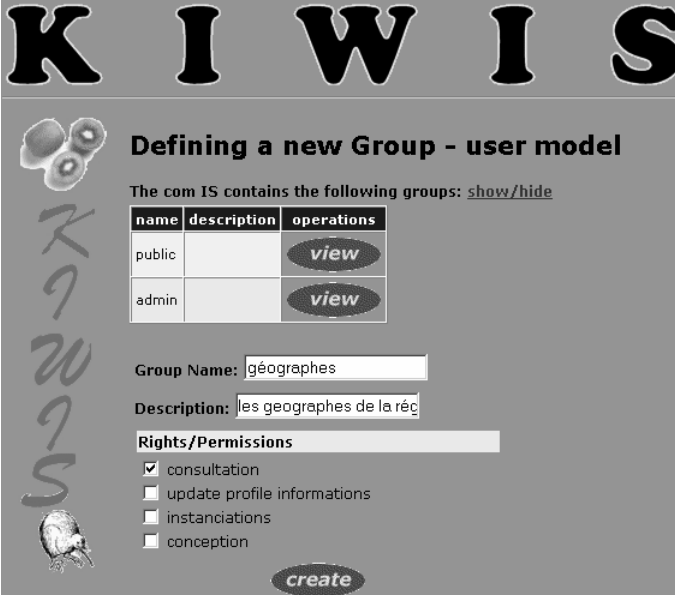
If you have finished describing the associations of this IS, you can pass to the next step [Describing the groups of users](#)

Figure 4.17 La page réponse de création d'une association

qui peut choisir de définir une nouvelle association ou de passer à la deuxième étape du processus de conception : la définition des groupes de ce SI.

#### 4.5.5 Interface de définition des groupes

Le concepteur va créer les groupes des utilisateurs qui sont concernés par le système. Il va associer aussi à ces groupes tous les droits dont ses potentiels membres peuvent détenir.



**K I W I S**

**Defining a new Group - user model**

The com IS contains the following groups: [show](#)/[hide](#)

name	description	operations
public		<a href="#">view</a>
admin		<a href="#">view</a>

Group Name:

Description:

**Rights/Permissions**

- consultation
- update profile informations
- instanciations
- conception

[create](#)

Figure 4.18 La création d'une groupe d'utilisateurs

N'oublions pas que dans la nouvelle approche, les groupes ne sont pas définis forcément par leurs droits, mais par l'espace informationnel qui leur est associé dans le modèle des fonctionnalités. Ainsi à ce moment de la démarche, il faut offrir aux groupes tous les droits dont les membres peuvent bénéficier.

Une page de réponse de même type que celle pour la création des classes est renvoyée permettant au concepteur d'enchaîner les étapes de conception.

#### 4.5.6 Interface de définition du modèle de fonctionnalités

Comme nous avons vu dans le premier paragraphe de ce chapitre, on peut avoir dans ce modèle des fonctionnalités simples construites en partant soit d'une classe, soit d'une association et des fonctionnalités *composées* dans lesquelles on inclut des fonctionnalités simples. Ceci nous a conduit à découper le processus de définition d'une fonctionnalité en deux étapes.

Dans la première étape, on choisit le type de fonctionnalité qu'on veut définir, son libellé et les groupes auxquels elle sera associée. Nous avons laissé la possibilité de définir une fonctionnalité pour plusieurs groupes en même temps, car il est possible que des groupes d'utilisateurs aient des caractéristique communes et donc des informations qui ont le même degré de pertinence pour eux.

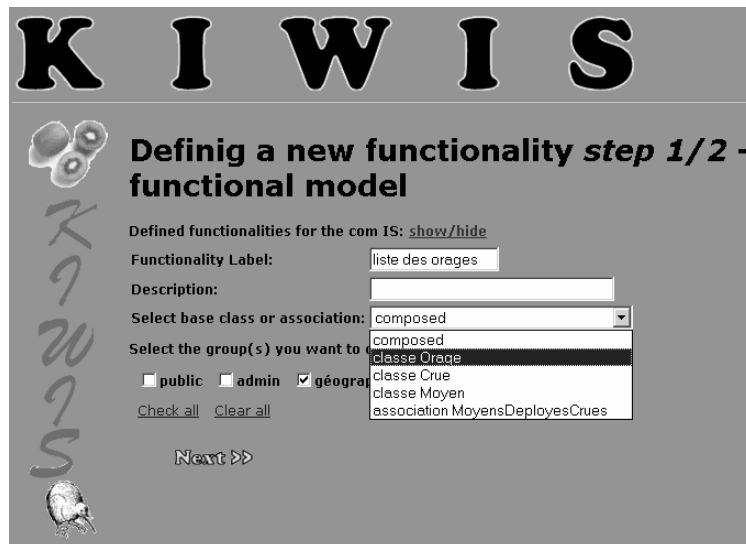


Figure 4.19 La première phase de création d'une fonctionnalité

En fonction des choix du concepteur, la page de conception d'une fonctionnalité *composée*, la page d'une fonctionnalité basée sur une classe ou sur un association lui sera renvoyée. Ici nous ne présentons que la page de conception d'une fonctionnalité basée sur une association, en incluant le principe de la création d'une fonctionnalité associée à une classe et aussi le principe de création d'une fonctionnalité composée.

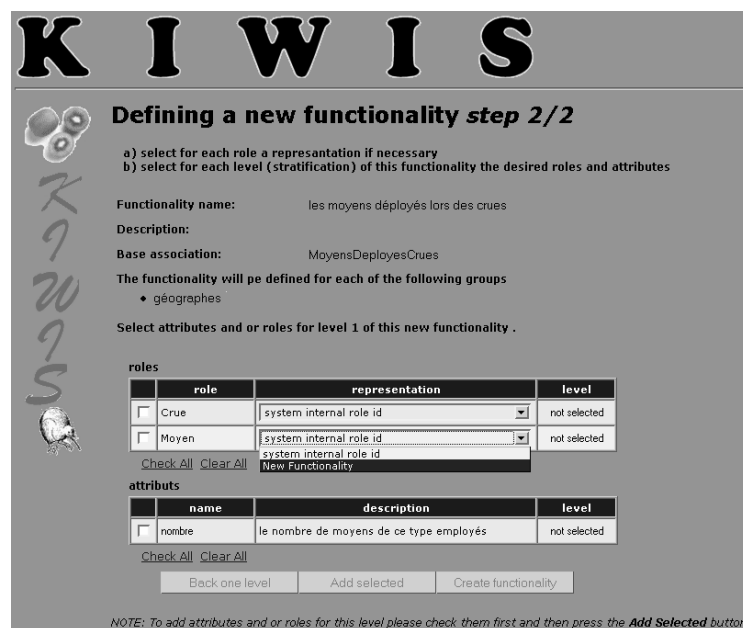


Figure 4.20 La deuxième étape dans la conception d'une fonctionnalité

Le concepteur va choisir les éléments qui seront visibles au premier niveau de détail, il valide ses choix en appuyant le bouton *Add Selected*, et sélectionne les attributs pour les niveaux successifs. Pour les rôles, le concepteur à le droit de choisir leur représentation soit par les valeurs intrinsèque de ceux-ci (*system internal role id*), soit en choisissant une fonctionnalité déjà définie pour ceux-ci ou même d'en définir une nouvelle. Dans le cas présent, comme il n'y a aucune fonctionnalité définie pour la classe *Moyen*, le concepteur

décide de définir une fonctionnalité pour celle-ci. Une fenêtre flottante correspondante à la page de définition d'une fonctionnalité pour une classe s'ouvre en lui permettant de définir cette nouvelle fonctionnalité. Une fois le processus de création de cette fenêtre terminé, la nouvelle fonctionnalité est automatiquement introduite dans la liste des choix pour le ou les rôles correspondants.

#### 4.5.7 Interface de modélisation de la carte de navigation

Une fois que le concepteur a défini toutes les fonctionnalités du site, il peut choisir dans la page réponse qui lui est envoyée après la création réussie d'une fonctionnalité, de passer à l'étape de définition de la carte de navigation.



Figure 4.21 Définition de la carte de navigation 1/2

Le concepteur va sélectionner le groupe sur lequel il veut travailler et ensuite en cliquant sur l'image *create* va définir les liens entre la fonctionnalité choisie et les autres fonctionnalités du système.

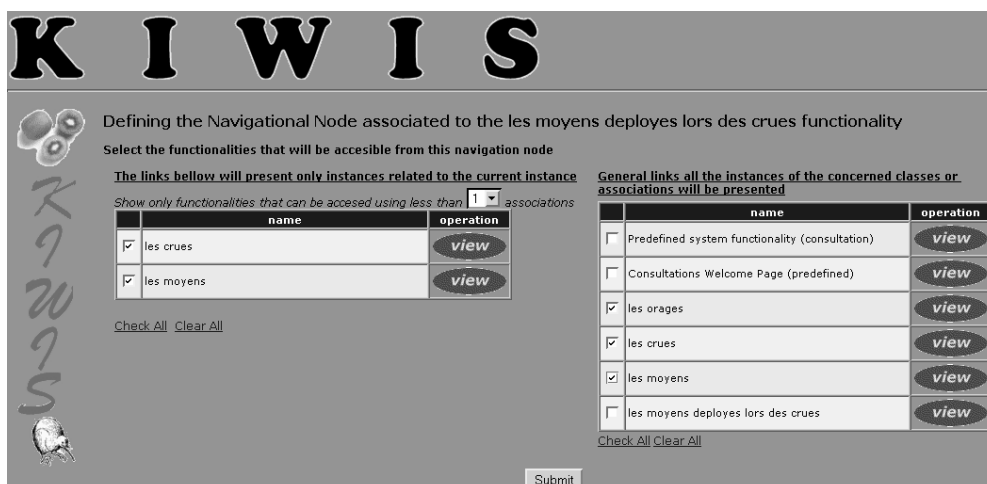


Figure 4.22 Définition de la carte de navigation 2/2

Lors de cette deuxième étape le concepteur va choisir les liens qui seront associés à cette fonctionnalité. On observe qu'il y a deux type des liens :

- des liens *généraux* qui donnent accès à toutes les instances d'une classe ou associations
- des liens *liées aux instances* qui ne vont permettre que la visualisations des instances des classes ou des associations reliées à la base de cette fonctionnalité

Si un nœud de navigation n'est pas défini alors ça ne veut pas dire qu'il n'est pas accessible, mais il ne faut pas que le concepteur oublie de créer des liens vers ceux-ci. Aussi pour permettre la consultation de site, il faut que pour chaque groupe soient définis les liens qui seront accessibles depuis la page d'accueil *Consultation Welcome Page*.

#### 4.5.8 La création d'un utilisateur du système

Après avoir créé le système d'information, tout ce qu'il reste à faire est de définir les utilisateurs qui vont utiliser le système. Par défaut chaque SI généré avec KIWIIS contient deux *logins* : celui de concepteur, *admin*, et celui de l'utilisateur générique *invite*, donc ces deux *logins* sont réservés et ne peuvent pas être utilisés.

Le concepteur choisit donc un *login* qui n'existe pas encore dans le système, saisit le mot de passe, et sélectionne les privilèges qu'il veut accorder à cet utilisateur.

The screenshot shows the 'KIWIIS' web interface for creating a new user. The main heading is 'Defining a new User - user model'. The form includes the following fields and options:

- Login:** marius
- Password:** [masked]
- Re-type Password:** [masked]
- This new user will be part of:** géographes group.
- Rights/Permissions:**
  - consultation
  - update profile informations
  - instantiations
  - conception
- Navigation:**
  - static adaptability  open link in new window
  - dynamic adaptability  track user navigation
- Presentation:**
  - Items will be presented: in a table
  - Items on page: 10
  - Default level detail: 1
- Window characteristics:**
  - no toolbar
  - no scrollbars
  - no menubar
  - no statusbar
  - don't load images

A 'create' button is located at the bottom of the form.

Figure 4.23 Création d'un nouveau *login*

Dans le cas présent, le *géographe marius* aurait seulement les droits de consulter le système et de mettre à jour son profil (changer son mot de passe, changer les préférences de la présentation).

## 4.6 L'adaptabilité de l'interface utilisateur

L'interface utilisateur dans cette version de prototype est assez simple, car nos travaux ont porté surtout sur l'interface concepteur. Tout de même, nous avons essayé d'implémenter les mécanismes de navigation au sein du système d'information et aussi d'anticiper un peu la personnalisation des pages web en associant à chaque composante d'une page web sa propre feuille de style. Pour le moment, les composantes qui sont prises en compte sont les éléments des différents tableaux.

Le seul aspect traité concernant l'adaptabilité de la présentation est la prise en compte des préférences de l'utilisateur sur le navigateur web (pas de barre de menu, pas de barre de défilement etc.).

La page d'accueil indique aux utilisateurs les droits qu'il a dans le système et lui propose d'exécuter une action parmi celles qui lui sont permises :

**K I W I S**

---

**Welcome to com a KIWIS generated Web Information System**

**1. Update your Profile**  
**2. Consult this IS**

page g n r e automatiquement par le syst me KIWIS

## 5. Conclusions et perspectives

Même si certains aspects du prototype antérieur ne se retrouvent pas dans ce nouveau KIWIS et notamment la possibilité de personnaliser la présentation de l'information en fonction des groupes d'utilisateurs, celui-ci constitue un pas en avant vers un outil plus puissant de génération des systèmes d'informations basée sur le web.

Le nouveau modèle des fonctionnalités donne la possibilité d'associer, à chaque groupe d'utilisateurs, son propre espace informationnel stratifié selon les besoins du groupe. Ainsi, on peut construire des systèmes d'informations avec un degré plus élevé d'adaptabilité en terme de contenu informationnel qu'auparavant.

La personnalisation de la carte de navigation qui devient propre à chaque groupe, permet de mieux adapter la navigabilité au sein des SI construits avec KIWIS.

Les travaux sur ces deux modèles, et la révision de modèle de données ont été assez longs et ils ont nécessité de nombreux retours en arrière. Aussi, le passage de Cocoon vers notre propre serveur de publication, intégrant les aspects de sécurité a nécessité un temps important, ce qui a fait que les aspects hypermédia des systèmes comme par exemple : la composition de la présentation, la diversité des formats hypermédia supportés n'ont pas été encore suffisamment développés et font partie des perspectives de ce travail.

Plus généralement, l'un des plus importants travaux en perspective est une réflexion sur le modèle hypermédia du KIWIS et ensuite une implémentation de celui-ci.

Le passage de l'application KIWIS en Cocoon se présente comme un autre projet futur. L'intégration de KIWIS dans l'environnement de publication Cocoon permettra d'exploiter tous les atouts de Cocoon.

Les aspects d'adaptabilité dynamique seront traités dans les versions à venir, car le fait que le suivi de session mis en place en utilisant les objets session, permet le stockage des données complexes qui peuvent modéliser l'activité de l'utilisateur et permettent de construire de manière automatique un profil utilisateur évoluant en fonction duquel le système pourra adapter le contenu et la présentation des informations délivrées.

## Bibliographie

[AROM2000] M. Page, J. Gensel, C. Capponi, C. Bruley, P. Genoud, D. Ziébelin, «*Représentation de connaissances au moyen de classes et d'associations : le système AROM*», LMO 2000.

[BERN2000] J.C. Bernadac, F.Knab, F.Lepoivre, F.Rivard, C.Sannier, «*XML et Java* », Eyrolles, Novembre 2000.

[CHAR2000] I. Charon, «*Le langage Java : concepts et pratique* », Hermes Sciences, mars 2000.

[CHAZ2001] <http://site.voila.fr/xmlschema/xschema.htm>.

[COCO] <http://xml.apache.org/cocoon>

[CONA2000] J.Conallen, «*Concevoir des applications web avec UML* », Eyrolles, septembre 2000, <http://www.conallen.com/technologyCorner/webextension/welcome.html>.

[DOM] <http://www.w3.org/DOM/>

[DONS1998] D. Donsez, «*Sécurité et Internet* », 1995-1998 <http://www.univ-valenciennes.fr/limav/donsez/pub/spedago/securite/secuip1.pdf>

[FLAN2000] D. Flanagan, «*Java in a Nutshell* », O'reilly, mai 2000.

[HABR1992] H. Habrias, revue Génie Logiciel & Systèmes Experts, mars 1992.

[HORS1999] Cay S. Horstmann, G. Cornell, «*Au cœur de Java 2* » volume1, Campus Press, mars 1999.

[HUNT1999] J. Hunter, W.Crawford, «*Java Servlets* », Edition française, O'Reilly, juillet 1999.

[JAVA] <http://java.sun.com/docs/white/index.html>

[KWEEL] <http://db.cis.upenn.edu/Kweelt>

[LEFE2000] A.Lefebvre (Groupe SQLI), «*La relation client sur Internet et le site intelligent* », novembre 2000

[MICH2000] A.Michard, «*XML langage et applications* », Eyrolles, décembre 2000.

[NETS] <http://www.netsilon.com/>

[ROUS2000] G. Roussel, E. Duris, «*Java et Internet* », Vuibert, janvier 2000.

[SAX] Simple API for XML, <http://www.megginson.com/SAX>

[TOMC] <http://jakarta.apache.org/tomcat/>, <http://java.sun.com/products/jsp/tomcat/>

[UML] <http://www.omg.org>

[VILL2001] M.Villanova, J.Gensel, H.Martin, « *Progressive Access to Knowledge in Web Information System through Zooms* », In Proc. of Object-Oriented Information Systems (OOIS2001), Calgary, Canada, August 2001.

[XML2000] recommandation W3C, 2000 <http://www.w3.org/TR/2000/REC-xml-20001006>

[XPAT1999] recommandation W3C, novembre 1999 <http://www.w3.org/TR/xpath>

[XQUE2001] working draft W3C, juin 2001 <http://www.w3.org/TR/xquery/>

[XSCH2001] recommandation W3C, mai 2001 <http://www.w3.org/TR/xmlschema-0/>,  
<http://www.w3.org/TR/xmlschema-2/>

[XSLT1999] recommandation W3C, 1999 <http://www.w3.org/TR/1999/REC-xslt-19991116>