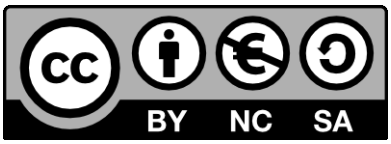
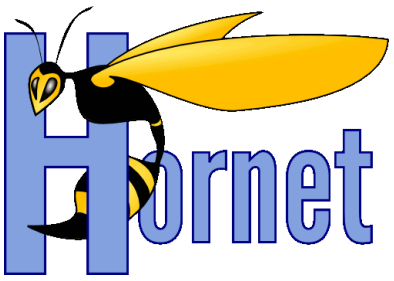


Guide du développeur Hornet



Développement Hornet 3.10

Version : 1.0 du 05/01/2015 - **Etat** : Validé

SUIVI DES MODIFICATIONS

Version	Auteur	Description	Vérification	Date
1.0	I. Kone / O. Rousseil	Mise à jour pour Hornet 3.10		

DOCUMENTS DE REFERENCE

Titre
HORNET_GUI_Création d'un projet Hornet_3.10_1.0
HORNET_GUI_Guide de paramétrage_1.4

SOMMAIRE

TABLEAUX

FIGURES

1 Principes et fonctionnement de Hornet

1.1 Les principes

1.1.1 « Progressive Enhancement » ou l'amélioration progressive

« L' **amélioration progressive** est une manière de concevoir un site web qui prend très largement en compte l'accessibilité, la sémantique et le référencement. En séparant strictement le fond (le contenu proposé à l'utilisateur) et la forme (l'enjolivement et les interactions avancées), cette technique permet de présenter un contenu simple et de rendre un service minimum à tous les utilisateurs, quel que soit le débit de leur connexion ou leur navigateur, tout en *améliorant progressivement* l'affichage proposé en fonction de l'équipement de l'internaute. »

Source : http://fr.wikipedia.org/wiki/Am%C3%A9lioration_progressive

Les pages web sont ainsi composées de trois parties :

- La partie sémantique en html ou xhtml (le fond),
- La partie présentation avec des styles CSS (la forme),
- La partie interactive, dynamique et événementielle avec du javascript (la forme).

Le service à l'utilisateur doit être « augmenté et enrichi » progressivement par chaque couche. Généralement la partie interactive du « *Progressive Enhancement* » inclut des échanges de type Ajax, d'où l'appellation « *Hijax* » fréquemment rencontrée pour la combinaison du *Progressive Enhancement* et d'Ajax.

1.1.2 Séparation des rôles

Les trois couches utilisées pour les pages web ont des rôles et usages différents détaillés dans les paragraphes suivants.

1.1.2.1 Partie sémantique

Cette partie est implémentée sous forme de document HTML ou XML conformément aux normes w3c et en respectant les règles d'accessibilité.

Aucun élément de présentation ne doit être présent dans ces documents (hormis les liens d'identification et de classe CSS pour la présentation).

Les éléments HTML doivent être utilisés conformément à leur usage prévu par le w3c, par exemple :

- Les « tables » HTML doivent être utilisées essentiellement pour présenter des données sous forme de tableaux, et non pour structurer l'affichage.

1.1.2.2 Partie Présentation

La partie présentation est réalisée via les styles CSS et les éléments HTML de type groupement (div, p, span …).

1.1.2.3 Partie Interactive

La partie interactive est réalisée en JavaScript à l'aide de « Yahoo ! User Interface Toolkit ».

1.1.3 Yahoo ! User Interface

1.1.3.1 Généralités

"The YUI Library is a set of utilities and controls, written with JavaScript and CSS, for building richly interactive web applications using techniques such as DOM scripting, DHTML and AJAX. YUI is available under a BSD license and is free for all uses.
YUI is **proven**, **scalable**, **fast**, and **robust**. Built by frontend engineers at Yahoo! and contributors from around the world, it's an industrial-strength JavaScript library for professionals who love JavaScript."

Il existe actuellement deux versions de YUI : YUI2 et YUI3.

Depuis la version 3.5 d'Hornet, seule la version YUI3 est utilisée dans Hornet. « 2in3 » n'est plus inclus dans le framework depuis cette version.

Tableau 1 : Correspondance entre version Hornet et version de YUI utilisée.

Version Hornet	Versions YUI
3.6	3.17.2
3.5	3.15.0
>= 3.1 et <= 3.4	3.8.1 / 2.9.0
<= 3.0	3.4.1 / 2.9.0

1.1.3.2 Outillage YUI pour le développement

YUI propose de nombreux outils et utilitaires pour les développements comme :

- Un logger à niveau « façon log4j »,
- Un compresseur de JavaScript et CSS,
- Un système de build basé sur Ant pour faciliter le développement et l'intégration d'un composant,
- Un framework de tests unitaire, à la Junit,
- Profiler Javascript
- Plugin Firebug Yahoo ! YSlow pour les performances.

1.2 Fonctionnement

Le fonctionnement vu par l'utilisateur dans un navigateur web est similaire à celui d'un site web classique. Cependant, il y a un certain nombre de différence au niveau de la structure des pages web et de l'emploi des composants comme :

- Les pages web peuvent être utilisées sans JavaScript, sans images (et autres informations visuelles) ou sans styles CSS,
- Chaque document HTML est conforme aux normes w3c et utilise les éléments et attributs HTML adéquat avec l'information véhiculée.

Trois grands modes d'utilisation sont possibles :

1. « web 1.0 » :
   L'utilisateur accède à des pages utilisant des styles, images et JavaScripts et chaque clic et appel entraine le chargement complet d'une nouvelle page dans le navigateur,

2. Hijax :
   Ce mode complète le précédent en ajoutant une couche interactive javascript sans perturber le fonctionnement « web 1.0 » de la page. Le code JavaScript intercepte certaines actions utilisateurs et remplace le chargement de nouvelle page par des appels Ajax et des manipulations du DOM pour mettre à jour la page avec les informations provenant des requêtes Ajax.
   Par exemple, dans le cas d'une liste de 50 items affichée dans un tableau de 25 items sur deux pages, la pagination est implémentée en Ajax et seulement le contenu du tableau est modifié lors de la pagination sans entrainer de chargement complet du document HTML.

3. Single Page Application (SPA) :
   Le mode SPA étend le fonctionnement Hijax à toute une application web.
   Dans le mode Hijax, les actions interceptées correspondent généralement aux fonctionnalités métiers représentées par la page elle-même, les changements de fonctionnalités entrainant le chargement complet d'une nouvelle page.
   Par exemple, dans le cas d'une liste de 50 items affichée dans un tableau de 25 items sur deux pages, la pagination est implémentée en Ajax et seulement le contenu du tableau est modifié lors de la pagination alors que l'accès à une autre fonctionnalité via le menu entraine le chargement complet de la page.

Dans le mode SPA, toutes les actions sont interceptées et remplacées par des appels Ajax modifiant partiellement le DOM de la page, il n'y a alors plus de chargement complet de page.
Ce mode permet une grande richesse et interactivité des applications web, mais il nécessite un cadre de conception et développement uniformisé entre les pages et fonctions.

Les paragraphes suivants présentent de façon plus détaillées les caractéristiques essentielles de la solution.

1.2.1 Mode Hijax

Le mode Hijax ajoute du code JavaScript modifiant le comportement d'une page et remplaçant les actions utilisateurs par des requêtes Ajax et ceci sans perturbation de manière à ce que le service soit disponible indépendamment du support ou non du JavaScript au sein du navigateur.

Ceci implique que le code JavaScript des requêtes Ajax doit être conçu pour interagir avec la partie serveur et interpréter les réponses aux actions utilisateurs.

Ceci implique également que la partie serveur doit être conçue pour servir à la fois les requêtes http issues des appels classiques « web 1.0 » et des appels Ajax et retourner les résultats appropriés à chaque mode.

La collaboration entre les parties serveurs et clientes se fait par le biais de contrat d'interface et de collaboration :

- la distinction entre les deux types de requêtes est faite soit en ajoutant un paramètre (mode=XML) ou en modifiant l'url (/monaction -> /monaction-hijax),
- la partie serveur publie la syntaxe, le format et les schémas des requêtes et réponses métiers,
- la partie cliente javascript interprète les réponses métiers du serveur et sait également traiter les cas particuliers issus des modules techniques côté serveurs tels que les demandes d'authentification ou les redirections http.

L'exemple suivant illustre la différence de fonctionnement entre les deux modes dans le cas d'une liste affichée sous la forme d'un tableau paginable :

- Cinématique sans Hijax :
  1. L'utilisateur demande la première page de la liste,
  2. L'application retourne un document HTML complet (avec entête, menu, corps, bas de page) avec un tableau des 25 premiers éléments et des liens hypertextes pour la pagination,
  3. L'utilisateur demande à accéder à la page suivante en cliquant sur le lien hypertexte correspondant,
     a. Le navigateur émet une requête http pour afficher la page correspondante,
  4. L'application retourne un document HTML complet (avec entête, menu, corps, bas de page) avec un tableau des 25 éléments suivants et des liens hypertextes pour la pagination.

- Cinématique avec Hijax :
  1. L'utilisateur demande la première page de la liste,
  2. L'application retourne un document HTML complet (entête, menu, corps, bas de page) avec un tableau des 25 premiers éléments et des liens hypertextes pour la pagination,
     a. *Le mode Hijax s'initialise dans le navigateur et les liens hypertextes pour la pagination sont court-circuités par le code javascript contenu dans la page et prévu pour l'interactivité et les échanges Ajax,*
  3. L'utilisateur demande à accéder à la page suivante en cliquant sur le lien hypertexte correspondant,
     a. *Le clic ayant été détourné par le mode Hijax, le code javascript émet une requête Ajax à la place de l'appel direct du navigateur en ajoutant un code permettant de reconnaitre la requête Ajax*
  4. L'application reconnait la requête Ajax et retourne un document partiel (sans entête, menu, corps, bas de page) contenant les 25 éléments suivants
     a. *Le code javascript réceptionne la réponse à la requête Ajax et met à jour le tableau.*

1.2.2 Mode Single Page Application (SPA)

Le mode SPA étend le mode Hijax à toute une application (ou une partie) ; seul le corps principal est rechargé lors d'un changement de fonction, les parties comme l'entête ou le menu restent inchangées.

En plus des modes « web 1.0 » et Hijax, la partie serveur doit également être conçue pour renvoyer des réponses HTML constituées uniquement du corps principal.

Les contrats d'interface et de collaboration sont ainsi étendus entre les différentes parties :

- Il doit être possible de distinguer les trois types de requêtes : html complet, html partiel, xml Ajax,
- les corps de page sont contraints de manière à pouvoir s'exécuter l'un après l'autre sans effet de bord,
- les identifiants DOM doivent être uniques dans l'ensemble des pages concernés par le mode SPA.

L'exemple suivant illustre le fonctionnement SPA par rapport au mode « web 1.0 » dans le cas d'une liste des pays et d'une liste des organisations :

- Cinématique sans Hijax et sans SPA :
  1. L'utilisateur demande la première page de la liste des pays,
  2. L'application retourne un document HTML complet (avec entête, menu, corps, bas de page) avec un tableau des 25 premiers pays et des liens hypertextes pour la pagination,
  3. L'utilisateur demande à accéder à la page suivante en cliquant sur le lien hypertexte correspondant,
     a. Le navigateur émet une requête http pour afficher la page correspondante,
  4. L'application retourne un document HTML complet (avec entête, menu, corps, bas de page) avec un tableau des 25 pays suivants et des liens hypertextes pour la pagination,
  5. L'utilisateur demande à accéder à la liste des organisations en cliquant sur le lien hypertexte correspondant dans le menu,
  6. Le navigateur émet la requête http correspondante,
  7. L'application retourne un document HTML complet (avec entête, menu, corps, bas de page) contenant la liste des organisations.

- Cinématique avec Hijax et SPA :
  1. L'utilisateur demande la première page de la liste des pays,
  2. L'application retourne un document HTML complet (entête, menu, corps, bas de page) avec un tableau des 25 premiers pays et des liens hypertextes pour la pagination,
     a. *Le mode Hijax s'initialise dans le navigateur et les liens hypertextes pour la pagination sont court-circuités par le code javascript contenu dans la page et prévu pour l'interactivité et les échanges Ajax,*
     b. *Les liens concernés par le mode SPA, dont l'accès à la liste des organisations, sont court-circuités et remplacé par du code javascript,*
  3. L'utilisateur demande à accéder à la page suivante en cliquant sur le lien hypertexte correspondant,
     a. *Le clic ayant été détourné par le mode Hijax, le code javascript émet une requête Ajax à la place de l'appel direct du navigateur en ajoutant un code permettant de reconnaitre la requête Ajax*
  4. L'application reconnait la requête Ajax et retourne un document partiel (sans entête, menu, corps, bas de page) contenant les 25 pays suivants
     a. *Le code javascript réceptionne la réponse à la requête Ajax et met à jour le tableau.*
  5. L'utilisateur demande à accéder à la liste des organisations en cliquant sur le lien hypertexte correspondant dans le menu,
     a. *Le clic ayant été détourné par le mode Hijax, le code javascript émet une requête Ajax à la place de l'appel direct du navigateur en ajoutant un code permettant de reconnaitre la requête SPA*
  6. L'application retourne un document partiel correspondant au document HTML de la liste des organisations sans l'entête, menu et bas de page.
     a. *Le code javascript remplace le corps de page de la liste des pays par celui de la liste des organisations.*

Chaque fragment HTML correspondant au corps de page est conçu pour être indépendant de son conteneur qui peut-être :

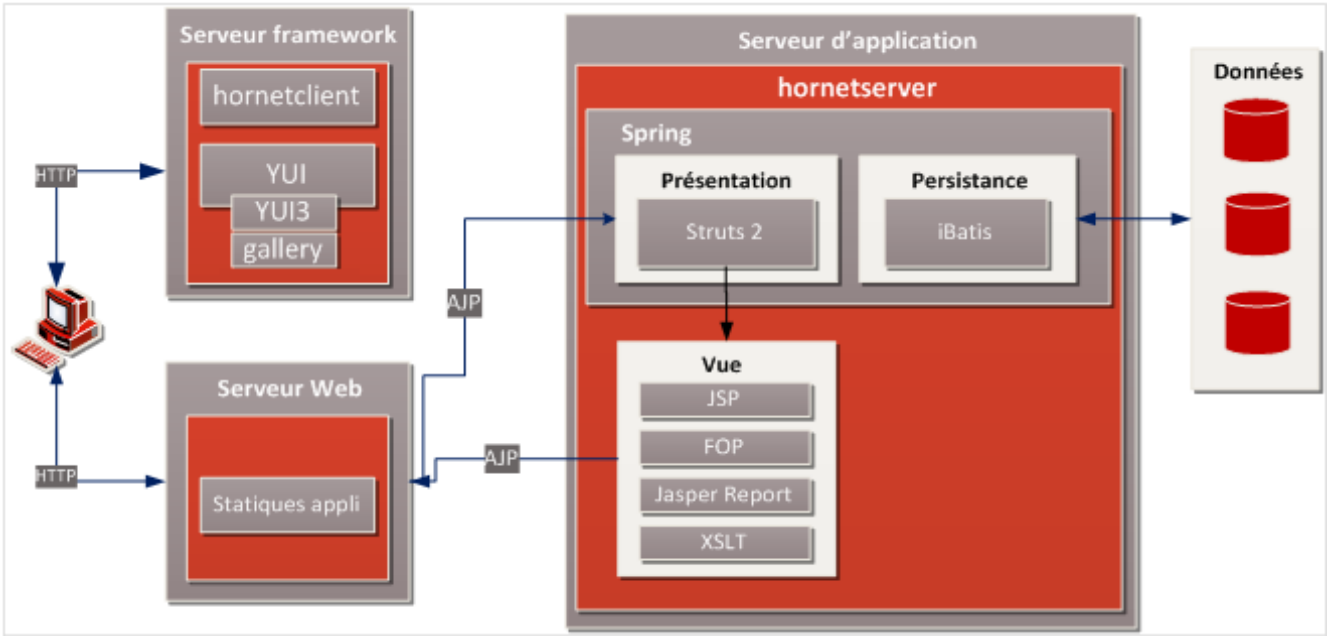
- soit la page complète obtenue lors d'un premier accès à la page liée au fragment HTML (ex. la liste des pays dans la page correspondante),
- soit la page complète obtenue lors d'un premier accès à une autre la page (ex. la liste des organisations accédée depuis la page de la liste des pays),
- soit un autre fragment HTML contenu lui-même dans un autre conteneur (ex. onglets)

Le concept de fragment HTML réutilisable dans différents conteneurs est appelé « pagelet » et est similaire aux concepts de portlets dans les spécifications jee portail sauf que les pagelets sont utilisables dans toutes pages html.

Le composant issu de la communauté yui « gallery-dispatcher » est utilisé pour faciliter le développement SPA, il permet notamment d'exécuter les scripts contenus dans les fragments de document échangés en Ajax.

2 Architecture Hornet

2.1 Architecture générale



2.2 Architecture HTML et navigateur

L'architecture retenue pour le fonctionnement dans le navigateur est basée sur un modèle en couche :

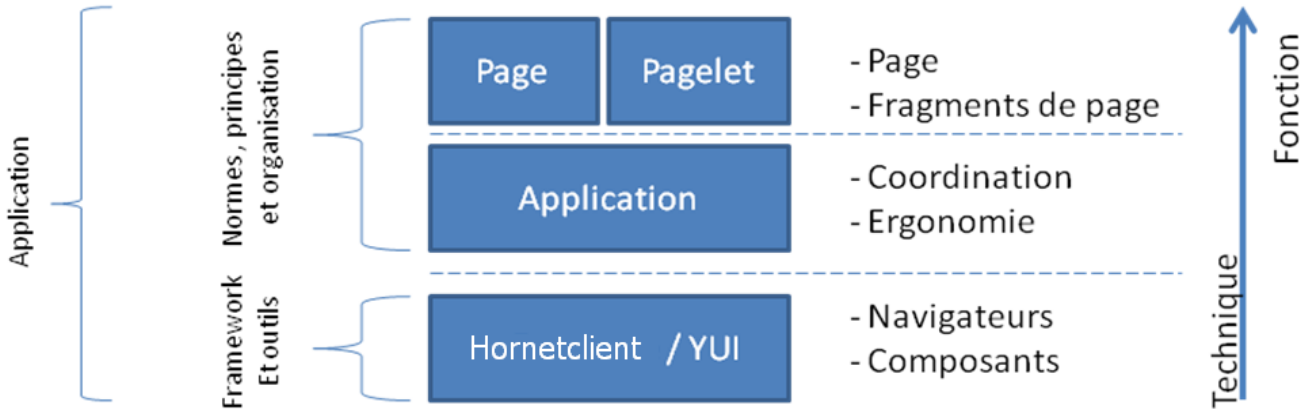
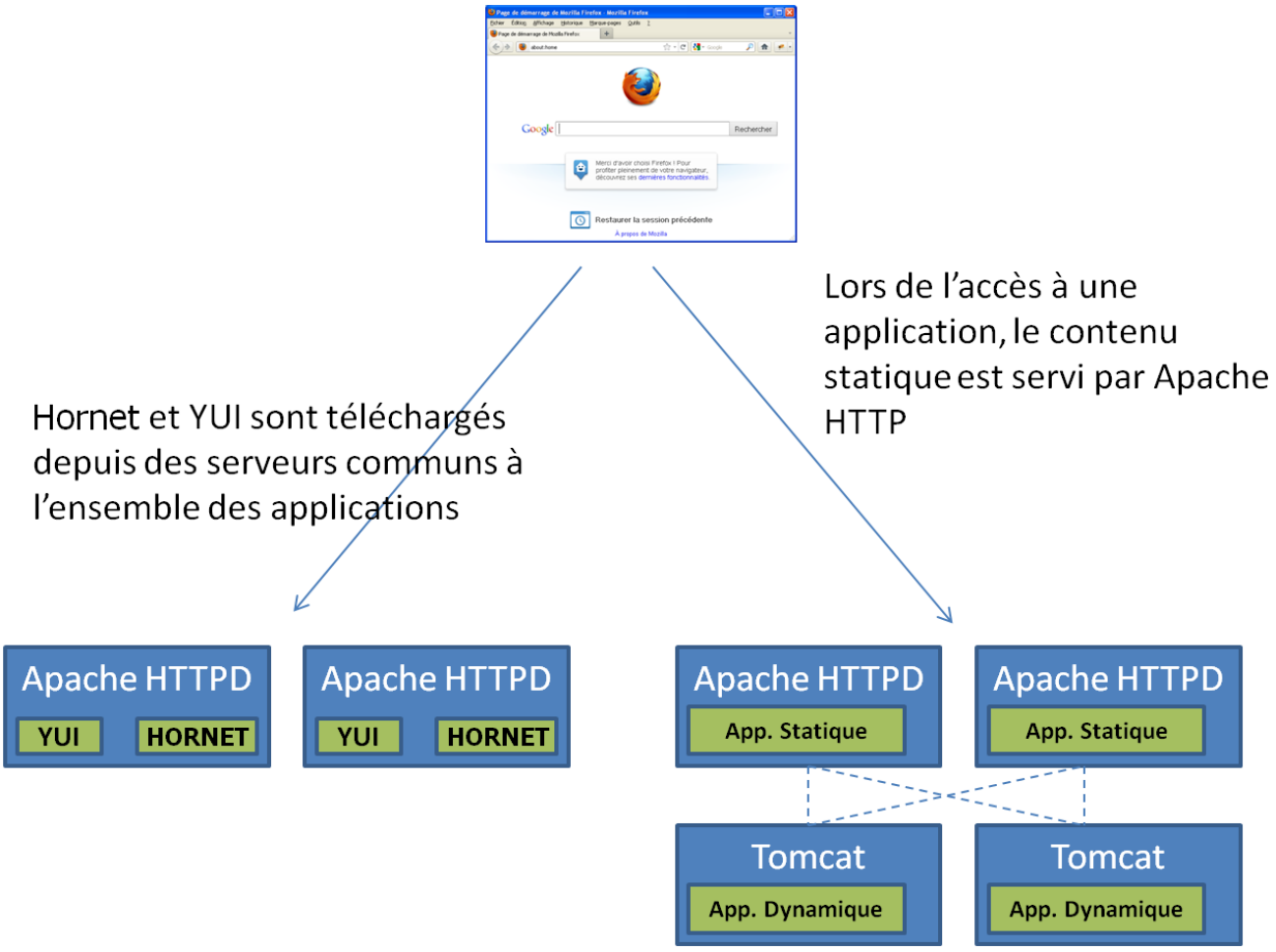


Figure 1 : Architecture en couche côté client

Les trois niveaux retenus sont :

- La couche Hornetclient et YUI permet de s'abstraire du navigateur et propose de nombreux utilitaires et composants pour faciliter le développement d'une application : Manipulation du DOM, Ajax …,
- La couche application définit l'ergonomie standard, et coordonne les interactions entre composants et pages,
- La couche page/pagelet compose les composants Hornet afin d'offrir à l'utilisateur les fonctionnalités demandées.

2.3 Distribution des composants mutualisés

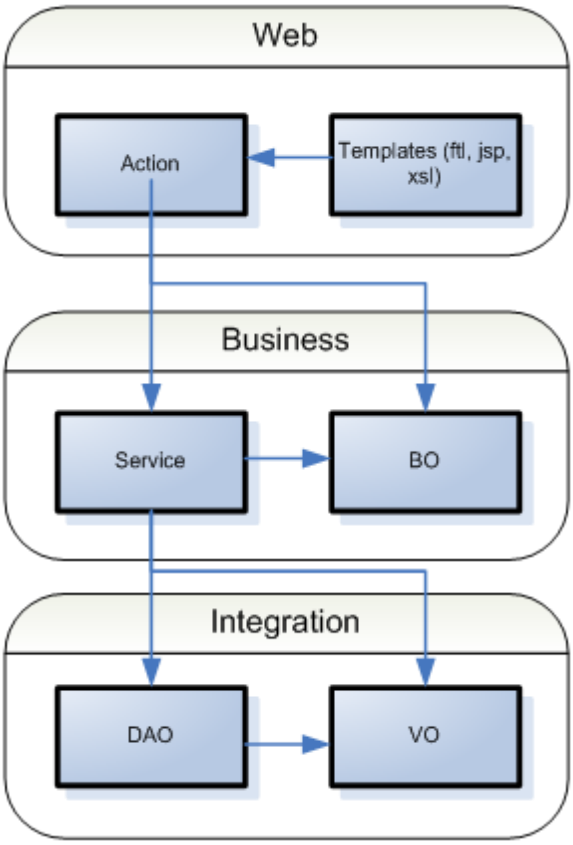


2.4 Architecture applicative

La partie serveur est implémentée pour la plateforme Java 6 et Java Enterprise Edition 1.6.

2.4.1 Découpage

La partie serveur utilise une architecture en trois couches applicatives (3-Tiers) :



Chaque couche utilise des stéréotypes de classes qui lui sont propres.
Les contraintes structurant cette architecture sont :

1. Limiter la dépendance du code applicatif envers le socle technique, afin de réutiliser le même code dans des contextes d'exécutions différents (tests unitaires, mode bouchonné, …).
2. Avoir un couplage faible entre les composants applicatifs.
3. Permettre la réutilisation des services métiers dans d'autres applications.

2.4.1.1 Couche Web

La couche web définit la partie Vue/Contrôleur du pattern MVC (Model-View-Controller). Elle s'organise autour des :

- Actions : Elles jouent le rôle de Contrôleur
- Templates : Ils sont en charges de fournir la Vue à la couche présentation

Pour plus d'informations consulter la section 3.3

2.4.1.2 Couche Business

La couche Business caractérise la logique métier de l'application. Elle se décompose en :

- Service : Les services se chargent d'effectuer les traitements sur les modèles.
- BO : Ils représentent le modèle métier de l'application.

Pour plus d'informations consulter la section 3.5

2.4.1.3 Couche Intégration

Cette couche représente l'accès aux données et s'inscrit dans les principes du pattern DAO (Data Access Object). Elle contient les :

- DAO : Ces objets ont pour seul rôle d'accéder aux données
- VO : Ils constituent les objets de transfert entre le modèle métier et le modèle persistant. Leurs seuls objectifs est d'accueillir les données.

Pour plus d'informations consulter la section 3.6

2.4.2 Framework

Les frameworks open-source structurant cette architecture sont :

- **Struts2** : web-tiers (mapping de requêtes, validation serveur, i18n, redirection result)
- **Spring** : business-tiers : centralisation de la configuration dans un fichier XML, fabrique d'objets et résolutions des dépendances, intégration de différents Framework (dont Struts2 et MyBatis)
- **Spring Security** : gestion du modèle de sécurité applicative
- **MyBatis** : mapping Objet/Relationnel

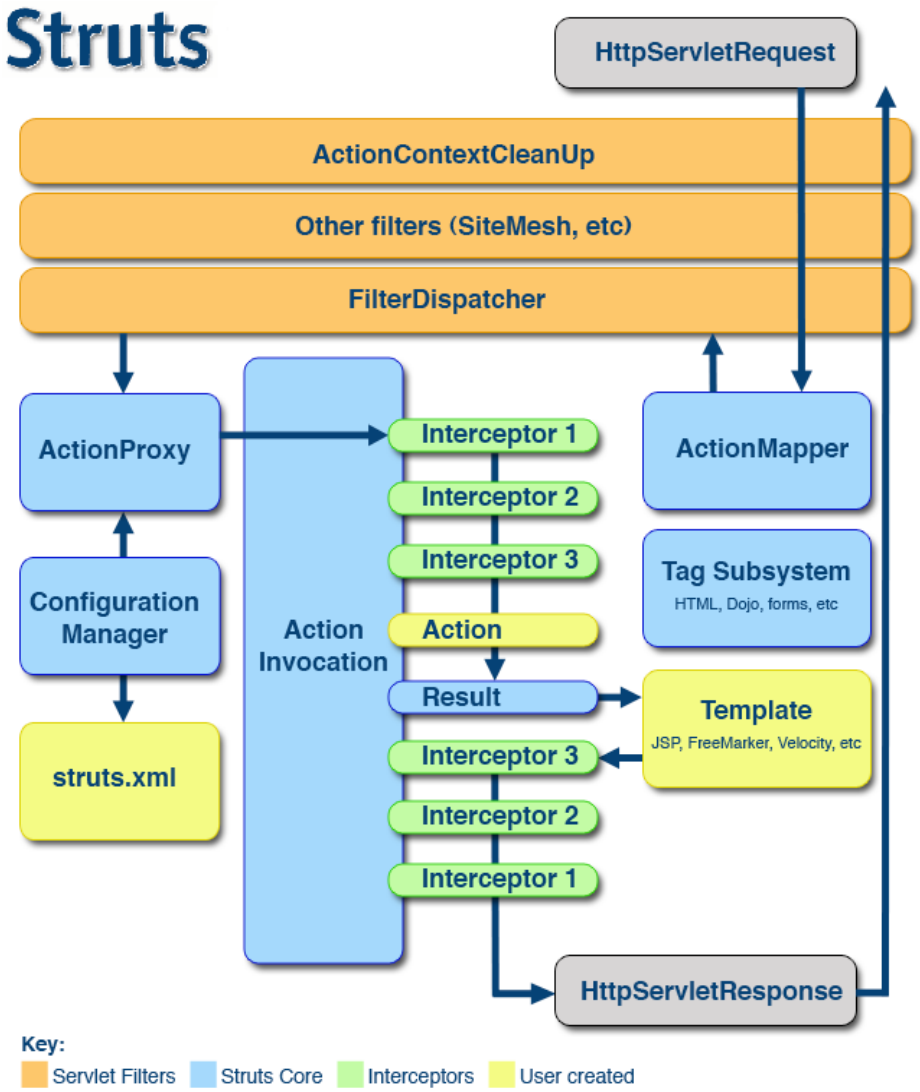
2.5 Architecture de Struts 2

La partie Web est structurée autour du framework Struts 2 car il permet de répondre aux deux premières contraintes de l'architecture.

Struts 2 utilise les patterns Model-View-Controller (MVC) et Inversion Of Control (IOC).

- MVC permet d'améliorer la structuration et la séparation des responsabilités au sein de l'application Web. Le pattern MVC utilisé ici dispose d'un seul contrôleur qui intercepte toutes les actions utilisateur (c'est le MVC de type 2). Le contrôleur est paramétrable (via un ou plusieurs fichiers xml) et retransmet les requêtes à des classes dédiée (classes Action). Ces classes permettent d'agir sur le Modèle et informe le contrôleur sur l'état d'exécution. Le contrôleur donne ensuite la main à la vue qui permet de générer la réponse à renvoyer.

- IOC permet de supprimer les dépendances du code applicatif envers l'API Servlet, dont les spécifications sont actuellement en version 2.5, ce qui permet l'exécution des mêmes classes en dehors du conteneur Web (pour les tests unitaires par exemple)



Etapes de traitement d'une requête http (http://localhost:8080/test/toto.html) :

1. La requête arrive sur le conteneur de Servlet disposant du connecteur http sur le port 8080. Le conteneur redirige la requête vers le contexte « test » où l'application est déployée, et crée un objet Request et Response.
2. Ces objets traversent une chaine de filtres, notamment le FilterDispatcher de Struts.
3. Le filterDispatcher demande à l'ActionMapper si la requête doit être traitée par une action Struts (règle basée sur la forme de l'url, par défaut : *.xml, *.pdf, *.xls, *.doc)
4. Le FilterDispatcher demande à l'ActionProxy de retrouver l'action associée à l'URI « toto », via le fichier de configuration struts.xml
5. Une pile d'intercepteur (ici 1 à 3) est exécutée afin de fournir différents services communs aux actions.
6. L'action est exécutée.
7. Le contrôle est donné à un objet Result chargé de produire le flux de réponse HTTP. L'objet s'appuie sur un langage de Template.
8. Les intercepteurs sont dépilés.
9. Les filtres sont dépilés.
10. La requête revient au conteneur et ce dernier ferme le flux de réponse.

Note : Les intercepteurs permettent de factoriser des traitements communs autour des actions.
Ils peuvent s'exécuter :

- avant une action, par exemple pour initialiser des données sur certains types d'actions (implémentant les interfaces telles que PrincipalAware, ScopedModelDriven,…).
- Après une action et l'appel du Result
- Après une action mais avant l'appel du Result (PreResultListener)

Dans une application basée sur Struts, seules les classes Action (partie controller), les Templates (partie View) et le fichier de configuration seront à développer (cf. schéma d'architecture).

2.5.1 Controller

La partie « Controller » de l'architecture est implémentée par des classes « Action ».

Une classe Action permet de traiter une requête http, d'agir sur le Model et de déléguer la construction de la réponse à la Vue.

Une classe Action est un javabean possédant au moins une méthode de traitement. Par défaut Struts s'attend à trouver une méthode execute(), mais il est possible de spécifier une autre méthode dans la configuration du mapping. Une action peut donc déclarer plusieurs méthodes de traitement, afin de centraliser les opérations de type CRUD.

Les classes Action peuvent étendre la classe `com.opensymphony.xwork2.ActionSupport` afin de gérer la validation (Cf. exemple).

2.5.2 Model

La partie « Model » correspond aux entités métier de l'application (les BO). Il sera accédé via les classes de Service. La ou les classes de service seront mises à disposition de la classe Action par Spring via le constructeur (Cf. exemple).

2.5.3 View

La partie « View » permet de créer un flux de réponse http en se basant sur le modèle des entités métier (BO), initialisés lors du traitement de l'action.

La vue invoquée par le contrôleur suite au traitement de l'action est paramétrée dans struts.xml (Cf paramétrage)

➲ **Afin de gagner en performance et en productivité, il est recommandé d'utiliser des templates JSP et non des templates XSLT pour la génération des flux HTML, XML, XLS, DOC et CSV.**

3 Développement

3.1 Préparation de l'environnement de développement

Cette section fait l'objet d'un document à part entière. Se référer au « Guide de Paramétrage ».

3.2 Partie Controller

3.2.1 Implémentation d'une classe action

- Dériver l'action de la classe ActionSupport, qui permet entre autres de bénéficier de la validation déclarative.
- Créer une propriété pour chaque paramètre de requête, en utilisant le type java compatible.
- Si une action utilise des services métiers, alors elle doit définir un constructeur ayant comme paramètres ces services. Ces derniers seront fournis par Spring à l'exécution.
- **Attention : pas de getter et setter sur les services.**
- Créer éventuellement des méthodes d'exécution distinctes s'il s'agit d'une action de type CRUD sur un formulaire ou un tableau.
- **Attention : les méthodes d'exécution ne doivent pas commencer par get, set ou is sinon elles seront sérialisées en XML, dans le cas d'une transformation XSLT.**
- Créer des getter/setter sur les propriétés devant être accédés par les templates et les propriétés mappant les paramètres de requête.

Exemple :

```
/**
 * Gestion de la liste de produits
 *
 * @author EffiTIC
 */
public class GererProduitsAction extends ActionSupport {

    /**
     * Comment for <code>serialVersionUID</code>
     */
    private static final long serialVersionUID = 5563129040740608297L;

    /** La produit de présentation */
    protected Product product;

    /** La liste de présentation des produits */
    private List<Product> list;

    /** Le service métier des Produits */
    Private transient ProductService productService;

    /*
     * Constructeur par défaut
     */
    public GererProduitsAction(ProductService productService) {
        this.productService = productService;
        this.list = new ArrayList<Product>();
    }

    /*
     * Fournie la liste des produits
     * @return la liste des produits
     */
    public String list() {
        this.list.addAll(productService.list());
        return ActionSupport.SUCCESS;
    }

    /*
```

```
     * Ajoute un produit
     * @return SUCCESS
     */
    public String add() {
        this.productService.add(product);
        return ActionSupport.SUCCESS;
    }

    /*
     * Modifie un produit
     * @return SUCCESS
     */
    public String update() {
        this.productService.update(product);
        return ActionSupport.SUCCESS;
    }

    /*
     * Supprime un produit
     * @return SUCCESS
     */
    public String remove() {
        this.productService.remove(product);
        return ActionSupport.SUCCESS;
    }

    /*
     * Accesseur du produit
     * @return le produit
     */
    public Product getProduct() {
        return this.product;
    }

    /*
     * Modifieur du produit
     * @param product le produit à modifier
     */
    public void setProduct(
            Product product) {
        this.product = product;
    }

    /*
     * Accesseur de la liste de produits
     * @return la liste de produit
     */
    public List<Product> getList() {
        return this.list;
    }

    /*
     * Modifieur de la liste de produits
     * @param list la liste de produit à modifier
     */
    public void setList(List<Product> list) {
        this.list = list;
    }
}
```

Attention, la récupération des paramètres (post ou get) passés par Struts à l'action n'est rendue possible que si le nom du paramètre, déterminé par le client, correspond au nom de l'attribut de la classe action. Il est cependant possible d'utiliser des appellations différentes.

Pour cela, il faut utiliser les Alias de Struts, à configurer dans le fichier « struts.xml » au moment de la déclaration de chaque action :

```
<package name="commandes.gestionFournisseurs" namespace="/dyn/protected/package"
extends="struts-hornet-hornetserver">
<action name="nomDeLAction"  class="hornet.projet.web.action.nomAction" method="nomMethode">
       <param name="aliases">#{
           'id'                : 'idFournisseur',
           'code'              : 'fournisseurBO.code',
           'libelle'           : 'fournisseurBO.libelle',
           'cadeaux'           : 'fournisseurBO.fournisseurDeCadeaux',
```

```
            'siren'             : 'fournisseurBO.siren',
            'adresse'           : 'fournisseurBO.adresse',
            'complementAdresse' : 'fournisseurBO.adresseComplementaire',
            'cp'                : 'fournisseurBO.codePostal',
            'ville'             : 'fournisseurBO.villeBO.libelle',
            'courriel'          : 'fournisseurBO.courriel'
        }
        </param>
        <result name="success" type="xslt">
             <param name="location">
         /templates/package/template.xsl
        </param>
        </result>
</action>
```

L'énumération des alias doit avoir la forme suivante :

'nom du paramètre passé par le client en post ou get' : 'Attribut dans l'action'.

On peut ainsi désigner dans l'action tout objet possédant les getters et setters correspondant. Il faut toutefois s'assurer que chaque objet a été préalablement instancié (dans le constructeur de la classe action, par exemple).

3.2.2 Configuration du controller

Un paramétrage est nécessaire à Struts pour mettre en correspondance les éléments de l'architecture MVC.

Ce paramétrage est défini dans le fichier **struts.xml**. Ce fichier doit être présent dans le *classpath* de l'application. Ici il est placé à la racine du répertoire des sources (*src*)

Il sera possible de découper ce paramétrage en plusieurs fichiers si l'application devient importante. Dans ce cas, struts.xml deviendra le fichier maitre et le nommage des fichiers fils se fera avec la convention struts-mondomaine.xml.

3.2.2.1 Principe

Le paramétrage consiste principalement à mapper des URL sur des actions java et pour chaque action, de définir les vues selon le type de résultat produit par l'action.

Le paramétrage est structuré : chaque action appartient à un package qui représente une partie de l'URL. De plus, chaque action possède un alias qui représente une autre partie de l'url.

Ainsi l'url suivante :

```
http:// domaine :port/contexte/dyn/protected/domaine/sousdomaine/monAliasAction.html
```

Sera mappée par :

```
<package name="domaine" namespace="/dyn/protected/domaine/sousdomaine" extends="hornet-default">
<action name="monAliasAction">
         <result name="success">
                WEB-INF/xml-jsp/domaine/monAction.jsp
         </result>
        </action>
</package>
```

En cas de succès, le traitement de la requête est transmis à un result, ici de type JSP pour construire la réponse sous forme d'une page HTML.

En cas d'erreur de validation (« input ») ou fonctionnelle (« error »), Cf Gestion des erreurs

Il existe plusieurs types de result : JSP, Redirect, RedirectAction, etc. Voir : http://struts.apache.org/2.x/docs/result-types.html

3.2.2.2 Mapping des méthodes d'execution

Il est possible d'apporter une structuration supplémentaire en regroupant plusieurs méthodes d'exécution dans une action (par exemple le chargement et l'enregistrement d'un formulaire).

Ainsi l'url suivante :

```
http://domaine:port/contexte/dyn/protected/domaine/monAliasAction-maMethode.html
```

Sera mappée par :

```
<package name="domaine" namespace="/dyn/protected/domaine" extends="hornet-default">
<action name="monAliasAction-maMethode" method="maMethode" class="fr.hornet.web.action.MonAliasAction">
         <result name="success">
                WEB-INF/jsp/domaine/monAction.jsp
         </result>
        </action>
</package>
```

3.2.2.3 Wildcard-mapping

Si une classe d'action regroupe beaucoup de méthodes d'exécution (type CRUD), alors pour ne pas surcharger la configuration il sera utile d'utiliser le Wildcard-Mapping :

```
<package name="domaine" namespace="/dyn/protected/domaine" extends="hornet-default">
   <action name="monAliasAction-*" class="fr.hornet.web.action.Action{1}">
  <result name="success">
                WEB-INF/jsp/domaine/{1}.jsp
  </result>
    </action>
</package>
```

***Attention** : ne pas utiliser le Dynamic Method Invocation (DMI), qui consiste à appeler la méthode d'exécution directement depuis l'url sans passer par le mapping (avec le !) :*

http://localhost:8080/test/dyn/protected/action.xml!methode

3.2.2.4 Configuration des autorisations

3.2.2.4.1 Présentation

Hornet utilisait dès ses premières versions le mécanisme de sécurité standard Java EE configuré dans le descripteur de déploiement (web.xml) et les utilisateurs étaient ceux du realm Tomcat par défaut.

Depuis la version 3.1 d'Hornet, Spring Security est intégré comme solution d'authentification et de gestion des droits au sein des applications. Spring Security a démarré fin 2003 sous le nom d' « Acegi Security » et est distribué sous licence Apache depuis 2004. Ses principales fonctionnalités couvrent à la fois l'authentification et le contrôle d'accès basé sur la notion de rôle, au niveau de l'IHM, des URL et du code Java (par annotation).

La configuration de ce composant s'effectue principalement dans un fichier xml acceptant les déclarations de bean grâce à l'injection de dépendance de Spring et offrant un namespace[1] spécifique facilitant les paramétrages les plus basiques en termes de sécurité.

Les concepts clés de mise en œuvre de la sécurité sont les suivants :

- **UserDetailService**

L'API Spring Security défini une interface UserDetailService dont les différentes implémentations servent à récupérer les informations sur un utilisateur, en particulier les couples login / mot de passe et la liste de ses rôles.

- **AuthenticationProvider**

[1] Namespace : espace de nommage

L'AuthenticationProvider utilise les informations de l'utilisateur fournies par le UserDetailService et traite les demandes d'authentification. Spring Security permet des authentifications basées sur CAS, LDAP, OpenID et X.509 entre autres.

Sécurisation HTTP

La sécurité HTTP permet principalement de filtrer les URL vis-à-vis des rôles de l'utilisateur connecté. Cela se traduit dans le fichier de configuration par l'élément XML « http ». Par exemple :

```
<sec:http auto-config='true'>
    <sec:intercept-url pattern='/secure/**' access='ROLE_A,ROLE_B'/>
</sec:http>
```

C'est aussi cette section qui permet d'activer des filtres plus complexes, comme l'authentification CAS ou la gestion de la déconnexion.

L'attribut « access » supporte par défaut une liste de rôles séparée par des virgules. Il est cependant possible d'activer un système de « Web Security Expressions » [2] spécifique à Spring Security.

En plus du filtrage par rôles, à l'aide des méthodes « *hasRole* » et « *hasAnyRole* », ces expressions permettent également un filtrage par IP (méthode « *hasIpAddress* ») qui peut s'avérer intéressant pour la sécurisation d'une interface d'administration par exemple. Les opérateurs logiques « or » et « and » sont aussi supportés.

Dans le cas où des ressources ne doivent pas faire l'objet d'une sécurisation de contenu, la directive suivante permet d'exclure tout contrôle de la part de Spring Security :

```
    <sec:http pattern="/static/images/**" security="none"/>
```

Affichage / masquage d'éléments graphiques

Spring Security contient une librairie de tag JSP dédiée permettant, entre autres, de conditionner l'affichage d'un bloc html en fonction des rôles de l'utilisateur connecté, comme illustré ci-après :

```
<sec:authorize access="hasRole('ROLE A')">
…
</sec:authorize>
```

L'attribut « access » prend en charge les Web Security Expressions décrite ci-dessus à une exception près : la méthode « hasIpAddress » n'est pas supportée.

Restriction d'exécution des méthodes de la couche service ET/OU action

Dans une application web, le masquage d'éléments de l'IHM et le filtrage d'URL ne suffisent pas toujours à assurer un niveau de sécurité suffisant. Considérons l'exemple suivant : sur une page publique, si un bouton de suppression est masqué mais que le pattern associé à l'action n'est pas filtré, il est possible d'appeler directement l'URL exécutant la suppression.

Pour traiter de tels cas, et surtout renforcer la sécurité au-delà de la couche web, Spring Security permet de restreindre l'accès directement au niveau des méthodes Java en utilisant une série d'annotations spécifiques.

Par défaut, les annotations sont désactivées. Il suffit d'ajouter le tag « global-security-method » au fichier de configuration pour pouvoir s'en servir :

```
<sec:global-method-security proxy-target-class="true" secured-annotations="enabled" pre-post-
annotations="enabled"/>
```

[2] Liste disponible à l'adresse suivante : http://static.springsource.org/spring-security/site/docs/3.0.x/reference/el-access.html#el-common-built-in

Reste alors à ajouter l'annotation « **@*Secured*** » suivie de la liste des rôles autorisés pour gérer les droits d'exécution d'une méthode ou d'une classe complète.[3]

En reprenant l'exemple présenté ci-dessus, dans l'interface du service nécessitant un filtrage par rôles, les méthodes d'action concernées doivent alors être marquées avec l'annotation suivante pour n'autoriser que les utilisateurs ayant le rôle suffisant à leur exécution:

```
@Secured(Role.A)
```

Pour autoriser plusieurs rôles, on utilisera la syntaxe (l'opérateur utilisé entre les rôles est le « OU ») :

```
@Secured({Role.A, Role.B})
```

3.2.2.4.2 Mise en œuvre technique de la sécurité

3.2.2.4.2.1 Création du fichier de configuration

C'est le fichier **spring-appContext-security.xml** qui centralise toute la configuration de sécurité de l'application.

Afin que ce nouveau fichier soit pris en compte, on l'importe dans le fichier spring-appContext-web.xml :

```
<import resource="classpath:/spring-appContext-security.xml"/>
```

Le fichier spring-appContext-web.xml est déclarédans la section « context-param » du fichier **web.xml**, comme illustré ci-après :

```
  <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
              classpath*:spring-appContext-web.xml.

        </param-value>
  </context-param>
```

Les éléments « *security-constraint* », « *login-config* » et « *security-role* » relatif au paramétrage standard Java EE peuvent alors être supprimés de même que la déclaration du « *realm* » Tomcat dans le fichier **context.xml**.

Le fichier **spring-appContext-security.xml** précise alors les éléments de validation par les espaces de nommage suivants :

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:sec="http://www.springframework.org/schema/security"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-security-3.1.xsd">

…

</beans>
```

3.2.2.4.2.2 Gestion d'une authentification « simple » avec déclaration des utilisateurs par configuration

Les mécanismes de base permettent de définir un registre de sécurité en mémoire, afin d'y définir des éléments d'authentification et d'autorisations. Un exemple de configuration est présenté ci-après :

```
<sec:user-service id="userService">
    <sec:user name="admin" password="admin" authorities="ROLE_Appli_USER,ROLE_Appli_ADMIN" />
    <sec:user name="user" password="user" authorities="ROLE_Appli_USER" />
</sec:user-service>
```

[3] La gestion de la JSR-250 est également supportée par Spring security : http://static.springsource.org/spring-security/site/docs/3.0.x/reference/springsecurity-single.html#ns-global-method

Ce registre d'utilisateurs est stocké en mémoire, contenant les définitions de deux identités et de deux rôles d'autorisations :

- Un utilisateur standard « ROLE_Appli_USER »
- Un administrateur avec des droits plus élevés « ROLE_Appli_ADMIN »

Quelle que soit la solution de stockage des utilisateurs retenue, il est possible d'encoder les mots de passe à l'aide d'un algorithme de hashage. L'ajout d'un salt pour renforcer la résistance de l'encodage est également possible. Il suffit pour cela d'implémenter l'interface « *SaltSource* » avec une classe calculant le salt à partir des informations de l'utilisateur.

La liste des utilisateurs peut également être stockée dans un fichier properties sous le format suivant (les éléments entre crochets sont optionnels) :

```
nomutilisateur=motdepasse,rôle[,rôle] [,enabled4|disabled5]
```

3.2.2.4.2.3 Gestion d'une authentification avec déclaration des utilisateurs en base de données

La configuration d'une base de données relationnelle comme fournisseur de données d'authentification consiste simplement à paramétrer un « *UserService* » utilisant une connexion JDBC de la manière suivante :

```
<sec:jdbc-user-service  data-source-ref="dataSource"
     authorities-by-username-query="select LOGIN, ROLE from ROLE where LOGIN = ?"
     users-by-username-query="select LOGIN, PASSWORD, ENABLED from UTILISATEUR where LOGIN = ?"
     role-prefix="ROLE_"/>
```

On retrouve la référence à une dataSource JDBC et deux requêtes SQL :

- Une pour récupérer les couples login/mot de passe : « users-by-username-query »
- L'autre pour récupérer les rôles d'un utilisateur à partir de son login « authorities-by-username-query »

Il est important de noter que les rôles stockés en base de données ne comportent pas le préfixe des rôles, spécifiable par le paramètre « role-prefix ».

Remarque : les objets de type AuthenticationManager peuvent parfaitement être chaînés. Dans ce cas, les informations de connexions sont testées successivement jusqu'à ce que l'un d'eux les valide. Si l'on arrive en fin de chaîne sans qu'aucune identité n'ait pût être vérifiée, alors la connexion est refusée.

3.2.2.4.2.4 Gestion d'une Authentification CAS

La configuration de CAS dans Spring Security s'effectue en déclarant :

- Les URL nécessaires à la mise en place de CAS, via le contenu du fichier **cas.properties**.

```
cas.loginUrl=https://serveur-cas/login
cas.logoutUrl=https://serveur-cas/logout
cas.securityCheckUrl=http://serveur-applicatif/applicationWeb/j_spring_cas_security_check
cas.ticketValidationUrl=https://serveur-cas/
```

Remarque : L'URL de valitation du ticket de service est ici simplifié par l'utilisation de Spring Security puisque le suffixe est généré automatiquement en fonction de l'implémentation du service de validation paramétré.

Ce fichier est chargé par l'intermédiaire d'un gestionnaire de propriétés spécifique.

```
    <!-- Properties CAS -->
    <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>classpath:cas.properties</value>
            </list>
        </property>
        <property name="placeholderPrefix" value="$securityCas{" />
```

[4] Enabled : compte utilisateur activé
[5] Disabled : compte utilisateur désactivé

```
    </bean>
```

- Les informations de l'application devant être protégée par CAS

```
    <bean id="serviceProperties" class="org.springframework.security.cas.ServiceProperties">
        <property name="service" value="$securityCas{cas.securityCheckUrl}"/>
        <property name="sendRenew" value="false"/>
    </bean>
```

- La classe implémentant l'interface « UserDetailService » fournissant les droits de l'utilisateur identifié par CAS. Une utilisation d'un service en mémoire est considéré dans l'exemple suivant (remarque: il n'y a pas d'utilisation de l'information de mot de passe à la différence de la configuration décrite en 3.2.2.4.2.2)

```
<sec:user-service id="casUserDetailService">
    <sec:user name="admin" authorities="ROLE_Appli_USER,ROLE_Appli_ADMIN" />
    <sec:user name="user" authorities="ROLE_Appli_USER" />
</sec:user-service>
```

- L'ensemble des beans nécessaires à l'instanciation de la classe org.springframework.security.cas.authentication.CasAuthenticationProvider. A savoir, un UserDetailService, une URL de validation du ticket de service, et une URL pour la validation du jeton de sécurité :

```
<bean id="casAuthenticationProvider"
 class="org.springframework.security.cas.authentication.CasAuthenticationProvider">
        <property name="userDetailsService" ref="casUserDetailService" />
        <property name="serviceProperties" ref="serviceProperties" />
        <property name="ticketValidator">
               <bean class="org.jasig.cas.client.validation.Cas20ServiceTicketValidator">
                      <constructor-arg index="0" value="$securityCas{cas.ticketValidationUrl}" />
               </bean>
        </property>
        <property name="key" value="hornettemplate"/>
 </bean>
```

La propriété « key » permet de référencer cet AuthentificationProvider lors de l'utilisation de la fonctionnalité « remember-me », encore applée login persistant.

Remarque : La version actuelle du client CAS utilisée sur les applications Acube est celle développée à l'origine par l'université de Yale (avant 2004) or celle supportée par Spring Security est celle reprise par le consortium JASIG. La librairie Cas Client fournie par JASIG est en version 3.1 mais contient néanmoins un service de validation du ticket rétro-compatible avec la version 2.0 (Cas20ServiceTicketValidator), c'est d'ailleurs celui-ci qui est utilisé.

- Un point d'entrée pour l'authentification nécessitant :
  - l'url du formulaire d'authentification CAS (cas de la première connexion)
  - l'url de vérification d'authentification auprès du serveur (cas d'un accès à une page sécurisée après authentification CAS) :

```
    <bean id="casEntryPoint" class="org.springframework.security.cas.web.CasAuthenticationEntryPoint">
        <property name="loginUrl" value="$securityCas{cas.loginUrl}" />
        <property name="serviceProperties" ref="serviceProperties" />
    </bean>
```

- Un filtre interceptant les requêtes et un gestionnaire d'authentification associé :

```
    <sec:authentication-manager  alias="authenticationManager">
        <sec:authentication-provider ref="casAuthenticationProvider" />
    </sec:authentication-manager>

    <bean id="casAuthenticationFilter"
        class="org.springframework.security.cas.web.CasAuthenticationFilter">
        <property name="authenticationManager" ref="authenticationManager"/>
    </bean>
```

- Un filtre pour la gestion de la déconnexion :

```
    <bean id="logoutFilter" class="org.springframework.security.web.authentication.logout.LogoutFilter">
```

```
        <!-- ==== Url de redirection après déconnexion ==== -->
        <constructor-arg value="$securityCas{cas.logoutUrl}" index="0"/>
        <constructor-arg index="1">
            <list>
                <bean
            class="org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler">
                    <property name="invalidateHttpSession" value="false"/>
                </bean>
            </list>
        </constructor-arg>
        <property name="filterProcessesUrl" value="/j_spring_cas_security_logout"/>    </bean>
```

3.2.2.4.2.5 Récupération des informations utilisateur après authentification

3.2.2.4.2.5.1 Pour l'IHM

La taglib Spring Security permet d'accéder aux informations de l'utilisateur connecté. Par défaut, l'identifiant de l'utilisateur peut être affiché par le tag suivant :

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
…
<security:authentication property="principal.username" />
…
```

La propriété « principal » représente les données de connexion, i.e. l'objet « **UserDetails** », stockées dans le contexte de sécurité de Spring.

3.2.2.4.2.5.2 Dans le code Java

Pour des besoins spécifiques, l'objet « **Authentication** » est également accessible en Java depuis la méthode statique suivante :

```
SecurityContextHolder.getContext().getAuthentication()
```

3.2.2.4.2.6 Gestion du menu et du plan du site

Dans Hornet le menu est entièrement paramétré dans un fichier XML.

La définition des items du menu est récursive et l'affichage dans le menu, le plan du site ou le fil d'Ariane est conditionné par un attribut XML booléen. Chaque élément « menu-item » est associé à une liste de rôles autorisés pour y accéder, séparés par des virgules.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<pagelet>
    <root
        id="root"
        href="/dyn/protected/accueil.html">
    </root>
<menu>
  <menu-item
         id="menu.accueil"
         href="/dyn/protected/accueil.html"
         visibleDansFilAriane="false"
         rolesAutorises="ROLE Appli USER, ROLE Appli ADMIN">
  </menu-item>
  <menu-item
         id="menu.administration"
         rolesAutorises="ROLE Appli ADMIN">
  </menu-item>
</menu>
```

3.2.2.5 Rappel sur la gestion de la navigation

La gestion de la navigation entre les différentes pages HTML se fait du côté client riche : aucune gestion de la navigation ne doit être faite côté serveur applicatif (via des redirections).

Exception : la redirection vers la page de login si on n'est pas authentifié, la redirection vers la page d'index et la redirection après déconnexion. Dans ce cas, struts 2 propose un result de type redirection.

3.2.2.6 Extension du paramétrage par défaut

Le Framework étend la pile par défaut de struts (default-stack) en une pile « hornet-stack ». Cette pile défini un comportement par défaut qu'il est possible d'étendre à nouveau selon les besoins du projet.

Intercepteurs spécifique Hornetserver:

- **SecurityInterceptor** : Extension de RolesInterceptor de struts2 afin de paramétrer les rôles autorisés par action dans le fichier struts.xml (tel que sur l'exemple vue en 3.2.2.4).
- **CacheControlInterceptor** : ajout du contrôle du cache navigateur (pas de mise en cache + compatibilité https avec IE)
- **ExceptionInterceptor** : interception des exceptions (techniques et applicatives) pour transmettre un message clair pour l'utilisateur (via le result « error ») et générer un rapport d'erreur détaillé dans les logs.

Intercepteurs optionnels Hornetserver (à ajouter en plus de la pile « hornet-stack ») :

- **HijaxResultInterceptor** : ajout de la gestion du mode Hijax (modification du code de retour d'une action selon son mode)

Surchage de paramétrage :

- Redirection des erreurs vers une page par défaut affichant l'ensemble des messages.

Note : La pile « hornet-stack » sera utilisée automatiquement si le package d'action étends « hornet-default » plutôt que « struts-default »

3.2.3 Gestion de la validation

Struts propose trois niveaux de validation :

1. Validation automatique des types des paramètres de requête en fonction des propriétés de l'action.
2. Exécution de règles sur les champs de formulaire définies dans un fichier XML (validation déclarative Struts-Validator)
3. Exécution de la méthode « validate() » dans l'action .

La méthode de traitement de l'action (execute() par défaut) est exécutée une fois que tous les contrôles sont effectués.

Prérequis :
L'action doit dériver de **com.opensymphony.xwork2.ActionSupport**

3.2.3.1 Validation des types de champs

Les paramètres de requêtes sont mappés sur des propriétés java typées de l'action (String, Integer, BigDecimal, Date, …).

Si une conversion est impossible (type non conforme), un message d'erreur est ajouté à la liste ActionErrors de la classe ActionSupport. (ExceptionInterceptor)

Il est possible de redéfinir le message dans le fichier com\opensymphony\xwork2\xwork-messages_fr.properties

3.2.3.2 Validation déclarative XML

Créer un fichier *MonAction-validation.xml* dans le package hornet.projet.web.action.
Ce fichier sera valable pour toutes les méthodes de traitement (méthode execute() par défaut).
Si la validation est propre à une méthode d'exécution, par exemple update() appelée par une action appelée updateAction :

- créer un fichier *MonAction-updateAction-validation.xml* dans le package *hornet.projet.web.action*

Les validators disponibles, ainsi que la syntaxe XML sont décrits ici :
http://struts.apache.org/development/2.x/docs/validation.html

Règle : utiliser les field-validator pour les contrôles par champ, et les non-field-validator pour les contrôles interchamps.
Règle : utiliser au maximum la validation XML, notamment les expressions OGNL, pour tous les contrôles n'effectuant pas d'accès à la base de données.

Exemple de fichier de validation :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//Apache Struts//XWork Validator 1.0.2//EN"
"http://struts.apache.org/dtds/xwork-validator-1.0.2.dtd">
<validators>

    <field name="criteres.startDate">
        <field-validator type="required">
            <message key="GESTION_PARTENAIRE.recherche.startDate.format"/>
        </field-validator>
        <field-validator type="date">
            <param name="min">01/01/1753</param>
            <param name="max">31/12/9999</param>
            <message key="GESTION_PARTENAIRE.recherche.startDate.valid"/>
        </field-validator>
    </field>

</validators>
```

3.2.3.3 Validation JAVA

Surcharger la méthode validate() de la classe ActionSupport

Règle : utiliser la validation JAVA uniquement pour tous les contrôles nécessitant un accès à la couche Service.

3.2.4 Gestion de la session

3.2.4.1 Principe

La session http est une zone de mémoire serveur allouée à un utilisateur (pour un navigateur web). Elle permet :

- De conserver les données entres les différentes actions utilisateur (ex : entre les pages d'un assistant de saisie)
- De faire office de cache : on charge les données en début de cas d'utilisation seulement, afin d'optimiser les accès au système de persistance.
- Rappel : lors de la déconnexion de l'utilisateur, les données sont perdues et la mémoire utilisée est rendue disponible (sauf si on y stocke des objets du contexte serveur ou si on s'amuse à y stocker des threads (interdit !) : ces derniers ne seront pas libérés)

IMPORTANT : S'assurer de supprimer les données à la fin du cas d'utilisation, afin d'éviter les effets de bords (pouvoir rejouer le cas d'utilisation de manière identique)

3.2.4.2 SessionAware

Pour qu'une action accède à la session, il suffit d'implémenter l'interface SessionAware. On récupère une map contenant les attributs de session sous forme de (clé, valeur)

```
public class CommandeAction extends ActionSupport implements SessionAware {
  Map session;
  public void setSession(Map session) {
         this.session = session;
  }
}
```

Localisation des clés de session :

- Les clés de session propre à l'action peuvent être placées dans l'action elle-même si la session sert de partage entre plusieurs méthodes d'exécution, ou dans une classe commune à l'ensemble des cas d'utilisation.
- Il est possible d'utiliser une classe globale à l'application mais uniquement dans le cas de stockage de clés permettant par exemple, la récupération de l'utilisateur de l'application, etc.

3.2.4.3 Scope Interceptor

But : Permet de simplifier les enchainements d'actions de type Assistant de saisie.

Cf. http://struts.apache.org/2.3.x/docs/scope-interceptor.html

Exemple :

```
<action name="page1" class="hornet.projet.web.action.AssistantAction1">
  <result name="success">/jsp/assistant.jsp</result>
  <interceptor-ref name="basicStack"/>
  <interceptor-ref name="scope">
         <param name="key">CLASS</param>
                <param name="session">mySharedBO</param>
                <param name="type">start</param>
         </interceptor-ref>
</action>
<action name=" page2" class=" hornet.projet.web.action.AssistantAction2">
  <result name="success">/jsp/assistant.jsp</result>
  <interceptor-ref name="scope">
         <param name="key">CLASS</param>
                <param name="session">mySharedBO</param>
                <param name="type">end</param>
  </interceptor-ref>
  <interceptor-ref name="basicStack"/>
</action>
```

NB : on peut aussi définir les traitements dans différentes méthodes d'une même classe d'action.

3.2.4.4 Modèle du formulaire (ScopedModelDriven) :

Struts 1.x nécessitait l'utilisation d'un objet « ActionForm » pour modéliser les données du formulaire. Cet objet pouvait être stocké en requête ou en session.

Struts 2 permet maintenant de mapper les données du formulaire directement dans l'action, Mais l'ancien mécanisme peut être reproduit avec l'intercepteur `ScopedModelDrivenInterceptor`. Les actions qui souhaitent passer par un objet ActionForm intermédiaire doivent implémenter l'interface ScopedModelDriven<ClasseActionForm>, où la classe « ClasseActionForm» est un simple bean Java sérialisable. L'intercepteur possède comme attribut le scope égal à « request » (par défaut) ou « session ».

Pour les actions utilisant cet intercepteur, les paramètres de requêtes seront alors mappés par défaut sur l'ActionForm et non l'Action. Et l'ActionForm pourra être partagée automatiquement entre plusieurs actions.

Il est généralement plus simple de mapper les paramètres de requête sur un ou plusieurs objets BO référencés dans l'action, qu'on stocke ensuite en session grâce à SessionAware.

3.2.4.5 Synchronisation session / base

Entre le début et la fin d'un cas d'utilisation, les données de session peuvent être périmées (à cause des accès concurrents au système de persistance).

Deux approches selon le niveau de concurrence d'accès aux données :

- on ne met à jour que les données modifiées par l'utilisateur ou, ce qui est équivalent, on relit les données, on merge avec les données modifiées, puis on met à jour dans la même transaction
- on met en place un système de versionning (principe de CVS ou SVN)

3.2.5 Intégration dans JEE

Reprendre le fichier web.xml de l'application d'exemple pour la définition du paramétrage de Struts 2.3.

```
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
        </filter-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>struts.xml</param-value>
        </init-param>
    </filter>
```

3.2.6 Types Java spécifique

Il est possible que certains objets métier de la partie Java utilisent des types Java spécifique. Par exemple on peut imaginer une classe qui représenterait une date avec des règles de gestion particulière et une structure particulière. Il est possible dans Struts2 de définir des convertisseurs particulier pour une classe données afin de faire le mapping d'une part entre le champ et la vue mais aussi entre les paramètres de la requête et l'objet java. Pour cela, il faut créer une classe qui étend la classe StrutsTypeConverter.

Exemple :

```
public class DateIncompleteConverter extends StrutsTypeConverter {

    /** {@inheritDoc} */
    @SuppressWarnings("unchecked")
    @Override
    public Object convertFromString(
            Map arg0, String arg1[], Class arg2) {

        DateIncomplete dateIncomplete = new DateIncomplete();
        dateIncomplete.setString(arg1[0]);

        return dateIncomplete;
    }

    /** {@inheritDoc} */
    @SuppressWarnings("unchecked")
    @Override
    public String convertToString(
            Map arg0, Object arg1) {

        return arg1.toString();
    }

}
```

Cette classe contient deux méthodes :

- convertFromString : qui permet de faire la conversion des paramètres de la requête en l'objet que l'on souhaite (ici DateIncomplete)
- convertToString : qui permet de définir le format de sortie pour la vue. (Exemple : sortie du s:property dans les JSP)

Ensuite il suffit de déclarer à Struts2 que pour le type DateIncomplete, il faut utiliser le converter DateIncompleteConverter. Ceci s'effectue dans un fichier xwork-conversion.properties comme ceci :

```
# syntax: <type> = <converterClassName>
hornet.projet.utility.DateIncomplete=hornet.projet.web.action.utility.DateIncompleteConverter
```

3.3 Partie Vue

3.3.1 Généralités

3.3.1.1 Structure des pages

Les pages sont structurés en blocs indépendants séparés selon leur usage :

- entête,
- menu,
- corps,
- et bas de page.

Les interactions entre blocs sont limitées et respectent des interfaces et contrats prévus par ces blocs, exemple :

- le menu agit sur le corps du document,
- le statut d'une action peut être affiché dans l'entête ou le bas de page.

Ces blocs sont visuellement indépendants dans le rendu d'une page, exemple :

Liens d'accessibilité et entête
Menu
Corps
Bas de page

Conformément aux critères du RGAA, le contenu de la page respecte un ordre logique de lecture au niveau HTML ; cependant les blocs sont entremêlés dans le document html, étant donné les contraintes propres à la syntaxe HTML, exemple :

```
<html>
<head> …
  …
  <liens CSS pour l'entete/>
  <liens CSS pour le menu/>
  <liens CSS pour le corps/>
  <liens CSS pour le bas de page/>
  …
  <liens javascript pour l'entete/>
  <liens javascript pour le menu/>
  <liens javascript pour le corps/>
  <liens javascript pour le bas de page/>
  …
</head>
<body>
…
Html de l'entête
Balise script hijax de l'entête
Html du menu
Balise script hijax du menu
Html du corps
Balise script hijax du corps
Html du bas de page
Balise script hijax du bas de page
…
</body>
</html>
```

3.3.1.2 Structure pour le JavaScript

La nécessité d'un modèle pour le javascript s'impose étant donné l'usage intensif en Hijax et SPA à la fois pour le développement et maintenance (uniformisation) et pour la fiabilité (portabilité, absence d'effet de bord).

Le modèle utilisé pour la structuration du code javascript est basé sur le « Javascript Module Pattern » popularisé par Douglas Crockford.

Ce pattern définit l'équivalent des portées privée/protected/public des langages objets et des modèles d'héritage et de composition pour les classes (cf. http://www.adequatelygood.com/2010/3/JavaScript-Module-Pattern-In-Depth).

Par exemple, le code suivant illustre une portée privée où une variable est accessible uniquement au sein de la classe la définissant (extrait de http://www.unleashed-technologies.com/blog/2010/12/09/introduction-javascript-module-design-pattern) :

```
var myNamespaceName = (function(){

    var myPrivateVariable = 0;

    var myPrivateMethod = function(text){
        alert(text);
    }
    return {

        myPublicVariable: "foo",

        myPublicFunction: function(bar){
            myPrivateVariable++;
            myPrivateMethod(bar);
        }

    }

})();
```

3.3.1.3 Modèle pour les interactions entre composants

Le modèle pour les interactions entre les différents composants javascript constituants une page est basée sur une isolation « bac à sable » et une approche événementielle.

L'isolation « bac à sable » est obtenu par l'application du « Javascript Module Pattern » :

Un conteneur (tableau, onglet, div …) crée d'autres composants sous forme de variables privées et relie les composants entre eux via leurs méthodes publiques et leurs événements.

Par exemple, avec un composant gauge définissant deux méthodes public « incremente » / « decremente », et un évènement « onClick », le pseudo code suivant permettant de créer deux gauges façon vase communicant est :

```
var gaugeGauche = new Gauge(50) ;
var gaugeDroite = new Gauge(50) ;
gaugeGauche.on("click").subscribe(new function(evt)
    {
        gaugeGauche.incremente();
        gaugeDroite.decremente();
    });

gaugeDroite.on(« click »).subscribe(new function(evt)
    {
        gaugeDroite.incremente();
        gaugeGauche.decremente();
    });
```

Yui offre un cadre de travail autour avec la classe YUI Base facilitant la création d'objet javascript avec getters/setters et événements.

3.3.1.4 Chargement JavaScript différé et événementiel

Le passage du mode « html normal » au mode Hijax se fait lors de l'exécution des sections JavaScript qui contiennent la logique pour ajouter l'interactivité et intercepter les appels « html ».

Bien que le rendu d'une page se fasse en trois étapes logiques (téléchargement html, puis des ressources liées comme les css, images et JavaScript, et exécution des codes JavaScript), le rendu d'une page HTML n'est pas linéaire, et la plupart des navigateurs modernes affichent des portions de pages dès qu'ils le peuvent et de manière incrémentale.

En conséquence :

- le code JavaScript doit impérativement fonctionner sans reposer sur l'ordre des déclarations dans le document HTML,
- les différents codes et logiques d'initialisation JavaScript doivent être structurés pour s'initialiser

Il existe un certain nombre d'événements standardisés par le w3c (chargement, focus, click …) et d'autres définis par YUI (disponibilité d'un nœud DOM …) permettant réagir lorsque les éléments nécessaires sont disponibles dans le navigateur.

Par exemple, avec une page du type :

```
<html> …
…
<liens CSS/>
<liens JavaScript/>
<body>
…
<form id="monformulaire">
…
</form>
…
<script>
// Accroche YUI
YUI().use('node',
        function(Y){
                var hijaxForm = function() {
                        …
                };
                Y.on("available", hijaxForm, "#monformulaire");
        });
</script>
</body>
</html>
```

Le navigateur affiche le document html, puis évalue le contenu de la balise script, ce qui déclenche l'appel de la fonction anonyme qui associe la fonction hijaxForm à l'événement « available » du nœud « monformulaire ».

Ceci a pour effet d'exécuter hijaxForm dès que « monformulaire » est créé et disponible dans la page html.

Ce fonctionnement est celui retenu pour le progressive enhancement :

- chaque document html doit être sémantiquement valide et accessible,
- les styles css sont utilisés pour la présentation,
- les scripts d'enrichissement JavaScript sont exécutés via les accroches événementielles.

3.3.2 Normes pour les JSP

Les normes de développement pour les JSP se basent sur les spécifications JavaServer Pages (JSP) en version 2.1.

3.3.2.1 Compilation des JSP

Toutes les JSP doivent compiler.

Les fragments de JSP inclus à la compilation via le tag <%@ include file…> doivent être suffixés

.jsp si le fichier correspondant compile;

.inc sinon

3.3.2.2 Entête JSP et jeu de caractères

Le jeu de caractère UTF-8 doit être positionné dans l'entête de chaque JSP (pas de factorisation dans une JSP incluse) :

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

- L'attribut contentType correspond au type de contenu envoyé au navigateur via un header http,
- L'attribut pageEncoding correspond au jeu de caractère de la JSP elle-même

3.3.2.3 JSP Incluse et jeu de caractères

Le jeu de caractères des fragments JSP inclus avec la directive `<%@ include …"%>` doit être précisé dans le web.xml avec la déclaration suivante :

```
<jsp-config>
<jsp-property-group>
<url-pattern>*.inc</url-pattern>
<page-encoding>UTF-8</page-encoding>
</jsp-property-group>
</jsp-config>
```

3.3.2.4 Escape & Encodage

Les chaînes de caractère doivent être converties en version compatible XML (remplacement > par > …).

On peut effectuer cette conversion de deux manières :

- Pour les JSP utilisant directement la JSTL, il existe deux façons de procéder :

une expression EL du type

```
${x.y}
```

doit être remplacée par un code du type

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
…
<c:out value="${x.y}"/>
```

ou encore

```
<%@ taglib prefix="f" uri="http://java.sun.com/jsp/jstl/functions"%>
…
${f:escapeXml(x.y)}
```

- Pour les JSP Struts, le tag à utiliser est s:property :

le code de remplacement est

```
<%@ taglib prefix="s" uri="/struts-tags"%>
…
<s:property value="x.y"/>
```

- Pour les JSP Struts générant un flux xml, les attributs escapeXml et escapeHtml doivent être correctement valorisés :

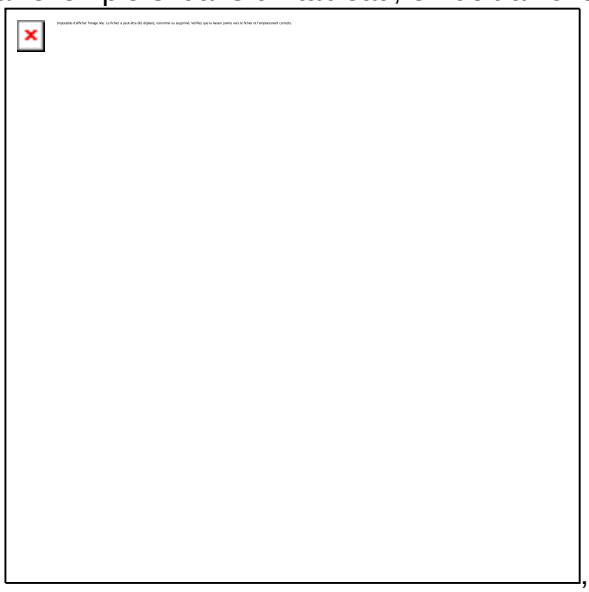
```
<nom><s:property value="nom" escapeXml="true" escapeHtml="true"/></nom>
```

- L'utilisation du tag s:property dans du code JavaScript fait également l'objet d'un échappement spécifique. Il faut activer l'attribut "escapeJavaScript" :

```
<s:property value="getText('GESTION_PARTENAIRE.recherche.tableau.caption')" escapeJavaScript="true" />
```

Pour des raisons de sécurité, toutes les données saisies par l'utilisateur doivent être échappées vis-à-vis des caractères < , > et " lors de la restitution.

C'est le cas pour les données destinées à être affichées telles quelles mais c'est également le cas pour les textes alternatifs associés à des icônes.

Par exemple si dans un tableau, on doit afficher le texte « modifier le client Nom Prénom » au survol de l'image  , le contenu de l'attribut `alt` du tag html correspondant devra lui aussi être échappé correctement.

3.3.2.5 Warnings OGNL et Erreurs Tomcat « Invalid chunk … »

La version 2.3 de Struts génère systématiquement des warnings dans les logs si un paramètre envoyé en POST ou en GET ne porte pas de nom. Bien que cette information soit utile durant la phase de développement, la fréquence d'apparition de ces traces est importante et peut polluer inutilement les logs en production.

Il est donc conseillé de passer le niveau de log OGNL à ERROR dans le fichier `log4j.properties` du livrable :

```
# Supression des warnings OGNL (visibles en dev)
log4j.logger.com.opensymphony.xwork2.util.OgnlUtil=ERROR
log4j.logger.com.opensymphony.xwork2.ognl.OgnlValueStack=ERROR
```

Tomcat 6 génère aussi des erreurs, dans ce même cas. Cela se traduit dans les logs catalina.out par des lignes de ce type :

```
INFO: Invalid chunk starting at byte [12] and ending at byte [12] with a value of [null] ignored
```

La solution la plus simple consiste à ajouter un nom (attribut « name ») à chaque paramètre posant problème. C'est souvent le cas sur les boutons « submit » des formulaires :

```
<s:submit cssStyle="button" type="button" value="%{getText('btn.rechercher')}" name="btnRechercher" />
```

Afin de ne pas envoyer inutilement le nom du bouton dans la requête, il est également possible de le filtrer grâce à l'interceptor Struts « param » en surchargeant l'existant. Pour plus d'information, se référer à la documentation en ligne qui fourni un exemple : http://struts.apache.org/release/2.3.x/docs/parameters-interceptor.html

3.3.2.6 Prologue XML et ContentType

Dans le cas d'une génération d'un document XML par une JSP, les contentType JSP et XML doivent être les mêmes :

```
<?xml version="1.0" encoding="UTF-8"?>
<%@ page language="java" contentType="text/xml; charset=UTF-8" pageEncoding="UTF-8"%>
```

3.3.2.7 URL et contexte Web

Les JSP ne doivent pas contenir d'URL ou le contexte de l'application « en dur ».
Pour les ressources dynamiques (ex. url servie par Struts), il faut utiliser le tag « c:url » ou « s:url ».

3.3.3 Implémentation des templates

Les templates JSP sont à définir dans les répertoires WebContent/WEB-INF/tiles-jsp et WebContent/WEB-INF/xml-jsp respectivement pour les fragments de page html et les flux XML.

Pour les autres types de pages (comme les exports), les templates JSP et XSL sont à définir dans le répertoire WebContent/WEB-INF/templates.

Exemple de template JSP pour générer un flux XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<%@ page contentType="text/xml; charset=UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s"%>
<ARTICLES>
  <s:iterator value="list">
        <ARTICLE>
              <ID><s:property value="productId"/></ID>
              <LABEL><s:property value="productLabel"/></LABEL>
              <PRICE><s:property value="productPrice"/></PRICE>
              <DATE><s:property value="productValidity"/></DATE>
        </ARTICLE>
  </s:iterator>
</ARTICLES>
```

Attention : Ne pas oublier le contentType (généralement text/xml pour les flux XML) !

Les données spécifiées dans les tags de template (JSP ou XSLT) sont recherchées :

- dans le pageContext (variables créées dans la JSP)
- dans les propriétés de l'action
- dans les attributs de requête (requestScope)
- dans les attributs de session (sessionScope)
- dans les attributs d'application (applicationScope)

Si le contexte n'est pas spécifié, Struts effectue une recherche dans cet ordre.

3.3.3.1 Inclusion / Import de XSL :

```
<xsl:include href="response:/templates/com/CommonMethod.xsl" />
<xsl:import href="response:/templates/com/CommonMethod.xsl" />
```

3.3.3.2 Content-type XSL

Struts 2 se base sur l'attribut media-type pour créer le content-type du flux de réponse http. Par exemple pour générer un document Excel il faut avoir:

```
  <xsl:output method="html" encoding="UTF-8" media-type="application/excel" />
```

3.3.4 Mise en page

Il est préconisé d'utiliser la librairie CSS YUI Grids pour structurer le contenu des pages html : http://yuilibrary.com/yui/docs/cssgrids.

La mise en page est réalisée à l'aide de grilles CSS. Une « grille » (yui3-g) contient une ou plusieurs « unités » (yui3-u). Le type d'unité choisi défini la dimension de l'élément (par exemple « yui3-u-1-2 » prend la moitié de la grille, « yui3-u-1-3 » prend un tiers, etc).

3.3.5 Personnalisation des tags Struts 2 pour Hornet

3.3.5.1 Déclaration du template FreeMarker « hornet »

Lors de l'utilisation d'un tag Struts 2 comme par exemple « s:select » dans une JSP, le framework Struts 2 génère le code HTML associé ainsi que le style de l'apparence du composant.

Le style et le code HTML généré pour le composant sont définis dans un thème de Struts 2.

Il y a 3 thèmes de base : simple, xhtml (utilisé par défaut) et css_xhtml.

Un nouveau thème Struts 2, « hornet », a été défini. Il répond à des normes d'accessibilités.

Le répertoire « src.template.hornet » contient les différents fichiers FreeMarker (.ftl).
Chaque fichier permet de configurer le code HTML généré pour un composant ou une partie de composant.
Par exemple le fichier « actionerror.ftl » se présente comme ceci :

```
<#if (actionErrors?exists && actionErrors?size > 0)>
    <#if parameters.title?exists>
        ${parameters.title}<#rt/>
    </#if>
    <ul>
    <#list actionErrors as error>
        <li><span<#rt/>
        <#if parameters.cssClass?exists>
         class="${parameters.cssClass?html}"<#rt/>
        <#else>
         class="errorMessage"<#rt/>
        </#if>
        <#if parameters.cssStyle?exists>
         style="${parameters.cssStyle?html}"<#rt/>
        </#if>
        >${error}</span></li>
    </#list>
    </ul>
</#if>
```

On trouve aussi un fichier de configuration « theme.properties » qui permet de déclarer le thème parent utilisé. Le thème parent utilisé dans Hornetserver 2.0 est le thème xhtml.
Le fichier « theme.properties » est le suivant :

```
parent = xhtml
```

Si un composant Struts 2 n'est pas défini dans le thème Hornet, le composant défini dans le thème parent xhtml sera utilisé. Cela permet de ne pas être obligé de redéfinir tous les composants de Struts 2 mais seulement ceux qui le nécessitent.

3.3.5.2 Redéfinition du thème à l'intérieur d'un projet

Si un projet n'a pas besoin de redéfinir certains composants, il peut utiliser directement le thème Hornet comme thème parent.
Pour cela, déclarer dans le fichier struts.xml le thème Hornet du Framework comme ce qui suit :

```
<constant name="struts.ui.theme" value="hornet" />
```

Si un projet a besoin de redéfinir certains composants de Struts 2, il faut qu'il créer un nouveau thème qui intégrera les nouvelles définitions des composants à redéfinir.
Pour cela, déclarer dans le fichier struts.xml le thème propre au projet, appelé THEME_PROJET, comme ce qui suit :

```
<constant name="struts.ui.theme" value="THEME_PROJET" />
```

Puis ensuite, à la racine des sources java, créer le répertoire « template » qui intègre un répertoire nommé THEME_PROJET.
Dans ce répertoire, créer un fichier de configuration « theme.properties » qui permet de déclarer le thème Hornet comme thème parent, comme ceci :

```
parent = hornet
```

Pour ajouter des actions au comportement d'un composant existant dans le thème parent Hornet, il suffit de faire appel à la déclaration du composant dans le thème parent puis de rajouter le code. Par exemple si on veut redéfinir le form.ftl du thème Hornet, il faudrait faire comme ce qui suit :

```
<#include "/${parameters.templateDir}/hornet/form.ftl" />
CODE AJOUTÉ
```

Ici on inclut le ftl initial d'Hornet puis on ajoute du code propre au projet.

Il est aussi possible de déclarer un composant qui n'est pas présent dans le thème Hornet. Si on veut appeler le composant du thème parent à Hornet, il suffit de faire comme ceci :

```
<#include "/${parameters.templateDir}/simple/textarea.ftl" />
CODE AJOUTÉ
```

Ici on inclut le ftl initial du thème parent d'Hornet puis on ajoute du code propre au projet.

3.3.6 Composants

Le framework hornetclient apporte :

- des normes et des documents d'accompagnement :
  - sélection de « comment faire » et « quoi utiliser » avec YUI,
- des composants et des surcouches :
  - création de composants,
  - extension de composants existants,
- pour l'ergonomie et l'accessibilité :
  - des normes et « comment faire » spécialisés autour de l'accessibilité en environnement maitrisé,
  - des normes et « comment faire » spécialisés autour de l'accessibilité en environnement non maitrisé.

3.3.6.1 Menu horizontal

Utilisation du composant YUI MenuNav-Node (cf. http://yuilibrary.com/yui/docs/node-menunav/) pour transformer du code existant sous forme de liste ul/li, en menus déroulants accessibles.

Les styles CSS sont positionnés pour définir la structure et l'orientation du menu dynamique construit par le composant YUI.

Par exemple, le code html suivant permet de créer un menu, le code javascript permet d'activer la version dynamique.

```
<div id="menu" class="yui3-menu yui3-menu-horizontal">
<div class="yui3-menu-content">
  <ul class="first-of-type">
        <li class="yui3-menuitem">
              <a class="yui3-menuitem-content" href="accueil.html">Accueil</a>
        </li>
        <li>
              <a class="yui3-menu-label" href="#submenu-1">Gestion des partenaires</a>
              <div id="submenu-1" class="yui3-menu">
              <div class="yui3-menu-content">
                    <ul>
                          <li class="yui3-menuitem">
                                <a class="yui3-menuitem-content"
href="recherchePartenaires.html">Rechercher</a>
                          </li>
                    </ul>
              </div>
              </div>
        </li>
        <li>
```

```
                <a class="yui3-menu-label" href="#submenu-2">Administration</a>
                <div id="submenu-2" class="yui3-menu">
                <div class="yui3-menu-content">
                        <ul>
                                <li>
                                        <a class="yui3-menu-label" href="#submenu-2-1">Secteurs</a>
                                        <div id="submenu-2-1" class="yui3-menu">
                                        <div class="yui3-menu-content">
                                                <ul>
                                                        <li class="yui3-menuitem">
                                                                <a class="yui3-menuitem-content"
href="listeSecteurs.html">liste des secteurs</a>
                                                        </li>
                                                </ul>
                                        </div>
                                        </div>
                                </li>
                        </ul>
                </div>
                </div>
        </li>
  </ul>
</div>
</div>

<script>
YUI().use("node-menunav", function(Y) {

Y.on("contentready", function (e) {
  this.plug(Y.Plugin.NodeMenuNav, { autoSubmenuDisplay: false });

}, "#menu");
});
</script>
```

Dès que le contenu du nœud « menuAppli » est près, le composant YUI node-menunav va créer le menu dynamique à partir de la définition HTML du menu présente dans le doc HTML sous forme de liste ul/li.

- Accueil
- Gestion des partenaires
  - Rechercher
- Administration
  - Secteurs
    - Liste des secteurs

Figure 2 : Exemple de menu affiché sans CSS ni JavaScript

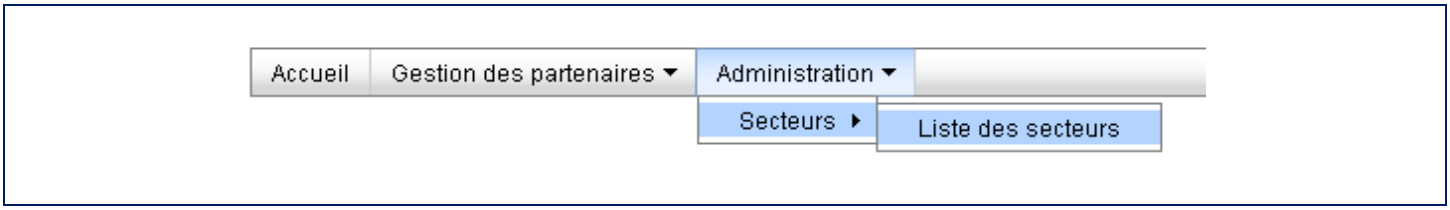


Figure 3 : Exemple de menu affiché dynamiquement avec CSS et Javascript

3.3.6.1.1Génération dynamique du menu

La génération du menu ainsi que le plan du site et le fil d'Ariane peuvent être entièrement dynamique en renseignant uniquement un fichier *menu.xml*. Ce processus est décrit dans le chapitre 3.9.1.

3.3.6.2 Formulaire

Utiliser les Tags Struts2 pour générer le formulaire et ses éléments
Exemple de création d'un formulaire :

```
<s:form action="" id="idform" >

   <s:label for="id1" value="%{getText('champs1.libelle')}" requiredLabel="true"/>
   <s:textfield id="id1" name="champs1" />

   <s:label for="id2" value="%{'champs2'}" requiredLabel="true" />
   <s:datetimepicker id="id2" name="champs2" displayFormat="dd/MM/yyyy" />

   <s:submit value="Rechercher" />
   <s:reset value="Reinitialiser" />
</s:form>
```

Remarques :

- Depuis le passage à Struts 2.3.16, les tags « s:form » doivent avoir une action définie.
- Un bouton submit non visible peut être placé au début du formulaire. Il permet de s'assurer que l'action appelée en cliquant sur « enter » dans un champ est bien l'action par défaut. (D'autres boutons submit associés à d'autres actions peuvent en effet être présents dans le formulaire).

```
 <s:submit id="btnValiderDefault" cssClass="none" value="Valider" name="btnValider" />
```

3.3.6.2.1Libellé de champ

Pour les labels : utilisation du Tag Struts2 « label » afin de générer automatiquement le label, le caractère obligatoire ou non, ainsi que la bulle d'aide si nécessaire (Cf. Tag Strut2 « tooltip »).

Exemple de création d'un label avec une bulle d'aide :

```
   <s:label id="id1Label" for="id1" value="<abbr lang='en' title='Label'>LBL</abbr>" requiredLabel="true"
      tooltip="%{'tooltip'}"
      tooltipConfig="#{'tooltipIcon':'/static/images/tooltip.png', 'jsTooltipEnabled':'true'}">
      <s:param name="escape" value="false" /></s:label>
```

Remarque : Pour désactiver l'encodage sur les valeurs des labels, il faut ajouter l'attribut « escape » à **false**.

Remarque : Dans le cas d'un label avec une bulle d'aide, il peut être nécessaire de préciser un identifiant qui sera utilisé pour générer le composant JavaScript « tooltip » (sinon le composant ne fonctionnera pas si plusieurs bulles d'aide ayant un libellé identique sont présentes dans la page).

3.3.6.2.2Calendrier

Note : Depuis la version 3.5 d'Hornet, ce composant n'utilise plus pour son implémentation de composants YUI2 ; il est donc 100% compatible YUI3.

3.3.6.2.2.1 YUI

Utilisation du composant YUI 3 Calendar : cf. http://yuilibrary.com/yui/docs/calendar/ associé avec le composant YUI 3 Panel : cf. http://yuilibrary.com/yui/docs/panel/.

Le composant proposé **Y.hornet.fieldcalendar** permet d'associer facilement une zone de saisie de date avec un calendrier. Il existe en deux modes :

- Avec un conteneur créé à la volée, le calendrier s'affiche alors en sur impression de la page HTML sans perturber le rendu,
- Avec un conteneur existant dans la page html, le calendrier s'affiche dans le conteneur désigné, les autres composants de la page s'ajustent alors en fonction de la place disponible.

L'assemblage fait par ce composant a pour but :

- De standardiser les libellés en français par défaut,
- De formater les dates et les zones de saisie selon les conventions françaises par défaut,
- D'ajouter des boutons et événements pour réinitialiser ou fermer la boite de dialogue.

3.3.6.2.2.2 Dans le cadre de Hornet

Utilisation du Tag Hornet « calendar » pour générer automatiquement le bouton et le composant calendrier associés à un champ de saisie.

Par exemple, le code suivant permet de créer un champ de saisie de date, et d'y associer un calendrier si le JavaScript est activé.

```
<hornet:calendar id="date" name="date" displayFormat="dd/MM/yyyy" />
```

Exemple de code généré dans la page :

```
<div>
<input type="text" size="10" name="date" id="date" />
<button type="button" class="js-disabled-hidden agenda icone" id="date_show">
  <img id="date_img" src='/Application/static/images/agenda.png' alt="Agenda" />
</button>
<div id="date container"></div>
</div>

<script type="text/javascript">
//<![CDATA[
hornet().use("event", "node", "hornet-fieldcalendar", function (Y) {
   Y.on("domready", function(){
      var calendar = new Y.hornet.fieldcalendar(
         "date", {
            displayFormat:"dd/MM/yyyy",
            button : "button[id='date show']",
            container : "div[id='date container']"
      });
   });
});
//]]>
</script>
```

Figure 4 : Exemple d'utilisation du calendrier associé à un champ de formulaire

La taille du champ de saisie est fixée avec une valeur par défaut égale à 10 (size="10"). Pour modifier cette valeur, il faut la passer comme paramètre au tag Hornet :

```
<hornet:calendar id="date" name="date" displayFormat="dd/MM/yyyy">
    <s:param name="size" value="20" />
</hornet:calendar>
```

3.3.6.2.3Listes liées

Le composant « liste liée » se présente sous la forme d'une liste déroulante associée à un bouton de type « submit ». Lors du clic sur ce bouton, le formulaire est envoyé afin de recharger la page avec les données courante et la liste liée mise à jour.

Javascript activé, le bouton associé doit être masqué et le rechargement de la liste liée effectué automatiquement à chaque changement de la liste parente.

Remarque : Utilisation du Tag Struts2 « doubleselect » afin de générer automatiquement la liste déroulante et le code Javascript permettant d'activer la version dynamique.

A fournir :

- L'id de la liste déroulante,
- Le nom associé à la liste déroulante pour envoyer les données avec le formulaire,
- La liste de données :
  1. directement sous forme de Map de clefs-valeurs,
  2. sous forme de liste d'objets, en précisant l'attribut à utiliser pour le libellé et celui pour la valeur.
- Le nom associé à la liste déroulante parente,
- L'id de la liste déroulante parente,
- La deuxième liste de données : vide (pour assurer la compatibilité avec la taglib struts),
- L'id du bouton associé à la liste déroulante,
- L'url à appeler pour recharger la liste déroulante avec requête AJAX.

Optionnels :

- Le libellé à ajouter comme titre dans la zone d'erreur (si erreurs durant la requête AJAX)
- Choix si une option vide doit être ajoutée au rechargement de la liste déroulante,
- Le texte de l'option vide,
- La valeur de l'option vide.

Par exemple, le code suivant permet de créer les listes déroulantes et le bouton à l'aide des taglibs Struts.

```
<!– Liste deroulante parente -->
<s:select
id="idListeParente"
name="idListeParente"
list="listeParente" listKey="id" listValue="libelle" />

<!– Bouton pour recharger la page -->
<s:submit
id="boutonRecharger"
type="button"
action="rechargerListeLiee"
value="Recharger" />
<!– Liste deroulante liee -->
<s:doubleselect
    id="idListeLiee"
    name="idListeLiee"
    list="listeLiee" listKey="id" listValue="libelle"
    doubleName="idListeParente" doubleId="idListeParente" doubleList="">
    <s:param name="buttonId">boutonRecharger</s:param>
    <s:param name="urlAction">rechargerListeLiee.xml?mode=XML</s:param>
    <s:param name="titleError">Erreur de rechargement de la liste</s:param>
    <s:param name="addEmptyOption" value="false"/>
    <s:param name="emptyOptionText"></s:param>
    <s:param name="emptyOptionValue"></s:param>
</s:doubleselect>
```

Par exemple, le code JavaScript suivant permet d'activer le rechargement dynamique de la liste à l'aide de requêtes AJAX.

```
<script type="text/javascript">
    //<![CDATA[
    hornet().use("node", "event", "io", function (Y) {

       var boutonRecharger = Y.one("#boutonRecharger button, #boutonRecharger"),
       listeParente = Y.one("select[id='idListeParente']"),
       listeLiee = Y.one("select[id='idListeLiee']");

       if(!!boutonRecharger && !!listeParente && !!listeLiee) {
           // masquage du bouton
           boutonRecharger.remove();

           var config = {
               url : "rechargerListeLiee.xml?mode=XML"
           };

           // A chaque changement dans la liste parente
           listeParente.after('change', function(e) {
               var newValue = this.get('value');

               // mise a jour du contenu avec requete AJAX
               reloadDataList(listeLiee, "idListeParente=" + newValue, config);
           });
       }

        /**
         * Rechargement des options de la liste a partir de requete AJAX
         * @param selectList {Y.Node} Liste deroulante a mettre a jour
         * @param data {String} donnees a envoyer
         * @param config {Object}
         *      - url {String} : uri pour la requete AJAX
         */
        function reloadDataList(selectList, data, config) {
         //…
       }

    });
    //]]>
</script>
```

Le flux de retour attendu sera un document XML du type :

```
<PAGE>
<DATA>
  <item>
         <value>1</value>
         <text>Option 1</text>
         <selected>true</selected>
  </item>
…
</DATA>
</PAGE>
```

3.3.6.2.4Rattachement

Pour répondre à des critères d'accessibilité, le composant rattachement doit être réalisé sous la forme d'une liste ul/li de check box et labels.

Javascript activé, le composant rattachement du Framework permet de transformer cette liste de check box, en deux listes « multiselect » combinées avec des boutons d'actions.

Remarque : Utilisation du Tag Hornet « rattachement » afin de générer automatiquement la liste de checkbox et le code Javascript permettant d'activer la version dynamique.

A fournir :

- L'id du composant rattachement,
- Le nom à associer aux éléments HTML pour envoyer les données avec le formulaire,
- La liste de données :
  1. directement sous forme de Map de clefs-valeurs,
  2. sous forme de liste d'objets, en précisant l'attribut à utiliser pour le libellé et celui pour la valeur.

- Le nom à associer à l'élément HTML pour la deuxième liste (chaîne vide autorisée),
- La deuxième liste de données : vide (pour assurer la compatibilité avec la taglib struts).

Optionnels :

- Les libellés des labels associés aux deux listes du composant rattachement
- La taille des listes du composant rattachement
- Les libellés des boutons d'action du composant rattachement
- Les boutons d'action à afficher avec le composant rattachement

Par exemple :

```
<hornet:rattachement
  id="composantRattachement" name="selectionValeurs"
  list="listeElements" listKey="valeur" listValue="libelle"
  doubleName="" doubleList=""
  rightTitle="Elements sélectionnés" leftTitle="Elements non sélectionnés"
  size="15"
  addToRightLabel="Sélectionner" addToLeftLabel="Désélectionner"
  addAllToRightLabel="Sélectionner tout" addAllToLeftLabel="Désélectionner tout"
  allowAddToRight="true" allowAddToLeft="true"
  allowAddAllToRight="true" allowAddAllToLeft="true"
/>
```

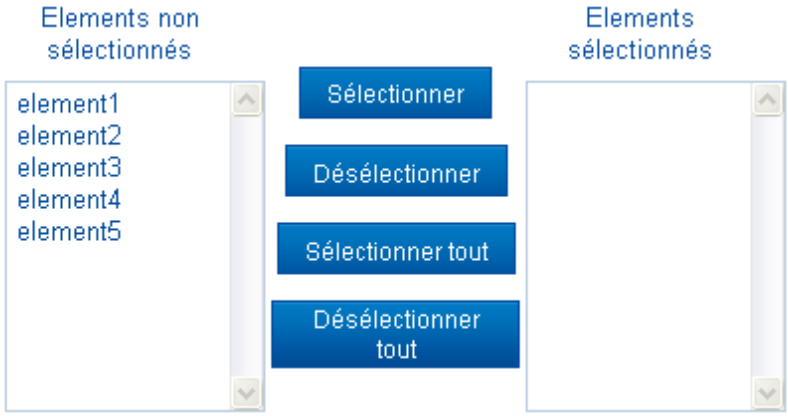
Permet d'obtenir le code suivant dans la page :

```
<div id="composantRattachement">
<ul>
<li>
    <input name="selectionValeurs" value="1" id="selectionValeurs-1" type="checkbox">
    <label for="selectionValeurs-1" class="checkboxLabel">element1</label>
</li>
<li>
    <input name="selectionValeurs" value="2" id="selectionValeurs-2" type="checkbox">
    <label for="selectionValeurs-2" class="checkboxLabel">element2</label>
</li>
<li>
    <input name=selectionValeurs"" value="3" id="selectionValeurs-3" type="checkbox">
    <label for="selectionValeurs-3" class="checkboxLabel">element3</label>
</li>
<li>
    <input name="selectionValeurs" value="4" id="selectionValeurs-4" type="checkbox">
    <label for="selectionValeurs-4" class="checkboxLabel">element4</label>
</li>
<li>
    <input name="selectionValeurs" value="5" id="selectionValeurs-5" type="checkbox">
    <label for="selectionValeurs-5" class="checkboxLabel">element5</label>
</li>
</ul>
</div>
<script type="text/javascript">
//<![CDATA[
    hornet().use('node', 'event', 'hornet-rattachement', function(Y) {
        Y.on("domready", function(){
            var checkboxList = Y.one("div[id='composantRattachement']");
            if(checkboxList && checkboxList.one('ul')) {
                var rattachement = new Y.hornet.rattachement({
                    srcNode: checkboxList
                    ,selectedListName: "selectionValeurs"
                    ,availableListName: ""
                    ,selectedLabel: "Elements sélectionnés"
                    ,availableLabel: "Elements non sélectionnés"
                    ,size: "15"
                    ,selectActionLabel: "Sélectionner"
                    ,deselectActionLabel: "Désélectionner"
                    ,selectAllActionLabel: "Sélectionner tout"
                    ,deselectAllActionLabel: "Désélectionner tout"
                    ,allowSelectAction: true
                    ,allowDeselectAction: true
                    ,allowSelectAllAction: true
                    ,allowDeselectAllAction: true
                }).render();
```

```
            }
        });
    });
//]]>
</script>
```

Résultat avec Javascript désactivé :

Résultat avec Javascript activé :



3.3.6.2.5Validation de formulaire

3.3.6.2.5.1 Composants YUI

Utilisation d'un composant de validation de formulaire pour permettre de contrôler les champs de saisie avant l'envoie du formulaire.

A fournir :
- L'id du formulaire à valider,
- L'élément containeur DIV où seront affichés les différents messages d'erreur,
- Pour chaque élément de saisie du formulaire à contrôler : les objets Validateur associés.

Construction d'objet de validation de type Validator.
A fournir :
- Le message d'erreur à renvoyer,
- La fonction de validation à exécuter,
- L'id de l'élément du formulaire à contrôler,

Optionnels :
- L'id de l'ancre ciblant l'élément du formulaire.

Par exemple, avec une page du type :

```
<html> …
<body>
<div class="messageBox errorBox"></div>

…
<form id="monformulaire">

   <div class="formmgr-row">
        <div>
            <label for="fieldElement">libellé</label>
        </div>
        <div>
            <div class="formmgr-message-text"></div>
            <a href="#fieldElement_anchor" id="fieldElement_anchor" tabindex="-1"></a>
            <input class="formmgr-field" id="fieldElement" name="fieldElement" />
        </div>
    </div>

  …

</form>
…
<script>
// Accroche YUI
YUI().use('node', 'hornet-validatorform',
  function(Y){
  var hijaxFormValid = function() {
  …
          // Fonction de validation du formulaire
          var f = new Y.hornet.ValidatorForm("monformulaire",{
              // activation des messages d'erreur au niveau des champs erronés
              inlineErrorMessage: true
          });
          f.prepareForm();
          this.on('submit', function(e) {
                  var testval = this.validateForm();
                  if( !testval ){
                          e.halt();
                          e.preventDefault();
                          return false;
                  }
                  return true;
          }, f);
          this.on('reset', function(e) {
                  this.clearForm();
                  return true;
          }, f);


          // Zone à utiliser pour afficher les messages d'erreur
          f.set("errorBox", Y.one(".errorBox"));

          // Ajout d'un validateur pour un élément du formulaire
          var requiredValidator = new Y.hornet.Validator({ errorMessage: "Erreur!"});
          requiredValidator.validate = function (form, elt) {
                  var testval = false;
                  var value = elt.get('value');
                  if (!!value && value != "") {
                          testval = true;
                  }
                  return testval;
          };
          f.addValidator("fieldElement", requiredValidator, "fieldElement anchor");

  };
  Y.on("available", hijaxFormValid, "#monformulaire");
  });

…
<script>
</body>
</html>
```

La fonction hijaxFormValid est associée à l'événement « available » du nœud « monformulaire » et est alors exécutée dès que « monformulaire » est créé et disponible dans la page html.

Lorsque l'événement « submit » du formulaire est déclenché, la fonction de validation est appelée et l'envoie du formulaire par le navigateur est alors interrompu si la validation a échouée.

Les messages d'erreurs sont affichés sous forme de liens si une ancre leur est associée. Ils sont regroupés sous forme de liste ul/li dans la zone d'erreur spécifiée.

Si le mode d'affichage des messages d'erreur « en ligne » est activé, les messages d'erreur de validation sont aussi affichés au niveau des champs erronés. Les classes CSS suivantes doivent être présentes au niveau des champs du formulaire :

- « formmgr-row » : autour du champ (ou groupe de champs) pour définir le bloc d'erreur,
- « formmgr-message-text » : à l'interieur du bloc associé pour contenir le message d'erreur,
- « formmgr-field » : au niveau du champ (ou groupe de champs).

3.3.6.2.5.2 Dans le cadre de Hornet

Utilisation du Tag Struts2 « form » conjointement avec le mécanisme de validation Struts pour générer automatiquement ces contrôles de validation. (cf. section 3.2.3.2)

Seuls les validateurs suivants sont supportés :
- required validator
- requiredstring validator
- stringlength validator
- regex validator
- email validator
- url validator
- int validator
- double validator
- date validator

Pour activer la validation en JavaScript, il faut définir l'attribut « validate » à **true**.

Pour ajouter des contrôles de validation supplémentaires au formulaire (en plus de ceux générés), il faut les déclarer dans une méthode JavaScript qui sera passée au paramètre « validators ».

```
<script type="text/javascript">
//<![CDATA[

/**
 * Ajoute des validations au composant de validation de formulaire
 * @param Y
 * @param f {Y.hornet.ValidatorForm} Composant de validation
 * @see {Y.hornet.Validator}
 */
function addValidationRules(Y, f) {

} ;

//]]>
</script>

<s:form validate="true" id="idForm">
    <s:param name="validators">addValidationRules</s:param>
    <s:param name="inlineErrorMessage" value="true" />
    <s:param name="titleError"><h2 class='titleError'>Erreurs</h2></s:param>

</s:form>
```

Si l'attribut « titleError » est précisé, il sera utilisé comme titre pour la zone d'erreur.

Pour afficher aussi les messages d'erreur de validation au niveau des champs erronés, il faut définir l'attribut « inlineErrorMessage » à **true**.

3.3.6.3 Tableau

Principe : Génération du tableau HTML (valide W3C) avec toutes ses données dans la page. Amélioration progressive du tableau si le JavaScript est activé.

Utilisation des Tag Struts 2 pour itérer sur les données et générer le tableau.

Par exemple :

```
<div id="tableauContainer">
<s:if test="(liste != null)">

<h2 class="titreTableau">Titre du tableau</h2>

<table id="tableau" summary="Résumé">
<caption>Titre du tableau</caption>
<thead><tr>
   <th scope="col" id="nom" >Nom</th>
   <th scope="col" id="prenom" >Prénom</th>
   <th scope="col" id="dateNaissance" >Date de naissance</th>
</tr></thead>
<tbody>
<s:iterator value="liste" status="iterator"><tr>
   <td headers="nom" ><s:property value="nom"/></td>
   <td headers="prenom" ><s:property value="prenom"/></td>
   <td headers="dateNaissance" ><s:date name="dateNaissance" format="dd/MM/yyyy" /></td>
</tr></s:iterator>
</tbody>
</table>

</s:if>
</div>
```

Remarque : le titre du tableau est affiché dans un élément ayant la classe « titreTableau » positionné avant le tableau. Il est utilisé pour appliquer les styles des thèmes sur le composant tableau.

Pour ajouter des actions « en ligne » dans le tableau, ajouter la colonne correspondante dans l'entête et les liens d'action dans chaque ligne du tableau. (Attention lors de la génération des lignes du tableau : les identifiants DOM doivent être uniques dans l'ensemble de la page).

Les actions « outils » sont à ajouter avant/après la balise « table » dans des zones « outils », dans l'élément portant la classe « actions » :

```
<h2 class="titreTableau">…</h2>

   <div class="outils haut">
      <div class="actions"></div>
   </div>

<table>
//…
</table>

   <div class="outils bas">
      <div class="actions"></div>
   </div>
```

Attention aux performances : cf. page 134 « Faire attention à l'utilisation des tags Struts dans les itérations JSP pour les listes volumineuses. »

3.3.6.3.1 Principes tableau YUI

Note : Depuis la version 3.5 d'Hornet, le composant `hornet-pagintable` utilisant YU2 est supprimé ; son remplacant `hornet-datatable` utilise le composant YU3 DataTable pour son implémentation.

Les données du tableau peuvent être récupérées à partir d'un tableau HTML déjà présent dans la page ou à partir de requêtes Ajax.

Une fois le tableau complété, la gestion des données (pour la pagination et le tri des colonnes) peut être réalisée de deux façons différentes :

- Localement : les données sont récupérées à l'initialisation et ne sont plus mises à jour.
- Dynamiquement : les données sont récupérés à chaque nouvelle action sur le tableau (tri ou changement de page du tableau).

Utilisation d'un composant YUI 3 Datatable : cf. https://yuilibrary.com/yui/docs/api/modules/datatable.html conjointement avec un composant YUI 3 Datasource : cf. https://yuilibrary.com/yui/docs/api/modules/datasource.htmlhttp://developer.yahoo.com/yui/datasource/.

A fournir :

- L'id ou la référence vers un élément containeur DIV qui contiendra le code html généré,
- L'ensemble définissant les colonnes du tableau,
- Un objet DataSource qui s'occupe de la récupération des données à partir de différentes sources possibles (tableaux JavaScript, contenu HTML, requêtes vers un serveur distant, …).

Optionnels :

- Les éléments de configuration pour modifier ceux par défaut :
  - Les libellés de statut (chargement, erreur de récupération des données, tableau vide, …)
  - Activation de la pagination

Par exemple, avec une page du type :

```
<html> …
<head>
  <style type="text/css">
        /* Element a masquer le temps du chargement */
        .yui3-js-enabled .hornet-table-loading {
              display: none;
        }
  </script>
</head>
<body>
…
<div id="montableauContainer" class="table yui3-table-loading">
  …
</div>

…
<script>
// Accroche YUI
YUI().use('node', 'hornet-datatable', function(Y){
  …

  // Création du tableau
  var tableau = new Y.hornet.PEDataTable.createTable(
        "montableauContainer",
        columnDefs,
        dataSource,
        configDataTable);


  Y.one("#montableauContainer").removeClass("hornet-table-loading");
});
</script>
</body>
</html>
```

Au début du chargement de la page, un style peut être appliqué pour cacher la partie de code liée au tableau jusqu'à son chargement complet. Cela permet d'éviter un affichage temporaire du tableau entre le moment ou le code html est affiché et le code JavaScript a fini de s'exécuter.

3.3.6.3.1.1 Avec données locales

Pour cela, on créé une datasource par la méthode statique **createDatasource** en lui passant l'identifiant du tableau HTML <table> contenant les données à récuperer.

Remarque : Le composant ne supporte pas l'utilisation de **rowspan** dans le corps du tableau.

Un pré-requis indispensable à l'utilisation de ce composant est que le tableau HTML soit valide et respecte les normes W3C.

Par exemple, avec une page du type :

```
<html> …
<body>
…
<div id="tableauContainer">
  <table id="tableau" summary="…">
        <caption>titre du tableau</caption>
        <thead>
              <tr>
                    <th scope="col" id="nom" >Nom</th>
                    …
              </tr>
        </thead>
        <tbody>
              <tr>
                    <td headers="nom" >nom 1</td>
                    …
              </tr>
              <tr>
                    <td headers="nom" >nom 2</td>
                    …
              </tr>
              …
        </tbody>
  </table>
</div>
```

```
<script type="text/javascript">
//<![CDATA[
hornet().use("event", "node", "hornet-datatable", function (Y) {

var myLocalDataSource = null, myDataTable = null;

// Definition des colonnes du tableau
var myColumnDefs = [
   { key: "nom", label: "Nom", sortable:true }
];

// Configuration du tableau
var tableConfig = {
  caption: "Titre du tableau",
  summary: "Résumé"
  initialLoad: true },

  tableId = "tableau";


   // DataSource en utilisation locale
   myLocalDataSource = Y.hornet.PEDataTable.createDatasource(null,
               myColumnDefs, null, { table: tableId }, null);

   // Initialisation du tableau yui
   myDataTable = Y.hornet.PEDataTable.createTable("tableauContainer",
               myColumnDefs, myLocalDataSource, tableConfig);

   }

});

//]]>
</script>
```

3.3.6.3.1.2 Avec données dynamiques et requête AJAX

Pour cela, on créé une datasource par la méthode statique **createDatasource** en lui passant en paramètre l'url du flux XML ainsi que le nom du nœud XML contenant les données. Celui-ci sera alors chargé dans le tableau par AJAX en utilisant la méthode **load()**.

Par exemple, avec une page du type :

```
<html> …
<body>
…
<div id="tableauContainer">
</div>

…
<script>
// Accroche YUI
YUI().use('node', 'event', hornet-datatable',
  function(Y){
        var hijaxTable = function() {
               // Definition des colonnes du tableau
               var myColumnDefs = [
                  { key: "ID", label: "Id", sortable:true },
                  { key: "NUMERO_COMPTE", label: "Numéro de compte", sortable:true },
                  …
               ];
               var urlFlux = "tableau.html?mode=XML";

               // DataSource en utilisation XHR
               var myDataSource = Y.hornet.PEDataTable.createDatasource(urlFlux,
                       myColumnDefs, "item", null, null);

               var myDataTable = Y.hornet.PEDataTable.createTable("tableauContainer",
                       myColumnDefs, myDataSource, tableConfig);

               myDataTable.datasource.load();

               …

               // On peut changer ensuite l'url de la source avec la method setLiveData
               myDataSource.setLiveData("tableau2.html?mode=XML")
               myDataTable.datasource.load();
        }

        …
  });
</script>
</body>
</html>
```

Le flux de retour attendu sera un document XML du type (tableau.html?mode=XML):

```
<PAGE>
<items>
  <item>
        <ID>1</ID>
        <NUMERO_COMPTE>0000000001</NUMERO_COMPTE>
        <DATE OPERATION>15/12/10</DATE OPERATION>
        <MONTANT>11</MONTANT>
        <QUANTITE>1</QUANTITE>
  </item>
  <item>
        <ID>2</ID>
        <NUMERO COMPTE>0000000002</NUMERO COMPTE>
        <DATE OPERATION>15/12/10</DATE OPERATION>
        <MONTANT>12</MONTANT>
        <QUANTITE>2</QUANTITE>
  </item>
…
</items>
</PAGE>
```

3.3.6.3.2Pagination et tri local

Cette solution implique que le tableau html est présent dans la page avec la totalité des données, sans pagination ni tri de colonne.

Le tableau amélioré est ensuite recréé, si le JavaScript est activé, à partir des données locales.
Pour un tri simple : simple configuration des colonnes du tableau.

Pour un tri avancé, se référer à la documentation du composant (cf. https://yuilibrary.com/yui/docs/api/classes/DataTable.html#attr_sortBy).

Pour la pagination : utilisation du composant Paginator dans la configuration du tableau (cf. https://yuilibrary.com/yui/docs/api/classes/DataTable.Paginator.html).

A fournir :

- le nombre d'item par page.

Optionnels :

- le format et les libellés pour créer les liens de pagination.
- Les autres choix offerts pour la pagination (items par pages)

A noter : dès que l'attribut **rowsPerPage** est renseigné, la pagination est automatiquement activée. Par exemple, le code suivant permet d'initialiser une pagination gérant 25 éléments par pages avec une personnalisation des libellés à utiliser pour le changement de page (autres que ceux employés sinon par défaut) :

```
<script type="text/javascript">
hornet().use("hornet-datatable", function (Y) {

        var configTableau = {
          // exemple de personalisation des libellés de la pagination
          firstPageLinkLabel    : "Première",
          previousPageLinkLabel : "Précédente",
          nextPageLinkLabel     : "Suivante",
          lastPageLinkLabel     : "Dernière",

          // titre
          caption : "Titre du tableau",

          // nombre de lignes par page (par défaut)
          rowsPerPage: 25,

          // indique les choix possibles pour la liste déroulante
          // afin de pouvoir changer le nombre de ligne par page
          pageSizes : [50, 100, 1000]
        };

        myDataTable = Y.hornet.PEDataTable.createTable(containerId,
          colonnesTableau, myDataSource, configTableau);
  }) ;
});
</script>
```

3.3.6.3.3Pagination et tri serveur

Dans l'action, implémenter l'interface « TablesStatesAware » comme dans l'exemple ci-dessous. La classe « TablesStatesMap » et l'interface « TablesStatesAware » sont fournies dans le projet *HornetTemplate* à partir de la version 2.0.

Exemple :

```
public class MonAction extends ActionSupport implements TablesStatesAware {

    /** Id du tableaux des partenaires */
    private static final String ID_TABLEAU_PARTENAIRE = "partenaires";

    /**
     * Etat du ou des tableau(x) de la page
     */
    private TablesStatesMap tablesStates;

    /** <code>totalItems</code> the totalItems */
```

```
    private int totalItems;



    /** {@inheritDoc} */
    public int getTotalItems() {
        return this.totalItems;
    }

    /**
     * Action de recherche
     *
     * @return success
     */
    public String recherche() {

        […]

        // Reset des criteres de filtrage
        if (this.isResetTableau()) {
            // RAZ de la pagination et du tri du tableau partenaire
            this.tablesStates.reset(RecherchePartenaire.ID_TABLEAU_PARTENAIRE);
        }

        […]

        // Pagination
        Pagination paginationPatenaires =
            this.tablesStates.get(
                RecherchePartenaire.ID_TABLEAU_PARTENAIRE).getPagination();

        // Recuperation totalCount
        this.totalItems = this.partenaires.size();
        // Recuperation nombrePages
        int nombrePages = this.totalItems / this.getItemsParPage();
        if ((this.totalItems % this.getItemsParPage()) > 0 || nombrePages == 0) {
            nombrePages++;
        }
        // Recalcul indexPage
        int indexPagePartenaires = Math.min(
            paginationPatenaires.getPageIndex(), nombrePages);
        paginationPatenaires.setPageIndex(indexPagePartenaires);

        // Pagination serveur
        int minIndex = (paginationPatenaires.getPageIndex() - 1) * this.getItemsParPage();
        int maxIndex = Math.min(
            this.totalItems, minIndex + this.getItemsParPage());
        this.partenaires = this.partenaires.subList(
            minIndex, maxIndex);

        if (this.hasErrors()) {
            return ERROR;
        }
        return SUCCESS;
    }
}
```

Ainsi les variables seront disponibles dans les JSP pour gérer la pagination et le tri.
L'action doit prendre en compte ces valeurs afin de récupérer une liste d'éléments réduite.

La classe « TableStatesMap » sert à stocker l'état des différents tableaux gérés dans une même action.
A un id de tableau on associe un objet représentant l'état du tableau en termes de pagination et de tri :

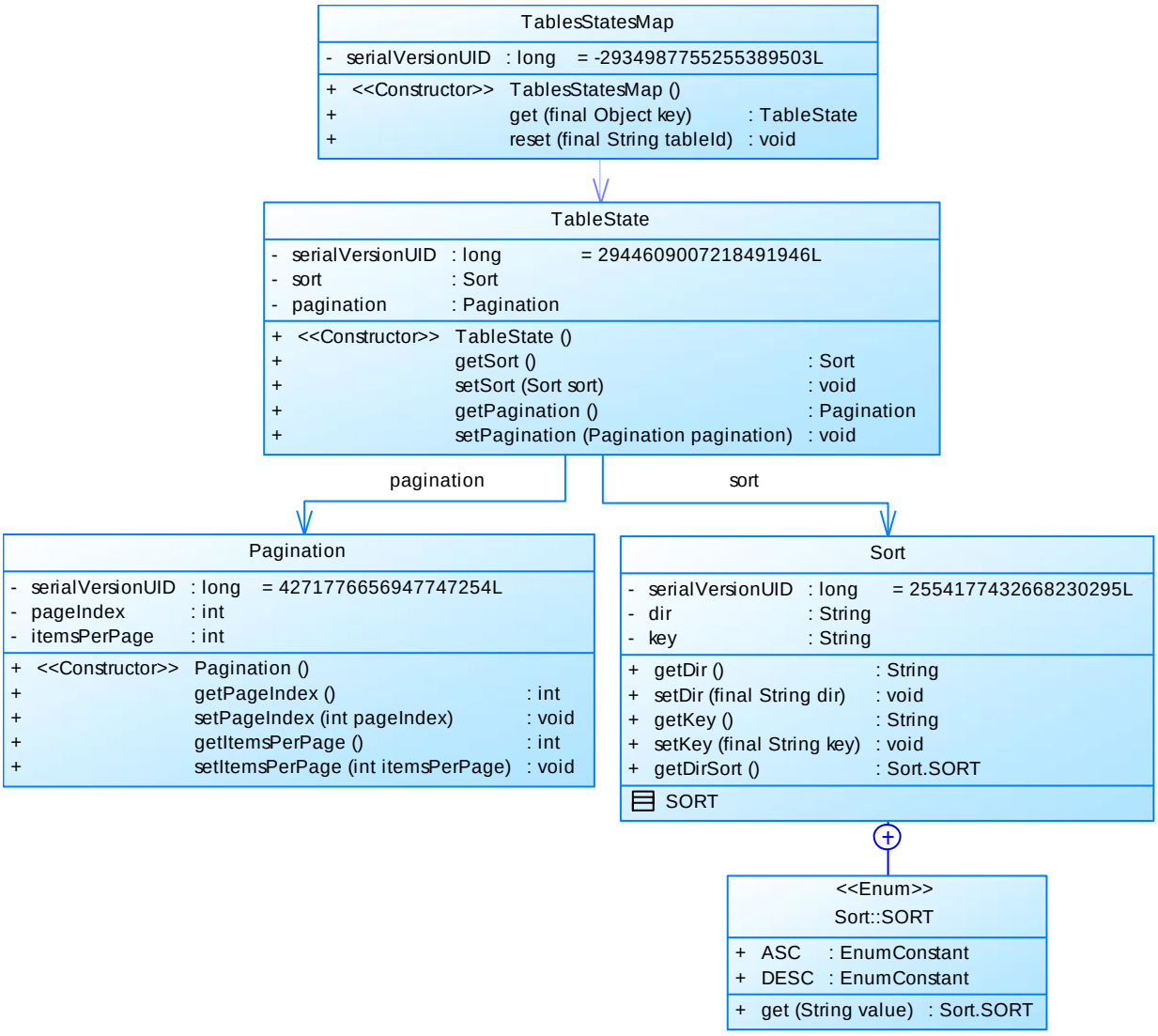


Figure 5 : Diagramme de classe TablesStatesMap

Dans la page : utilisation du tag Hornet « pagination » pour générer les liens de pagination.

Pour le tri, chaque entête de colonne est modifié en ajoutant un lien permettant de trier sur la colonne. Chaque tableau doit être identifié par un id unique pour permettre l'utilisation de plusieurs composants de pagination dans la même page.

L'accès à l'état d'un tableau dans une JSP se fait grâce à la notation entre crochets suivante :

```
${tablesStates['tableau1'].pagination.pageIndex}
${tablesStates['tableau1'].sort.key}
${tablesStates['tableau1'].sort.dir}
…
```

Exemple :

```
<div id="tableauContainer">
<s:if test="(liste != null)">

<s:url id="nomColumnUrl" includeParams="get" anchor="tableauContainer">

   <s:set var="sort" value="tablesStates['tableau1'].sort" />

   <s:param name="sort.key">nom</s:param>
   <s:param name="sort.dir"><c:choose><c:when test="${sort.key == 'nom' and sort.dir ==
'ASC'}">DESC</c:when><c:otherwise>ASC</c:otherwise></c:choose></s:param>
   <s:param name="indexPage">1</s:param>
</s:url>
```

```
<h2 class="titreTableau"></h2>
<table id="tableau" summary="">
<caption></caption>
<thead><tr>
   <th scope="col" id="nom" ><a href="${nomColumnUrl}">Nom</a></th>
</tr></thead>
<tbody>
<s:iterator value="liste" status="iterator"><tr>
   <td headers="nom" ><s:property value="nom"/></td>
</tr></s:iterator>
</tbody>
</table>

<!-- PAGINATION -->
<div class="yui-pg-container">
<hornet:paginator tableid="tableau1" count="${totalItems}" selected="${indexPage}"
itemPerPage="${itemsParPage}" anchor="tableauContainer" />
</div>

</s:if>
</div>
```

JavaScript activé : les valeurs de pagination et tri dans l'action sont utilisées pour initialiser les éléments YUI (configuration du tableau, pagination, …). Elles doivent être présentes aussi dans le flux XML retourné pour les requêtes AJAX.

Coté javascript le tableau est créé en utilisant la méthode statique **createServerTable** à la place de **createTable** et doit être initialisée avec l'attribut de configuration **tableName**.

```
// Configuration du tableau & de la pagination
var tableConfig = {
  tableName        : "tableau1"
  ,caption         : "<s:property value="getText('GESTION PARTENAIRE.recherche.tableau.caption')"/>"
  ,summary         : "<s:property value="getText('GESTION_PARTENAIRE.recherche.tableau.summary')"/>"
  ,initialLoad     : false
  ,rowsPerPage     : ${itemsParPage} <c:if test="${empty itemsParPage}">10</c:if>
  // if not provided, there is no last page or total pages.
  <c:if test="${!empty totalItems && totalItems > 0}">,totalRecords:${totalItems}</c:if>
  // page to activate at load
  <c:if test="${!empty pagination.pageIndex}">,initialPage:${pagination.pageIndex}</c:if>
  // current sort
  <c:if test="${!empty sort.key}">,sortedBy:{ ${sort.key}: "${fn:toLowerCase(sort.dir)}" }</c:if>
}

…

// DataSource instance
myDataSource = Y.hornet.PEDataTable.createDatasource(urlRecherche, myColumnDefs, "item", {
  table: tableId
  <c:if test="${!empty pagination.pageIndex}">,initialPage:${pagination.pageIndex}</c:if>
  <c:if test="${!empty totalItems && totalItems > 0}">,totalRecords:${totalItems}</c:if>
}, configNotification);

// DataTable instance
myDataTable = Y.hornet.PEDataTable.createServerTable(tableauContainerId,
  myColumnDefs, myDataSource, tableConfig);
```

3.3.6.3.4Filtres

L'ajout d'un formulaire de filtrage peut être effectué entre la barre d'outils et le tableau. Une icône présente dans cette barre permet d'afficher ce formulaire.

Deux boutons sont présents dans le formulaire de filtrage :

- « Filtrer » : Application des filtres au résultat de la recherche
- « Annuler » : Réinitialisation des filtres, relance de la recherche et fermeture du formulaire.

JavaScript activé : Une icône est ajoutée dans la barre d'outils ainsi qu'un bouton supplémentaire dans le formulaire pour gérer l'affichage de ce formulaire.

L'application de filtres est similaire au formulaire de recherche, le bouton « Filtrer » étant l'équivalent du bouton « Rechercher ».

Une fonction est appelée lors d'un événement « submit » sur le formulaire de filtre, qui va utiliser une Datasource XHR envoyant les données du formulaire et recharger le tableau avec les données récupérées.

Voici un exemple de fonctions appliquant les filtres sur un résultat de recherche et annulant ces filtres :

```
<s:url id="icoFiltrerUrl" value="%{#application.themeDefautPath}/assets/ico_filtrer.png"/>

<s:form id="formTableau" action="filtrage">
    <s:submit id="filterButtonDefault" cssClass="none" type="button" value="filtrer"/>

   <div class="outils haut">
      <div class="actions yui3-filtre-buttons-hd">
   </div>

<div id="filtres">
<div class="yui3-g">
    <div class="yui3-u-1-4"><s:label for="filtreNom" value="nom" /></div>
    <div class="yui3-u-1-4"><s:textfield cssClass="yui3-filtre-field" name="nom" id="filtreNom"/></div>

    <div class="yui3-u-1 button-group yui3-filtre-buttons-ft">
        <s:submit id="filterButton" type="button" value="filtrer"/>
        <s:submit id="cancelFilterButton" type="button" value="annuler" action="initFiltrage"/>
    </div>
</div>
</div>

<table>
//…
</table>
</s:form>

…
<script>
hornet().use('node', 'event', 'hornet-filtre', function(Y){
    var myDataTable, myFilter;
         …
    Y.on('domready', function(){
        var formTableNode = Y.one("#formTableau");

        // Init du composant filtre
        myFilter = new Y.hornet.filtre({
            contentBox: '#filtres',
            icon: '${icoFiltrerUrl}',
            buttonsHeader: '.yui3-filtre-buttons-hd',
            buttonsFooter: '.yui3-filtre-buttons-ft'
        });
        myFilter.render();

        // Masquage si filtre inactif
        if(! myFilter.get('active')) { myFilter.hide();  }

            // Filtre
            Y.delegate ("click", function (e) {
                if (myFilter!= null) {
                    // Mise a jour de l'etat du filtrage
                    myFilter.fire('update');
                }
                if (myDataTable != null) {
                    // Stop la propagation pour eviter le submit par le navigateur
                    e.halt();
                    e.preventDefault();

                    // Ajout des criteres de filtres
                    var myRequest = Y.IO.stringify(formTableNode.getDOMNode());

                    // Chargement des donnees
                    myDataTable.datasource.load({request: myRequest});
                }
                return false;
            }, formTableNode, "#filterButton, #filterButtonDefault ");


            // Annulation du filtre
            Y.on ("click", function (e) {
```

```
                if (myFilter!= null) {
                    // Reinit des filtres
                    myFilter.reset();
                    // Fermeture du bloc de filtrage
                    myFilter.hide ();
                }
                if (myDataTable != null) {
                    // Stop la propagation pour eviter le submit par le navigateur
                    e.halt();
                    e.preventDefault();

                    // Ajout des criteres de filtres
                    var myRequest = Y.IO.stringify(formTableNode.getDOMNode());

                    // Chargement des donnees
                    myDataTable.datasource.load({request: myRequest});
                }
                return false;
            }, "#cancelFilterButton");
    });
});
</script>
</body>
</html>
```

Remarque : Un bouton submit non visible peut être placé au début du formulaire de filtre entourant le tableau. Il permet de s'assurer que l'action par défaut de ce formulaire est l'action de filtrage. (D'autres boutons submit comme la suppression de masse peuvent en effet être présents).

De plus, JavaScript activé, il permet de distinguer l'évènement de filtrage (envoi du formulaire en cliquant sur « enter » dans un champ de filtre) avec l'envoi général du formulaire (autres actions du tableau, action d'annulation du filtrage, …).

3.3.6.3.5Export CSV

Dans l'action implémenter l'interface « ExportAware», et créer une méthode permettant de remplir l'objet TableVO comme l'exemple ci-dessous :

```
/** VO Tableau */
private TableVO exportTable;

public TableVO getExportTable() {
    return this.exportTable;
}

public String exporterCsv() {
    // Création d'un TableVO pour l'export
    // …
    return SUCCESS;
}
```

Dans le fichier de configuration Struts, associer l'action d'export Csv à la méthode et transmettre le traitement de la requête (en cas de succès) à un result de type **csv** pour construire la réponse sous forme d'un fichier CSV.

```
    <result name="success" type="csv" />
```

Dans le tableau, l'action « export CSV » peut être ajoutée avant la balise « table » avec les autres actions :

```
   <div class="outils haut">
      <div class="actions">
         <s:a cssClass="icone exporterCsv" id="exporterCsv" href="%{exportCsvUrl}"
            title="Exporter au format Csv">
            <img alt="Exporter au format Csv"
                src="${application.themeDefautPath}/assets/ico_export_csv.png"/>
```

```
            </s:a>
         </div>
      </div>

<table>
//…
</table>
```

l'url de l'action devant être définie précédement dans la page :

```
<s:url id="exportCsvUrl" value="exportCsv.csv"/>
```

3.3.6.3.6Export Excel

De la même façon que pour l'export Csv, ajouter une méthode dans l'action (implémentant l'interface « ExportAware») pour remplir l'objet TableVO.

Dans le fichier de configuration Struts, associer l'action d'export Excel à la méthode et **chainer** le traitement de la requête (en cas de succès) vers l'action d'export générique pour construire la réponse sous forme d'un fichier Excel.

```
    <result name="success" type="chain">
        <param name="actionName">exportExcelGenerique</param>
        <param name="namespace">/dyn/tech</param>
    </result>
```

Dans le tableau, l'action « export Excel » peut être ajoutée avant la balise « table » avec les autres actions :

```
   <div class="outils haut">
      <div class="actions">
         <s:a cssClass="icone exporterExcel" id="exporterExcel" href="%{exportExcelUrl}"
            title="Exporter au format Excel">
            <img alt="Exporter au format Excel"
                src="${application.themeDefautPath}/assets/ico export excel.png"/>
         </s:a>
      </div>
   </div>

<table>
//…
</table>
```

l'url de l'action devant être définie précédement dans la page :

```
<s:url id="exportExcelUrl" value="exportExcel.xls"/>
```

3.3.6.3.7Sélection de masse

Principe : Un formulaire est associé autour du tableau, et une colonne supplémentaire est ajoutée permettant de sélectionner les lignes de la page courante du tableau.

De la même façon que les autres actions « outils », les actions « en masse » peuvent être ajoutées avant/après la balise « table » dans l'élément de classe « actionsMasse » :

```
<s:form id="selectionTableau">

   <div class="outils haut">
      <div class="selection"></div>
      <div class="actions actionsMasse"></div>
      <div class="actions"></div>
   </div>

   <table>
```

```
   <thead><tr>
      <th scope="col" id="selection"> </th>

   </tr></thead>
   <tbody><s:iterator value="liste"><tr>
      <td headers="selection"><s:checkbox cssClass="hijackCheckbox" name="listeSelection"
fieldValue="%{id}" title="Sélectionner" /></td>

   </tr></s:iterator></tbody>
   </table>

   <div class="outils bas">
      <div class="selection"></div>
      <div class="actions actionsMasse"></div>
      <div class="actions"></div>
   </div>
</s:form>
```

Pour fonctionner avec le JavaScript activé, la colonne de sélection doit aussi être ajoutée dans la configuration du tableau YUI.
De plus, le flux XML retourné pour les requêtes AJAX devra ajouter les cases à cocher pour la sélection de chaque ligne.

3.3.6.3.7.1 Suppression

Dans le tableau, l'action « suppression de masse » peut être ajoutée avant/après la balise « table » :

```
   <div class="outils haut">
      <div class="selection"></div>
      <div class="actions actionsMasse">
         <span class="icone supprimer">
            <s:submit type="button" value="Supprimer" action="supprEnMasse"/></span>
      </div>
      <div class="actions"></div>
   </div>

<table>
//…
</table>
```

Pour obtenir un rendu « Hornet », remplacer la ligne dans la zone d'actions par :

```
<span class="icone supprimer">
    <s:submit type="image" src="%{#application.themeDefautPath}/assets/ico_supprimer.png"
       label="Supprimer" title="Supprimer" action="supprEnMasse"/></span>
```

3.3.6.3.8Tableau éditable

Principe : Consiste en la création d'une fenêtre modale intégrant un formulaire et d'un tableau.

Interception des événements « click » sur les actions d'édition et d'ajout du tableau, affichage de la fenêtre modale intégrant un formulaire d'édition, mise à jour du tableau avec les nouvelles données après l'envoi du formulaire.

Remarque : Pour la notification des messages serveur lors de l'envoi du formulaire d'édition (et plus généralement pour tout envoi par requête AJAX de formulaire intégré dans une fenêtre modale), il est nécessaire de préciser la configuration pour ces notifications :

- « zoneError » : la zone de notification des messages d'erreur à utiliser,
- « zoneInfo » : la zone de notification des messages d'information à utiliser,
- « titleError » : le titre pour la zone de notification des messages d'erreur,
- « titleInfo » : le titre pour la zone de notification des messages d'information.

Si aucune zone de notification d'erreur n'est précisée, la valeur donnée par défaut (« .errorBox », classe associée à toutes les zones d'erreur incluses dans les pages), correspondra probablement à la première zone de notification trouvée : les messages d'erreurs risquent alors d'être affichés en arrière-plan de la fenêtre modale.

3.3.6.4 Recherche

Principe : Similaire à la création d'un formulaire et d'un tableau avec des données locales et/ou distantes.

Au chargement de la page, construit le tableau HTML si les résultats existent.

Javascript activé :
Initialise le tableau YUI avec les données locales si le tableau HTML est présent au chargement de la page.
Interception de l'événement « submit » du formulaire de recherche, et rechargement du tableau avec les données récupérées de la requête AJAX.

Voici un exemple de page de recherche :

```
<html> …
<body>

<%-- Recherche --%>
<s:form id="recherche" action="recherche" validate="true" cssClass="formRecherche">
</s:form>

<%-- Tableau recherche --%>
<div class="table hornet-table-loading">
    <div <s:if test="(liste == null)">class="js-disabled-hidden"</s:if>>
        <h2 class="titreTableau"></h2>
        <div class="outils haut"></div>
    </div>
    <div id="tableauContainer">
    <s:if test="(liste != null)">
        <table id="tableau" summary="">
            <caption></caption>
            <thead>

            </thead>
            <tbody>
            <s:iterator value="liste" status="rowstatus">

            </s:iterator>
            </tbody>
        </table>
    </s:if>
    </div>
    <div <s:if test="(liste == null)">class="js-disabled-hidden"</s:if>>
        <div class="outils bas"></div>
    </div>
</div>

</body>
</html>
```

Remarque : Pour faciliter l'enrichissement avec le JavaScript, la zone de tableau n'est pas recréée à chaque recherche mais initialisée à la création de la page puis masquée par des styles CSS. La zone de tableau est alors masquée à chaque envoi/réinitialisation du formulaire de recherche, puis réaffichée au chargement des données du tableau de résultat.

Exemple de fonctions JavaScript pour enrichir le comportement par défaut du formulaire de recherche.

```
<script type="text/javascript">
//<![CDATA[
   // Ajout/Suppression de styles CSS pour masquer le tableau
   function hideResultatsRecherche(zoneTableau) {
   }
   function showResultatsRecherche(zoneTableau) {
   }
//]]>
</script>

<s:form id="recherche" validate="true" onsubmit="hideResultatsRecherche(".table")"
cssClass="formRecherche">
</s:form>

<script type="text/javascript">
//<![CDATA[
hornet().use("event", "node",'hornet-datatable', 'io-form', 'hornet-ajax', function (Y) {
    var myDataTable,
    myDataSource;
         …

   // Fonction appelee une fois que la page est chargee
   Y.on('domready', function(){

      // Recuperation du formulaire d'id "idForm"
      var form = Y.one("form[id='recherche']"),
      urlRecherche = form.get('action'),
      tableNode = Y.one("table[id='tableau']");

      // creation instance DataSource myDataSource
      //…

      // creation instance DataTable myDataTable
      //…
      myDataSource.xhrDatasource.after("response", function(e) {
            showResultatsRecherche(".table");
      });

      // Si tableau html n'est pas present
      if (!tableNode) {
            // Masquage du tableau
            hideResultatsRecherche(".table");
      }

      // Fonction appelee au reset du formulaire de recherche
      form.on ("reset", function(){ hideResultatsRecherche(".table"); });

      // Fonction appelee au submit du formulaire
      form.on ("submit", function (e) {
             if (myDataTable != null) {
                 // Stop la propagation pour eviter le submit par le navigateur
                 e.halt();
                 e.preventDefault();

                 // Ajout des criteres de recherche
                 var myRequest = Y.IO.stringify(this.getDOMNode());

                 // Chargement des donnees par requete AJAX
                 myDataTable.datasource.load({request: myRequest});
             }
             return false;
         });


  });

});
//]]>
</script>
```

3.3.6.5 Onglet

Principe : Tous les éléments d'onglet sont chargés dans la page, l'affichage sous forme d'onglets est ensuite réalisé si le JavaScript est activé.
Un composant d'onglet consiste en une liste de liens qui ciblent des éléments de contenu.

Utilisation du composant yui Tabview: cf. http://yuilibrary.com/yui/docs/tabview/.

Exemple de code utilisé pour créer des onglets :

```
<div id="demo" class="yui3-tabview-loading">
  <ul>
        <li><a href="#onglet1">titre 1</a></li>
        <li><a href="#onglet2">titre 2</a></li>
        <li><a href="#onglet3">titre 3</a></li>
  </ul>
  <div>
        <div id="onglet1">
            <h3 class="hornet-tabview-title-hidden">titre 1</h3>
            contenu onglet 1
        </div>
        <div id="onglet2">
            <h3 class="hornet-tabview-title-hidden">titre 2</h3>
            contenu onglet 2
        </div>
        <div id="onglet3">
            <h3 class="hornet-tabview-title-hidden">titre 3</h3>
            contenu onglet 3
        </div>
  </div>
</div>
<script>
YUI().use('tabview', function(Y) {
  var tabview = new Y.TabView({
        srcNode: '#demo'
  });
  tabview.render();

  // Séléction d'un onglet
  tabviewCadre.selectChild("0");
});
</script>
```

Note relative à l'accessibilité : chaque libellé d'onglet doit être présent sous forme de titre dans chaque zone de contenu. L'utilisation de la classe appropriée permettra de masquer ces titres au rendu du composant JavaScript.

Remarque : Si un élément de contenu contient du code JavaScript, celui-ci peut être exécuté deux fois : au chargement de la page, puis à la création des onglets.

3.3.6.6 Popup In Line

Utilisation du composant YUI 3 Panel : cf. https://yuilibrary.com/yui/docs/panel/

Par exemple, avec une page du type :

```
<html> …
<body>
…
<div id="monContainer">
  <p>Fenetre modale</p>
</div>
<a href="#" id="show-monContainer">Afficher la popin</a>


…
<script>
// Accroche YUI
YUI().use('node', 'event', 'panel',
```

```
  function(Y){
        var hijaxDialog = function() {

                // Configuration de la popin
                var myConfigDialog = {
                        modal: true, // fenetre modale
                        close: true,
                        visible: false,
                        centered: true,
                        width: "800px",
                        …
                };

                // Déclaration de la popupin
                var dialog = new Y.Panel(myConfigDialog);

                // Ajout à la page sous l'élément body
                dialog.render(document.body);

                // Fonction appelée lors de l'ouverture/fermeture de la popup in line
                dialog.on('visibleChange', function(e) {
                        …
                });

                //Ajout de l'événement  onclick sur le lien show-monContainer
                //pour afficher la popin
                Y.on("click", dialog.show, "#show-monContainer", dialog);

                …
        };
        Y.on("available", hijaxDialog, "#monContainer");
  });
</script>
</body>
```

La fonction hijaxDialog est appelée dès que le nœud « monContainer » est disponible dans la page html, et va alors initialiser une fenêtre modale avec le contenu de « monContainer ».

3.3.6.6.1Chargement différé

Utilisation du composant de la Gallerie de YUI 3 : cf. http://yuilibrary.com/gallery/show/dispatcher.

Il suffit alors de recharger le corps de la fenêtre modale, avant son affichage, à partir d'une url qui correspond au fragment de page à afficher.

Par exemple, avec une page du type :

```
<html> …
<body>
…
<div id="monContainer"></div>
<a href="modale.html?content=1" class="show-container">Afficher le contenu 1 dans la popin.</a>
<a href="modale.html?content=2" class="show-container">Afficher le contenu 2 dans la popin.</a>



…
<script>
// Accroche YUI
YUI().use('node', 'event', 'panel', 'gallery-dispatcher',
  function(Y){
        var hijaxDialog = function() {

                // Configuration de la popin
                var myConfigDialog = {
                        modal: true, // fenetre modale
                        close: true,
                        visible: false,
                        centered: true,
                        width: "800px",
                        …
                };

                // Déclaration de la popupin
                var dialog = new Y.Panel(myConfigDialog);
```

```
                // Ajout à la page sous l'élément body
                dialog.render(document.body);

                // Déclaration du composant pour le rechargement du contenu de la popin
                var myDispatcher = new Y.Dispatcher ({
                    node: dialog.get('bodyContent'),
                    normalize: true,
                    autopurge : true
                });
                myDispatcher.after('ready', function( e ) {
                    // affiche la popin apres mise a jour du contenu
                    myPanel.changeContentEvent.fire();
                    myPanel.center();
                    myPanel.show();
                });

                Y.all(".show-container").each(function(lien) {

                   //Ajout des événements  onclick sur les liens show-container
                   //pour afficher la popin avec rechargement du contenu
                    Y.on("click", function(event) {
                         // Stop la propagation de l'evenement
                         event.halt(true);
                         var url = event.currentTarget.get('href');

                         // reinitialisation du contenu
                         myDispatcher.set('content', '');
                         if (myDispatcher.get("uri") == url) {
                             // force le rechargement de l'url
                             myDispatcher._fetch(url);
                         } else {
                             // charge le contenu de l'url dans la popin
                             myDispatcher.set("uri", url);
                         }

                    }, lien);
                });


                …
        };
        Y.on("available", hijaxDialog, "#monContainer");
  });
</script>
</body>
```

3.3.6.7 *Autocomplete natif YUI*

Utilisation du composant YUI 3 Autocomplete : cf. http://yuilibrary.com/yui/docs/autocomplete/

Le composant Autocomplete de YUI permet d'afficher une liste déroulante en dessous d'un champ texte, et d'y afficher la liste des éléments « matchant » avec le texte saisi.

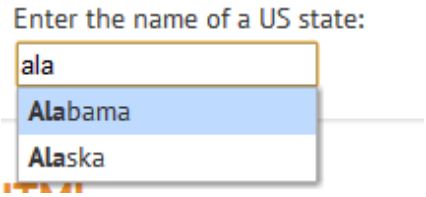


Figure 6 : Exemple de composant Autocomplete

Note relative au « Progressive Enhancement » : Javascript non activé, la fonctionnalité n'est pas disponible. Le composant devient un simple champ de saisie.

3.3.6.7.1Avec données locales

Dans ce cas, les données locales sont stockées sur le client dans un tableau JavaScript. Voici un exemple d'implémentation :

```
<div id="demo" class="yui3-skin-sam"> <!-- You need this skin class -->
  <label for="ac-input">Enter the name of a US state:</label><br>
  <input id="ac-input" type="text">
</div>

<script>
YUI().use('autocomplete', 'autocomplete-filters', 'autocomplete-highlighters', function (Y) {
  var states = [
     'Alabama',    'Alaska',    'Arizona',    'Arkansas',    'California',    'Colorado',
'Connecticut',    'Delaware',    'Florida',    'Georgia',    'Hawaii',    'Idaho',    'Illinois',
'Indiana',    'Iowa',    'Kansas',    'Kentucky',    'Louisiana',    'Maine',    'Maryland',
'Massachusetts',    'Michigan',    'Minnesota',    'Mississippi',    'Missouri',    'Montana',
'Nebraska',    'Nevada',    'New Hampshire',    'New Jersey',    'New Mexico',    'New York',    'North
Dakota',    'North Carolina',    'Ohio',    'Oklahoma',    'Oregon',    'Pennsylvania',    'Rhode Island',
'South Carolina',    'South Dakota',    'Tennessee',    'Texas',    'Utah',    'Vermont',    'Virginia',
'Washington',    'West Virginia',    'Wisconsin',    'Wyoming'
  ];

  Y.one('#ac-input').plug(Y.Plugin.AutoComplete, {
    resultFilters    : 'phraseMatch',
    resultHighlighter: 'phraseMatch',
    source           : states
  });
});
</script>
```

Explication :

Ici le composant est connecté (« *pluggé* ») sur la source locale, à savoir le tableau JavaScript « states ». Le paramètre "resultFilter" va permettre de filtrer parmi cette source locale. L'import de la librairie « autocomplete-filters » doit être déclaré.

Cette utilisation est adaptée par exemple dans le cas d'une liste d'éléments de taille modeste.

Les paramètres de la méthode "plug" seront expliqués aux §3.2.6.7.2 et §3.2.6.7.3.

3.3.6.7.2Avec données distantes

Dans ce cas, les données proviennent du serveur au format JSON.
Le composant est « pluggé » sur la source distance. Il n'y a pas de filtre coté client. Les données sont filtrées coté serveur, soit en SQL soit en Java. L'exemple qui suit filtre les données en Java.

3.3.6.7.2.1 Implémentation coté client

```
<div id="demo" class="yui3-skin-sam"> <!-- You need this skin class -->
  <label for="ac-input">prenom</label><br>
  <input id="ac-input" type="text">
</div>


<script>
// Create a new YUI instance and populate it with the required modules.
YUI().use('autocomplete','autocomplete-highlighters', function (Y) {

      // XHR URL source (no callback). Leave the {query} placeholder
      // as is; AutoComplete will replace it automatically.
      Y.one('#ac-input').plug(Y.Plugin.AutoComplete, {
          resultHighlighter: 'startsWith',
          resultListLocator: 'data.liste',
          source: 'http://localhost:8080/hornettemplate/dyn/protected/json/actionJson.html?query={query}'

      });

});
```

```
</script>
```

La chaine {query} va être remplacée par le framework par la donnée saisie dans le champ texte de l'autocomplete. La requête va ensuite être exécutée pour récupérer les données de l'autocomplete.

> ➲ **Attention la classe « yui3-skin-sam » est très importante, c'est cette classe qui applique le style de l'autocomplete.**

3.3.6.7.2.2 Implémentation coté serveur avec Struts2

L'utilisation de JSON dans Struts2 implique une dépendance vers "struts2-json-plugin" :

<dependency org="org.apache.struts" name="struts2-json-plugin" rev="2.3.16.3"/>

Une action doit être déclarée de cette façon dans le struts.xml

```
<package name="struts-hornet-hornettemplate" extends="struts-hornet-hornetserver">
   <result-types>
           <result-type name="json" class="org.apache.struts2.json.JSONResult" />
   </result-types>
...
</package>


<package name="json" namespace="/dyn/protected/json" extends="struts-hornet-hornettemplate">
        <action name="actionJson"
            class="fr.gouv.diplomatie.hornettemplate.web.action.ActionJSON">
            <result type="json"></result>
        </action>
</package>
```

L'action doit posséder un attribut de type Map qui contient la liste des suggestions à retourner.
L'action devra aussi prendre en compte le paramètre "query" envoyé depuis le client.

Voici un exemple complet de cette classe :

```
package fr.gouv.diplomatie.hornettemplate.web.action;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.struts2.json.annotations.JSON;

import com.opensymphony.xwork2.Action;

public class ActionJSON {

    /**
     *
     */
    private static final long serialVersionUID = 1861490789092363378L;

    /**
     *
     */
    private Map<String, List<String>> jsonData =
            new HashMap<String, List<String>>();

    /**
     *
     */
    private String query;
```

```
    /**
     * Constructeur
     */
    public ActionJSON() {

        int i = 0;
        i++;
    }

    /**
     * Méthode d'exécution de l'action
     *
     * @return success
     */
    public String execute() {

        final List<String> listeFull = new ArrayList<String>();
        listeFull.add("michel");
        listeFull.add("martin");
        listeFull.add("simon");
        final List<String> listeFiltered = new ArrayList<String>();
        for (final String string : listeFull) {
            if (string.startsWith(this.query)) {
                listeFiltered.add(string);
            }
        }
        this.jsonData.put(
            "liste", listeFiltered);

        return Action.SUCCESS;
    }

    /**
     *
     * @return
     */
    @JSON(name = "data")
    public Map<String, List<String>> getJsonData() {

        return this.jsonData;
    }

    /**
     *
     * @param jsonData
     */
    public void setJsonData(
            final Map<String, List<String>> jsonData) {

        this.jsonData = jsonData;
    }

    /**
     *
     * @param query
     */
    public void setQuery(
            final String query) {

        this.query = query;
    }
}
```

Notons que l'annotation @JSON(name="data") permet de changer le nom de la clé, "data" au lieu de "jsonData".

Ici le résultat JSON (non filtré) envoyé au client sera :

```
{data: {liste:{michel,martin,simon}}}
```

C'est pourquoi, côté client, le paramètre "resultListLocator" doit être positionné sur "data.liste".

3.3.6.7.3Options de recherche

3.3.6.7.3.1 ResultFilters : Résulats de la recherche

Le paramètre **resultFilters** permet de préciser le type de recherche sur la liste de l'autocomplete. Cela est utile dans le cas ou les données sont rapatriées localement.

Plusieurs options sont possibles :

- Soit la recherche se fait dans tout le mot. Par exemple la saisie (en reprenant l'exemple des données locales) de « **al** » remonte les résultats :
  - **Alabama**
  - **Alaska**
  - **California**

  Dans ce cas, l'option « **resultFilters** » doit être « **phraseMatch** »
- Soit la recherche doit se faire sur le début du mot. Dans ce cas « **al** » remonte les résultats :
  - **Alabama**
  - **Alaska**

  Dans ce cas, l'option « **resultFilters** » doit être « **startsWith** »

Voici les autres options possibles :

- **charMatch** : retourne les résultats qui contiennent les caractères de la requête qui se produisent n'importe où dans les résultats, dans n'importe quel ordre (pas nécessairement consécutif).
  Par exemple, la saisie de « **x** » retourna les résultats « **New Mexico** » et « **Texas** »
- **subWordMatch** : retourne les résultats dans lequel tous les mots de la correspondance de requête (mots entiers ou parties de mots) dans le résultat. Des mots comme des espaces et certains signes de ponctuation sont ignorés.
  Par exemple « **Je** » retournera « **New Jersey** »
- **wordMatch :** retourne les résultats qui contiennent tous les mots dans la requête, dans n'importe quel ordre (pas nécessairement consécutif).
  Par exemple il faudra saisir le mot « **Jersey** » pour voir afficher dans la liste « **New Jersey** »

3.3.6.7.3.2 ResultHighlighter : Mise en surbrillance des résultats de recherche

De la même façon que **resultFilters**, le paramètre **resultHighlighter** permet de modifier la mise en surbrillance des résultats (filtrés ou non), les mêmes options que la recherche sont possibles.

3.3.6.7.3.3 MaxResults

Cette option permet de limiter le nombre de résultat affichés dans liste.

3.3.6.7.4Action sur la sélection d'un élément

Une fois l'autocomplete plugué sur le nœud, on peut ajouter au nœud des événements parmi lesquels :
« select ». Cet « event » permet d'intercepter le click de la souris (ou la sélection par clavier) sur un élément.

On accède à l'instance de l'autocomplete en appelant la propriété « ac » du nœud.

Voici un exemple de code permettant d'afficher un message d'alerte en cas de click sur un élément.

```
<div id="demo" class="yui3-skin-sam">
  <label for="ac-input">prenom</label><br>
  <input id="ac-input" type="text">
</div>


<script>
// Create a new YUI instance and populate it with the required modules.
YUI().use('autocomplete','autocomplete-highlighters', function (Y) {

      // XHR URL source (no callback). Leave the {query} placeholder
      // as is; AutoComplete will replace it automatically.
      Y.one('#ac-input').plug(Y.Plugin.AutoComplete, {
          resultHighlighter: 'startsWith',
          resultListLocator: 'data.liste',
           maxResult: 50,
          source: 'http://localhost:8080/hornettemplate/dyn/protected/json/actionJson.html?query={query}'

      });

    Y.one("#ac-input").ac.after('select', function (e) {
           alert("click ! ");
   });

});
</script>
```

Cf http://yuilibrary.com/yui/docs/autocomplete/#list-events pour plus d'options

3.3.6.8 Autocomplete Hornet

Depuis Hornet 3.3, un composant Hornet gérant l'autocomplétion a été ajouté.

3.3.6.8.1 Tag hornet:autocomplete

Utilisation du tag Hornet « autocomplete » pour générer les éléments HTML et le code JavaScript associé.

Exemple :

```
<fieldset class="inline yui3-g formmgr-row">
    <s:component template="legend" value="Nationalité" cssStyle="label" requiredLabel="true"
        cssClass="yui3-u-1-4" />
    <div class="yui3-u-3-4">
        <div class="formmgr-message-text"></div>


        <hornet:autocomplete id="nationalite" cssClass="formmgr-field"

            inputLabel="Aide à la saisie d'une nationalité"
            buttonLabel="Rechercher les nationalités"
            selectLabel="Sélection d'une nationalité"

            inputName="nationalite.libelle"
            selectName="nationalite.id"

            iconSrc="%{#icoConsulterUrl}"
            action="reloadNationaliteEdition"

            list="listeNation"
            listKey="id"
            listValue="libelle"

            queryDelay="500"
            minQueryLength="3"
            maxResults="0"
            resultHighlighter="phraseMatch"

            url="reloadNationaliteEdition.json?mode=JSON"
            errorsAreaTitle="%{#titleError}"
        />


    </div>
```

```
</fieldset>
```

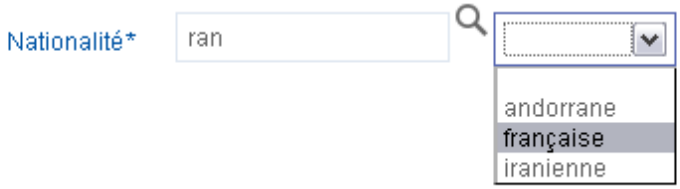


Figure 7 : Exemple de composant Hornet-Autocomplete sans JavaScript

Un champ texte d'aide à la saisie est associé à une liste déroulante contenant les suggestions possibles à sélectionner. Un bouton supplémentaire permet de recharger la page pour mettre à jour la liste des valeurs sélectionables en fonction du texte saisi.

Figure 8 : Exemple de composant Hornet-Autocomplete avec JavaScript

JavaScript activé, seul le champ texte sera visible : la liste des éléments correspondant avec le texte saisi sera affichée sous forme de liste déroulante sous le champ et gérée par JavaScript. La sélection d'un élément de la liste sera mise à jour dans un champ caché adjacent.

On retrouve les paramètres suivant :

Paramètre	Description	Obligatoire	Valeur par défaut
id	Identifiant HTML à associer à l'élément conteneur. Ceux des sous-éléments (champ text, bouton et liste déroulante) seront suffixés respectivement par : - « _autocomplete_input », - « _autocomplete_button », - « _autocomplete_select ».	oui	
cssClass	Classes CSS à ajouter sur l'élément conteneur		
inputLabel	Libellé pour le champ de texte d'aide à la saisie. Utilisé comme titre.		
buttonLabel	Libellé pour le bouton submit. Utilisé comme titre, valeur du bouton et alternative textuelle de l'image.		
selectLabel	Libellé pour la liste déroulante de sélection. Utilisé comme titre.		

inputName	Nom du champ de texte et valeur de saisie. Utilisé comme paramètre de la requête JavaScript activé.	oui	
selectName	Nom de la liste déroulante et valeur de sélection. Utilisé pour le champ caché JavaScript activé.	oui	
list	Nom de la liste contenant les données à afficher dans la liste déroulante.	oui	
listKey	Nom de la propriété des données à utiliser comme valeur dans la liste déroulante.		key
listValue	Nom de la propriété des données à utiliser comme libellé dans la liste déroulante.		value
JavaScript désactivé uniquement			
iconSrc	Source de l'image du bouton submit.		
action	Action à appeler pour la mise à jour de la liste de données.		
JavaScript activé uniquement			
queryDelay	Délai à attendre après chaque saisie avant lancement d'une requête (en milliseconde).		500
minQueryLength	Nombre de caractères minimum avant lancement d'une requête.		3
maxResults	Nombre maximum de suggestion à afficher dans la liste déroulante (« 0 » pour ne pas filtrer).		0
resultHighlighter	Pour mettre en surbrillante les éléments de la liste déroulante. Les options possibles sont : - « charMatch », - « phraseMatch », - « startsWith », - « subWordMatch », - « wordMatch ».		phraseMatch
url	Url à appeler par requête Ajax contenant les données pour mettre à jour la liste déroulante.	oui	
errorsAreaTitle	Titre à afficher dans la zone d'erreur si des erreurs se produisent durant la requête.		

3.3.6.8.2Chargement Ajax

JavaScript activé, la liste déroulante est alimentée par requêtes Ajax-json. La requête est créé à partir du paramètre « url » du tag hornet :autocomplete et attend un flux JSON en retour.

La valeur saisie est envoyée comme paramètre et associée avec le nom du champ (paramètre « inputName »).

Les flux JSON de sortie doivent respecter un format défini :

- la liste de données sérialisée doit être regroupées sous la clé « data »,
- chaque objet sérialisé de la liste doit posséder les propriétés définies par les paramètres « listeKey » et « listeValue ».

Voici un exemple de flux attendu :

```
{
  "data" : [
      {"id":1,"libelle":"andorrane"},
      {"id":2,"libelle":"française"},
      {"id":3,"libelle":"iranienne"}
    ]
  },
  "fielderrors" : {},
  "errors" : [],
  "messages" : []
}
```

Attention : dans le cas de données volumineuses, il est recommandé de filtrer la liste de données au niveau de la requête, coté serveur. En effet, le paramêtre « maxResults » permet uniquement de filtrer l'affichage du nombre de résultats mais est aussi envoyé dans la requête Ajax. A charge du serveur de limiter le nombre de résultat pour ne pas renvoyer la liste complète dans le flux JSON.

3.3.6.9 Arborescence

Utilisation du composant JavaScript Hornet Arborescence basé sur le composant Flyweight TreeView issu de la Gallerie de YUI 3 : cf. http://yuilibrary.com/gallery/show/fwt-treeview.

Le composant est construit à partir du code HTML existant structuré sous la forme de liste imbriquée de ul/li, ou à partir d'un tableau d'objet JavaScript passé en paramètre.

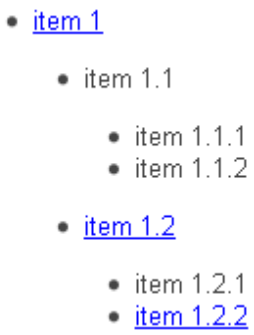


Figure 9 : Exemple d'arborescence sans JavaScript

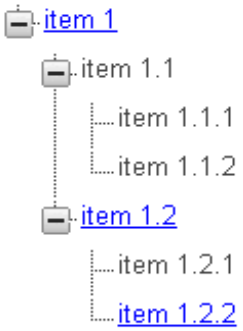


Figure 10 : Exemple d'arborescence avec JavaScript

Le cas d'utilisation classique consiste à générer entièrement l'arborescence au chargement de la page et améliorer le rendu avec le composant JavaScript :

- JavaScript désactivé : l'intégralité de l'arbre est affichée ;
- JavaScript activé : l'arborescence est entièrement chargée mais les sous-niveaux de l'arborescence sont cachés par défaut.

Cependant, cette utilisation n'est pas recommandée lorsque les données sont trop volumineuses et impacteraient fortement les performances réseau. Dans ce cas de figure, l'utilisation du chargement dynamique est préférable :

- JavaScript activé : seul une partie de l'arborescence est présente au chargement de la page, les sous-éléments seront récupérés par requêtes AJAX au moment voulu.

3.3.6.9.1 Génération de l'arborescence complète

Utilisation du tag Hornet « Tree » pour générer récursivement l'arborescence complète en HTML. Le composant JavaScript sera initialisé à partir du code existant.

3.3.6.9.1.1 Génération du code

A fournir :

- Id : L'id du composant arborescence,
- rootNode : Le nom de l'objet définissant l'arborescence complète,
- childCollectionProperty : Le nom de la propriété pour récupérer la liste des sous-éléments,
- nodeIdProperty : Le nom de la propriété pour récupérer les identifiants à associer aux éléments,
- nodeTitleProperty : Le nom de la propriété pour récupérer le libellé à associer aux éléments.

Optionnels :

- nodeHrefProperty : Le nom de la propriété pour récupérer l'url à appeler lors du clic sur un élément,

- name : Le nom à associer au composant arborescence pour initialiser l'état d'affichage : liste correspondant aux identifiants des éléments à déplier,
- jsTreeEnabled : La génération ou non du composant JavaScript.

```
<hornet:tree cssClass="yui3-arborescence-loading"
        id="folder"

        rootNode="tree"
        childCollectionProperty="children"
        nodeIdProperty="id"
        nodeTitleProperty="title"

        nodeHrefProperty="url"
        name="treeState"
        jsTreeEnabled="true"
/>
```

Remarque : la classe fournie dans le paramètre « cssClass » permet de masquer temporairement le rendu le temps du chargement lorsque le composant JavaScript est utilisé.

Le paramètre « jsTreeEnabled » permet d'activer la génération du composant JavaScript par la taglib, le code généré est de la forme :

```
<script type="text/javascript">
YUI().use('node', 'event', 'hornet-arborescence', function(Y) {


    var treeContainerId = "folder",
        arborescence;

    Y.on('domready', function(){
        var treeContainer = Y.one("*[id='" + treeContainerId + "']");
        if (treeContainer) {
            var treeMarkup = treeContainer.one("ul");
            if (treeMarkup) {
                arborescence = new Y.hornet.Arborescence({
                    sourceNode: treeMarkup,
                    boundingBox: treeContainer
                }).render();
            }
        }
    });
</script>
```

3.3.6.9.1.2 Récupération des données

3.3.6.9.1.2.1 Création d'une arborescence

Pour pouvoir utiliser le tag Hornet « tree », l'objet racine contenant les données d'une arborescence doit être structuré de façon récursive : chaque élément de l'arborescence est basé sur une interface commune qui défini un certain nombre de propriétés et une liste de sous-éléments.

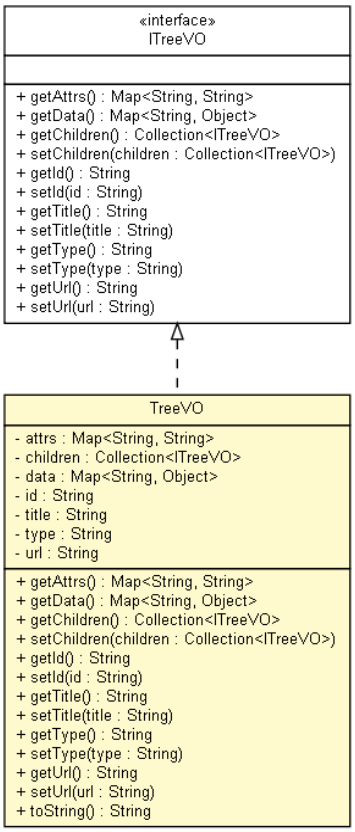


Figure 11 : Objet générique « TreeVO » pour définir une arborescence

Le Tag Hornet « tree » permet d'utiliser n'importe quel objet pour définir l'arborescence à condition que tous les objets de l'arborescence respectent la même structure récursive (Cf. Objet « TreeVO » du Framework). Il suffit alors de préciser le nom de chaque propriété dans les attributs du Tag.

3.3.6.9.1.2.2 Création d'une arborescence complexe

Pour créer une arborescence plus complexe avec des données provenant d'objets métier, l'utilisation d'une fabrique et de classes spécialisées sera préférable pour construire cette structure imbriquée d'objets.

La construction sera faite en deux étapes :

- Construction d'une arborescence d'objets encapsulant des données métiers,
- Conversion de cette arborescence métier en objets génériques à l'aide de classes gérant la transformation suivant le type de l'élément.

Pour chaque objet métier, il faudra créer une classe implémentant « ITreeNode » qui permettra d'encapsuler l'objet, et un builder associé qui se chargera de créer l'objet final.

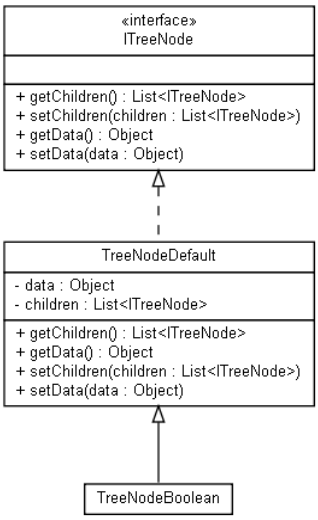


Figure 12 : Objets d'arborescence encapsulant des données métier

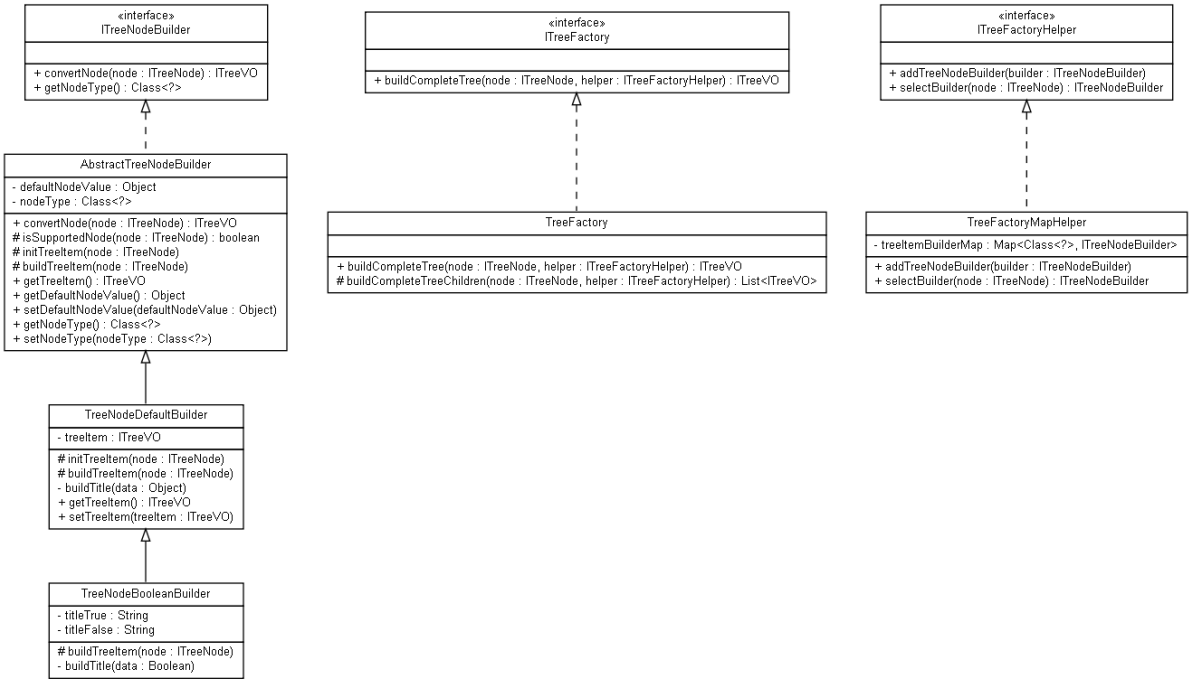


Figure 13 : Fabrique et builders pour créer une arborescence complexe

L'arborescence finale est construite en appelant la fabrique avec l'arborescence d'objets métier, et un helper qui contiendra la liste des différents builders à utiliser.

Exemple d'implémentation d'une action pour l'affichage d'une arborescence :

```
    // Récupération de la hiérarchie d'objets métier depuis le service
    ITreeNode treeNode =
            this.partenaireService.listerArborescenceOrganismePartenaire();

    // Création de Builder chargés de générer les VO de présentation
    final ITreeNodeBuilder builderOrganisme =
            new TreeNodeDefaultBuilder(
                    TreeNodeOrganismeVip.class,
                    this.getText("ARBORESCENCE_PARTENAIRE.sans_organisme"));
    final ITreeNodeBuilder builderVip =
            new TreeNodeBooleanBuilder(
                    this.getText("ARBORESCENCE_PARTENAIRE.vip"),
                     this.getText("ARBORESCENCE_PARTENAIRE.nonvip"));
```

```
    final ITreeNodeBuilder builderPartenaire =
            new TreeNodePartenaireBuilder(
                    "/dyn/protected/partenaire/initFichePartenaire.html?retour=arborescence",
                    "idPartenaire");

    // Helper pour gerer les builders
    ITreeFactoryHelper helper = new TreeFactoryMapHelper();
    helper.addTreeNodeBuilder(builderOrganisme);
    helper.addTreeNodeBuilder(builderVip);
    helper.addTreeNodeBuilder(builderPartenaire);

    // Construction de l'arborescence finale
    this.arborescencePartenaire = new TreeFactory().buildCompleteTree(treeNode, helper);

    return SUCCESS;
```

Dans la JSP, le composant chargé de générer le rendu de l'arbre est déclaré de la manière suivante :

```
<hornet:tree cssClass="yui3-arborescence-loading"
  id="folder" name="treeState"
  jsTreeEnabled="true"

  rootNode="arborescencePartenaire"
  childCollectionProperty="children"
  nodeIdProperty="id"
  nodeTitleProperty="title"
  nodeHrefProperty="url"
/>
```

3.3.6.9.2Chargement dynamique avec AJAX

Le composant JavaScript Hornet Arborescence gère aussi le chargement des données de façon dynamique avec des requêtes AJAX.

Remarque : ce mode d'utilisation n'est valable que lorsque JavaScript est activé, il n'y a pas d'alternative mise en place si JavaScript est désactivé.

Un paramètre « dynamicLoading » permet de définir une fonction qui sera appelée la première fois qu'un nœud est déplié dans l'arborescence. Elle sera chargée de récupérer les données des sous-éléments du nœud courant sous forme d'un tableau d'objets ayant les propriétés suivantes :

Nom	Type	Obligatoire	Valeur par défaut	Description
id	String	oui		Identifiant à assigner à l'élément DOM contenant le nœud et utilisé pour le chargement dynamique de ses sous-éléments.
label	String	oui		Libellé à utiliser pour être affiché dans le nœud.
url	String	non		URL à utiliser pour générer un lien dans le nœud.
isLeaf	Boolean	non	false	Si le nœud n'a pas de sous-éléments.
type	FWTreeNode / String	non	hornet.ArborescenceNode	Type à utiliser pour créer les sous-éléments du nœud.

Par exemple, le code suivant permet d'afficher le premier niveau d'une arborescence, les sous-éléments seront chargés dynamiquement, à l'aide d'une requête AJAX, lors du dépliement d'un nœud :

```
<s:url id="loadTreeUrl" action="…">
        <s:param name="mode">XML</s:param>
</s:url>

<hornet:tree id="treeItems" cssClass="yui3-arborescence-loading"
```

```
        jsTreeEnabled="false"

        rootNode="…"
        childCollectionProperty="…"
        nodeIdProperty="…"
        nodeTitleProperty="…"
/>

<script type="text/javascript">
hornet().use('node', 'event', 'hornet-arborescence', 'hornet-ajax', function(Y) {

        var treeId = "treeItems",
                arborescence, dynamicLoader;

        Y.on('domready', function(){
                var treeContainer = Y.one("*[id='" + treeId + "']");
                if (treeContainer) {

                        arborescence = new Y.hornet.Arborescence({
                                boundingBox: treeContainer,
                                dynamicLoader: dynamicLoader
                        });

                        var treeMarkup = treeContainer.one("ul");
                        if (treeMarkup) {

                                arborescence.load(treeMarkup);
                        }

                        arborescence.render();
                }
        });

        dynamicLoader = function (treeNode, callback) {

                var updateTree = function (id, response, args) {
                        var data, tree,

                        // structure du XML attendue
                        schema_data = {
                                resultListLocator: "item",
                                resultFields:  [{key:"id"}, {key:"label"}, {key:"url"}, {key:"hasChildren"}]
                        };
                        try {
                                data = Y.DataSchema.XML.apply(schema_data, response.responseXML);
                                tree = [];
                        }
                        catch(e) {
                                Y.log("Error while parsing XML data: " + e.message, "error");
                        }

                        // initialisation des sous-elements a ajouter
                        Y.Array.each(data.results, function (entry) {
                                tree.push({
                                        id: entry.id,
                                        label: entry.label,
                                        url: entry.url,
                                        isLeaf: (entry.hasChildren === 'false'),
                                        type: Y.hornet.ArborescenceNode
                                });
                        });

                        // appel du callback
                        callback(tree);
                };

                // differents comportements sont possibles suivant le type du noeud
                if (treeNode instanceof Y.hornet.ArborescenceNode) {

                        // initialisation des parametres a envoyer a partir des proprietes du noeud
                        var params = Y.Lang.sub('parentId={id}', treeNode.getAttrs()),

                        // url a appeler
                        url = '<s:property value="loadTreeUrl" />';

                        // recuperation des donnees par requete AJAX et appel du callback au retour
                        Y.hornet.Ajax.submitAJAXRequest(
                                params, {
                                url : url,
                                onSuccess : updateTree,
```

```
                                onFailure : function(){
                                        callback();
                                }
                        });
                } else {
                        callback();
                }
        };
});
</script>
```

3.3.6.10 Google Maps Pop-in

Note : Depuis la version 3.5 d'Hornet, ce composant n'utilise plus pour son implémentation de composants YUI2 ; il est donc 100% compatible YUI3.

Utilisation du composant hornet-googlemaps-overlay.

Le composant permet d'afficher une carte google maps dans une fenêtre modale et de sélectionner une adresse à partir d'un pointeur sur la carte.

Par exemple, avec une page du type :

```
<html> …
<head>
</head>
<body>
…
        <a id="btnCarte" href="#">Afficher carte de la France</a><div id="container"></div>
…
<script>
// Accroche YUI
YUI().use('node', 'hornet-googlemaps-overlay', function(Y){
  …
```

```
  // Création du composant

        var map = new Y.hornet.googlemapsOverlay().render("#container");

        Y.on('click', function(){
            map.openAdresse("France");
        },'#btnCarte');
});
</script>
</body>
</html>
```

Il peut être utilisé comme assistant pour la saisie des coordonnées GPS de lieux de destination. Par exemple :

```
<html> …
<head>
</head>
<body>
…

        <button type="submit" id="btnAdresse">ouvrir (Adresse)</button>
        <button type="submit" id="btnPositionGps">ouvrir (Position Gps)</button>
        <fieldset>
            <legend>Localisation</legend>
                <div>
                    <label for="adresse">Adresse</label>
                    <input type="text" id="adresse" name="adresse" value="" />
                </div>
                <div>
                    <label for="latitude">Latitude</label>
                    <input type="text" id="latitude" name="latitude" value="" />
                </div>
                <div>
                    <label for="adresse">Longitude</label>
                    <input type="text" id="longitude" name="longitude" value="" />
                </div>
        </fieldset>
        <div id="container"></div>
…
<script>
// Accroche YUI
YUI().use('node', 'hornet-googlemaps-overlay', function(Y){
  …

  // Création du composant

        var map = new Y.hornet.googlemapsOverlay({
            height: '400px',
            width: '500px'
        }).render("#container");

  // Gestion des erreurs et alertes

        map.on('failure', function(e){
            alert(e.msg);
            map.hide();
        });
        map.on('info', function(e){
            if(!confirm(e.msg)) {
                map.hide();
            }
        });

  // Association du composant avec des champs de saisie

        var adresse = Y.one('#adresse'),
            latitude = Y.one('#latitude'),
            longitude = Y.one('#longitude');

        Y.on('click', function(e){
            e.preventDefault();
            map.openAdresse(adresse.get('value'));
        },'#btnAdresse');

        Y.on('click', function(e){
            e.preventDefault();
```

```
            map.openPositionGps(latitude.get('value'), longitude.get('value'));
        },'#btnPositionGps');

        map.on('select', function(e){
            latitude.set('value', e.latitude);
            longitude.set('value', e.longitude);
            adresse.set('value', e.adresse);
        });
});
</script>
</body>
</html>
```

3.3.7 Tiles

Le framework Tiles implémente le pattern « Composite View » : des modèles de pages extensibles et génériques permettent de factoriser le rendu HTML, chaque page étend le modèle voulu et seule la partie variante (le body) est à écrire.

Tiles peut être vu comme l'équivalent des frames côté serveur.

3.3.7.1 Mise en place

Dans le fichier « web.xml », le code suivant permet d'ajouter le HornetTilesListener afin d'intercepter correctement les requêtes pour Tiles :

```
<listener>
    <listener-class>hornet.framework.web.listener.HornetTilesListener</listener-class>
</listener>
```

Tous les fichiers correspondants aux règles suivantes seront automatiquement chargés :

- `/WEB-INF/**/tiles*.xml` du WAR
- `/META-INF/**/tiles*.xml` des JAR du classpath de l'application

Dans le fichier « struts.xml », pour chaque action qui fait appel à Tiles, il suffit d'utiliser le package Tiles comme ceci :

```
<package name="layout" namespace="/dyn/protected/layout"
  extends="struts-hornet-hornettemplate">
  <action name="LayoutInternet"
  class="fr.gouv.diplomatie.hornettemplate.web.action.frameset.Layout">
        <result name="success" type="tiles">layoutInternet</result>
  </action>

  <action name="LayoutIntranet"
  class="fr.gouv.diplomatie.hornettemplate.web.action.frameset.Layout">
        <result name="success" type="tiles">layoutIntranet</result>
  </action>
</package>
```

3.3.7.2 Déclaration des Layout

Le fichier « tiles.xml » permet de définir les différents Layout appliqué, comme ceci :

```
    <definition name="baseLayout"
  template="/WEB-INF/tiles-jsp/layout/baseLayout.jsp">
        <put-attribute name="title" value="baseLayout" />
        <put-attribute name="header"
            value="/WEB-INF/tiles-jsp/frameset/hd.jsp" />
        <put-attribute name="menu"
            value="/WEB-INF/tiles-jsp/frameset/nav.jsp" />
        <put-attribute name="footer"
            value="/WEB-INF/tiles-jsp/frameset/ft.jsp" />
    </definition>
```

Le layout « BaseLayout » est le layout de base : il contient la déclaration de toutes les parties de pages communes (header, menu, footer). Il s'appuie sur la JSP « baseLayout.jsp » et permet de déclarer les JSP utilisées pour les parties de type header, menu et footer.

```
    <definition name="layoutInternet" extends="baseLayout">
        <put-attribute name="title" value="layoutInternet" />
        <put-attribute name="path" value="" />
```

```
        <put-attribute name="content"
         value="/WEB-INF/tiles-jsp/layout/layoutInternet.jsp" />
    </definition>
```

Ensuite, le layout « layoutInternet » étend le layout de base mais implémente la page « layoutInternet.jsp » en temps que page de contenu « content ». Par le principe d'héritage, on trouve pour ce layout le header, le menu et le footer déclarés dans le layout de base.

```
    <definition name="layoutIntranet" extends="baseLayout">
        <put-attribute name="title" value="layoutIntranet" />
        <put-attribute name="path" value="" />
        <put-attribute name="content"
         value="/WEB-INF/tiles-jsp/layout/layoutIntranet.jsp" />
    </definition>
```

On trouve aussi le layout « layoutIntranet » qui de la même manière étend le layout de base mais implémente une page « content » différente.

Le contenu de la JSP « baseLayout.jsp » est le suivant :

```
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<html>
<head>
  <title><tiles:getAsString name="title" /></title>
</head>
<body>
  <tiles:insertAttribute name="header"/>
  <table>
        <tr>
              <td>
                    <tiles:insertAttribute name="menu"/>
              </td>
              <td>
                    <tiles:insertAttribute name="content"/>
              </td>
        </tr>
  </table>
  <tiles:insertAttribute name="footer"/>
</body>
</html>
```

Une balise « tiles:insertAttribute » permet d'insérer une page au niveau où elle est placée.

Pour chaque layout qui utilise « baseLayout.jsp », seul le contenu de « content » sera différent. Le reste des parties de la page (header, menu et footer) sera identique.

3.3.7.3 Wildcard

Tiles 3 permet l'utilisation de wildcards (« * »), ainsi que d'expressions régulières, dans les définitions des tiles. Cela permet de factoriser le code de définition de plusieurs pages dont la construction est cimilaire.

Par exemple on pourra utiliser les définitions suivantes pour toutes les pages de premier niveau hiérachique :

```
  <!-- niveau 1 (exemple : contact.tile) -->
  <definition name="WILDCARD:*.tile.error" extends="baseLayout">
        <put-attribute name="pageErreur" value="true" />
        <put-attribute name="filArianeKey" value="menu.{1}" />
        <put-attribute name="content" value=" {1}.content" />
  </definition>
  <definition name="WILDCARD:*.tile" extends="baseLayout">
        <put-attribute name="filArianeKey" value="menu.{1}" />
        <put-attribute name="content" value="{1}.content" />
  </definition>
  <definition name="WILDCARD:*.content" template="/WEB-INF/tiles-jsp/{1}/{1}.jsp" />
```

On note également la possibilité de référencer les éléments correspondants aux wildcards via la notation `{x}` (avec x = 1, 2, etc. pour chaque wildcard utilisée)

Ainsi, avec l'exemple précédent, pour une action Struts défini avec le résultat suivant :

```
        <result name="success" type="tiles">contact.tile</result>
```

On a :

- filArianeKey : `menu.contact`

- content : `/WEB-INF/tiles-jsp/contact/contact.jsp`

Afin de maximiser le bénéfice de l'utilisation des wildcards, il faut s'attacher à uniformiser le nommage des tiles ainsi que l'organisation des JSP et ainsi faciliter la factorisation des définitions de tiles.

Il faut noter qu'il n'est pas possibile d'étendre une défintion créée de cette façon, l'exemple suivant ne fonctionne pas :

```
    <definition name="WILDCARD:*.tile" template="/WEB-INF/tiles-jsp/{1}/{1}.jsp">
        <put-attribute name="attr1" value="{1}" />
    </definition>
    <definition name="autre.tile" extends="autre.tile">
        <put-attribute name="attr2" value="valeur attribut 2" />
    </definition>
```

> ➲ **Pour utiliser le mode « expession régulière », il sufit d'utiliser le préfixe « REGEXP » à la place de « WILDCARD ».**

3.3.8 Prise en charge des flux JSON

Hornetserver repose sur le framework Jackson (https://github.com/FasterXML/jackson-docs) pour la génération des flux JSON.

3.3.8.1 Résultat JSON

L'utilisation du JSON comme sortie d'une action se traduit par l'utilisation d'un résultat Struts de type « Jackson » :

```
    <package name="fr.gouv.diplomatie.applitutoriel.web.action"
namespace="/dyn/protected/test"
        extends="struts-hornet-applitutoriel">
        <action name="jsontest"
        class="fr.gouv.diplomatie.applitutoriel.web.action.test.JsonTestAction" method="list">
            <result name="success" type="jackson">
                <param name="root">partenaires</param>
                <param name="indent">true</param>
                <param name="excludeNullProperties">true</param>
            </result>
        </action>
    </package>
```

Associé à ce type de résultat, on retrouve trois paramètres :

Paramètre	Description	Valeur par défaut
root	Indique de ne sérialiser qu'une propriété de l'action en particulier. La valeur renseignée est une expression OGNL (ex : action.sousobjet.propriete).	Si non renseigné, toutes les propriétés de l'action sont sérialisées.
indent	Permet l'indentation du flux JSON pour une meilleure lisibilité. Augmente la taille du flux avec des espaces non significatifs. A n'utiliser qu'en développement !	false
excludeNullProperties	Permet d'exclure du flux généré les propriétés nulles sur toute l'arborescence d'objets à sérialiser. Allège le flux JSON.	false

3.3.8.2 Configuration du flux par annotations

Le framework Jackson offre une série d'annotations permettant de jouer sur le contenu du flux JSON de sortie :

- @JsonIgnore : Permet de filtrer les propriétés ne devant pas être prise en compte dans le flux JSON.

```
/** Propriété à ne pas sérialiser */
@JsonIgnore
private String garbage;
```

- @JsonProperty : Permet d'attribuer un nom de clé JSON différent de celui de la propriété de l'objet à sérialiser.

```
/** Propriété renommée */
@JsonProperty("title")
private String titre;
```

- @JsonInclude : Sert à indiquer si la propriété annotée doit être inclue selon sa valeur (NON_NULL, NON_EMPTY, NON_DEFAULT). Dans l'exemple suivant, si la valeur de « partenaire » est nulle, la propriété n'apparaitra pas en sortie.

```
/** Liste */
@JsonInclude(Include.NON_NULL)
private List<Partenaire> partenaires;
```

> ➲ **Les attributs de l'action marqués comme « transient » sont sérialisés s'ils possèdent un getter public.**
> **Pour pallier à ce comportement par défaut, il faut ajouter d'une annotation « @JsonIgnore » sur le getter associé.**

3.3.8.3 Format des flux de sortie

Afin de restituer au client les éventuels messages et erreurs remontés par l'action, un format standard de flux a été mis en place. Voici un exemple (excluant les propriétés nulles) :

```
{
  "data" : {
    "partenaires" : [ {
      "ville" : {
        "pays" : { }
      },
      "civilite" : { },
      "nationalite" : { },
      "nom" : "Panier",
      "prenom" : "Eric",
      "labelIsVIP" : "non"
    } ],
    "title" : "Test Jackson"
  },
  "fielderrors" : {
    "objet.propriete" : [ "Une erreur sur un champ" ]
  },
  "errors" : [ "Une erreur" ],
  "messages" : [ "Un message", "Un deuxième message" ]
}
```

Les données sérialisées sont regroupées sous la clé « data ».

Les trois autres entrées sont purement techniques et correspondent aux messages, erreurs et erreurs de niveau « champ » affectées à l'action après validation, erreur métier ou technique.

3.3.9 Gestion des messages

Pour afficher les messages d'erreur et d'information dans une page, il faut utiliser les tags Struts2 suivant :

```
<div class="messageBox errorBox errorBox_idForm">
   <s:fielderror title="<h3 class='titleError'>Erreurs de saisie</h3>"/>
   <s:actionerror title="<h3 class='titleError'>Erreurs</h3>"/>
</div>
<div class="messageBox infoBox infoBox_idForm">
   <s:actionmessage title="<h3 class='titleInfo'>Infos</h3>"/>
</div>
```

Les messages d'erreurs et d'information doivent être affichés dans des zones de notification portant la classe « ***messageBox*** ». Ces deux éléments se distinguant par leurs classes respectives « ***errorBox*** » et « ***infoBox*** ».

Remarque : Pour la validation des formulaires, ajouter en plus les classes correspondantes avec l'id du formulaire concerné.

Pour rediriger les erreurs vers la page courante, il faut surcharger le retour de l'action dans la configuration *Struts* :

```
<package name="recherche" namespace="/dyn/protected" extends="struts-hornet-hornettemplate">
  <action name="recherche" class="fr.gouv.diplomatie.hornettemplate.web.action.Recherche"
method="rechercher">

    <result name="error" type="tiles">recherche.tile.error</result>
    <result name="input" type="tiles">recherche.tile.error</result>
  </action>
</package>
```

Les retours en erreur doivent être associés à une définition de page dans le fichier de configuration *tiles*.

Cette définition de page doit étendre la page courante en modifiant le paramètre d'erreur :

```
<definition name="recherche.tile.error" extends="recherche">
   <put-attribute name="pageErreur" value="true" />
</definition>
```

> ➲ **Une bonne pratique peut être de factoriser les zones de notification des pages dans une jsp (« notification.jsp ») comme dans l'applitutoriel ou hornettemplate, puis d'inclure celle-ci à la page via une ligne du type :**
> `<%@ include file="/WEB-INF/tiles-jsp/commun/notification.jsp" %>`

3.4 Utilisation de Spring dans Struts2

Struts doit être paramétré pour déléguer la construction des actions à Spring (c'est le cas pour l'application d'exemple). Cela permet à Spring de fournir les instances de Services aux actions Struts.

La factory utilisée par struts2 est créée par le ServletContextListener fourni par Spring.

```
org.springframework.web.context.ContextLoaderListener
```

Les fichiers de configuration Spring sont indiqués par les nœuds suivants dans le web.xml

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath*:spring-appContext-web.xml</param-value>
</context-param>
```

Dans le fichier de configuration spring « spring-appContext-web.xml » global sont importés les autres fichiers de configurations spring :

```
<!-- Import des fichiers de configuration spring -->
<import resource="classpath:/spring-appContext-common.xml"/>
```

```
<import resource="classpath:/spring-appContext-security.xml"/>
<import resource="classpath:/spring-appContext-aopMetrologie.xml"/>
<import resource="classpath:/spring-appContext-mail.xml"/>
<import resource="classpath:/spring-appContext-quartz.xml"/>
```

Dans le fichier de configuration spring « spring-appContext-common.xml » sont importés les fichiers de configurations des différentes couches :

```
<!-- Import des fichiers de configuration spring -->
<import resource="classpath:/spring-appContext-common-service.xml"/>
<import resource="classpath:/spring-appContext-common-datasource.xml"/>
<import resource="classpath:/spring-appContext-common-dao.xml"/>
```

Le ContextLoaderListener met le « ApplicationContext » de Spring à disposition de l'application sous forme d'attribut du servletContext.

Le nom de l'attribut est

org.springframework.web.context.WebApplicationContext.*ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE*

Le pseudo code pour récupérer la factory est :

```
ApplicationContext springServiceFactory =(ApplicationContext)
getServletContext().getAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE);
```

3.5 Partie Business

3.5.1 Services

- Les Services représentent la logique métier de l'application. Ils doivent être réutilisables au sein de l'application (partage entre domaines fonctionnels), mais également entre applications (au travers d'une couche Web-Service par exemple).

- Les services sont implémentés avec des classes JAVA simple. Ils possèdent obligatoirement une interface JAVA.

- Les services peuvent effectués des traitements simple ou complexe en réutilisant d'autres services (internes à l'application, ou externe).

- Les services collaborent avec les autres services uniquement via leur interface. C'est Spring qui est chargé de fournir l'implémentation appropriée

- Les services collaborent avec les DAO uniquement via leur interface. C'est Spring qui est chargé de fournir l'implémentation appropriée

- Les services sont généralement des singletons. Ils ne doivent donc pas contenir de données membres constituant un état sinon celles-ci conserveront leurs valeurs d'initialisation d'un appel à l'autre (voir exemple).

- Les services sont décrits dans le fichier Spring **src/ spring-appContext-common-service.xml**

Exemple de service contenant une donnée membre qui constitue un état :

```
public class MonServiceImpl {

    /** <code> maListeErreurs </code> the maListeErreurs     */
    private List<HistoriqueBO> maListeErreurs;

    /** <code>daoHistorique</code> the daoHistorique     */
    private HistoriqueDAO daoHistorique;

    /** <code>daoPurge</code> the daoPurge     */
    private PurgeDAO daoPurge;

    /**
     * @param id identifiant d'une boite aux lettres
```

```
     * @return la liste des historiques
     */
    public List<HistoriqueBO> doTraitement(String id) {
        List <HistoriqueBO> liste = new ArrayList<HistoriqueBO>();
        HistoriqueBO historique = new HistoriqueBO();
        historique.setIdBal(id);
        liste = this.daoHistorique.selectHistoriqueByIdBal(
                                historique.getHistoriqueVO());
        for (HistoriqueBO histo : liste) {
            if (histo.getHisAction() == null) {
                //Ajout d'elements à un objet membre d'un singleton
                this.maListeErreurs.add(histo);
            }
        }
        liste.removeAll(this.maListeErreurs);
        return liste;
    }

    /**
     * @return liste des purges
     */
    public List<Purge> doSomethingElse() {
  //utilisation de la liste du singleton dont le contenu peut être modifié
  //au moment de sa lecture par simple appel à la méthode doTraitement
        for (HistoriqueBO histo : this.maListeErreurs) {
            Purge maPurge = new Purge();
            maPurge.setPurIdbal(histo.getIdBal());
            maPurge.setPurDatepurge(new Date());
            this.daoPurge.insert(maPurge);
        }
        PurgeExample example = new PurgeExample();
        example.setOrderByClause("DATE PURGE");
        return this.daoPurge.selectByExample(example);
    }
}
```

Remarque : Pour cette exemple il est conseillé de supprimer la donnée membre `maListeErreurs` du service et de passer cet objet liste en paramètre des fonctions `doTraitement` et `doSomethingElse`.

Exemple de déclaration de service utilisant un autre service :

```
<bean id="EntrepriseService" class="hornet.projet.business.service.EntrepriseServiceImpl">
        <constructor-arg ref="EntrepriseDAO" index="0" />
        <constructor-arg ref="ReferenceService" index="1" />
</bean>

<bean id="ReferenceService" class="hornet.projet.business.service.ReferenceServiceImpl">
  <constructor-arg ref="ReferenceDAO" index="0"/>
</bean>
```

3.5.2 Ambigüités sur les instanciations

L'utilisation de l'injection de dépendance par constructeur peut mener à des ambigüités.

Prenons un exemple :

```
public class UnObjet {

    public UnObjet(String str, int entier) {
        …
    }

}
```

Avec la configuration Spring suivante :

```
<bean id="UnObjet" class="com.example.UnObjet">
    <constructor-arg value="string1" />
    <constructor-arg value="123" />
</bean>
```

Les valeurs fournies ici pour les arguments du constructeur ne permettent pas à Spring de les différentier. Chacune pourrait potentiellement représenter un nombre ou une chaîne.

Afin de lever l'ambigüité, il est possible de spécifier le type des arguments :

```
<bean id="UnObjet" class="com.example.UnObjet">
    <constructor-arg value="string1" type="java.lang.String" />
    <constructor-arg value="123" type="int"/>
</bean>
```

Une autre solution consiste à renseigner le nom de l'argument associé à la valeur ou à la référence :

```
<bean id="UnObjet" class="com.example.UnObjet">
    <constructor-arg value="string1" name="str" />
    <constructor-arg value="123" name="entier"/>
</bean>
```

Cette méthode n'est cependant pas fiable car elle fonctionne uniquement si le code source Java est compilé avec le flag « debug » activé. Dans ces conditions, il est tout à fait envisageable que cette configuration soit correcte pour les tests en développement mais provoque une erreur lors du déploiement si le flag « debug » est désactivé pour la génération du war.

Pour rendre la configuration Spring indépendante de la méthode de compilation, l'attribut « index » (basé à 0) du tag « constructor-arg » est donc à privilégier :

```
<bean id="UnObjet" class="com.example.UnObjet">
    <constructor-arg value="string1" index="0" />
    <constructor-arg value="123" index="1"/>
</bean>
```

Même dans le cas de l'utilisation d'un seul argument, l'index doit être précisé afin de lever toute ambiguïté sur son utilisation.

Pour plus de détails, se référer à la documentation en ligne de Spring sur ce sujet :
http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html#beans-factory-ctor-arguments-name

En plus de palier à ce type d'erreurs, l'activation du flag de « debug » pour la compilation des sources permet également de rajouter des informations supplémentaires en cas de disfonctionnement. Ces éléments permettent de faciliter le diagnostique en cas d'incident.

3.5.3 Services Spring et scope

Par défaut, un service Spring est un singleton, la valeur par défaut de l'attribut scope étant « singleton » :

```
<bean id="monService" class="hornet.projet.business.service.ReferenceServiceImpl" scope="singleton">
  <constructor-arg ref="ReferenceDAO" index="0"/>
</bean>
```

Singleton signifie qu'une seule instance (ou bean) sera utilisée pour toutes les références à « monService

Les autres valeurs possibles pour scope dans une application web sont :

- prototype : une instance est créée pour chaque référence au bean,
- request : chaque requête http possède sa propre instance du bean,
- session : chaque session http possède sa propre instance du bean.

Dans la majeure partie des cas, le scope singleton est celui à privilégier, chaque service devant impérativement être « thread safe » (permettre et gérer correctement les accès concurrents réalisés par plusieurs threads).

Dans certains cas, les classes et beans nécessaires au fonctionnement d'un service Spring ne sont pas thread safe et des nouvelles instances doivent impérativement être crées à chaque appel.

Un pattern possible est alors :

1. Déclarer le bean non « thread safe » en prototype (« request » peut aussi convenir) :

```
<bean id="monBeanNonThreadSafe" class="hornet.projet.business.BeanNonThreadSafeImpl" scope="prototype">
</bean>
```

2. Déclarer le bean de service en y ajoutant la définition « lookup-method »

```
<bean id="monBeanService" class="hornet.projet.business.BeanThreadSafeImpl">
       <look-up name="getBeanNonThreadSafe" bean="monBeanNonThreadSafe">
</bean>
```

3. Déclarer la méthode abstraite dans la classe appelante

```
public class BeanThreadSafeImpl {
    private abstract BeanNonThreadSafe getBeanNonThreadSafe();

    public void travail() {
        getBeanNonThreadSafe().faitQuelqueChose() ;
    }
}
```

3.5.4 Business Objets (BO)

- Les BO représentent le modèle métier de l'application.

- Un BO est une classe JAVA simple sans dépendance sur l'architecture technique. Il possède au moins un constructeur par défaut et chaque propriété est accessible via des getter/setter.

- Un BO est constitué de propriétés simples, mais peut aussi étendre ou agréger d'autres BO.

- Un BO peut être construit à partir d'un unique VO (mapping un vers un) ou d'un ensemble de VO provenant éventuellement de sources différentes (mapping un à plusieurs). La conversion entre BO et VO sera toujours effectuée dans la couche service.

- Un BO peut contenir de la logique métier, mais ne doit pas faire appel à la couche service, dao, accéder à d'autres ressources, …

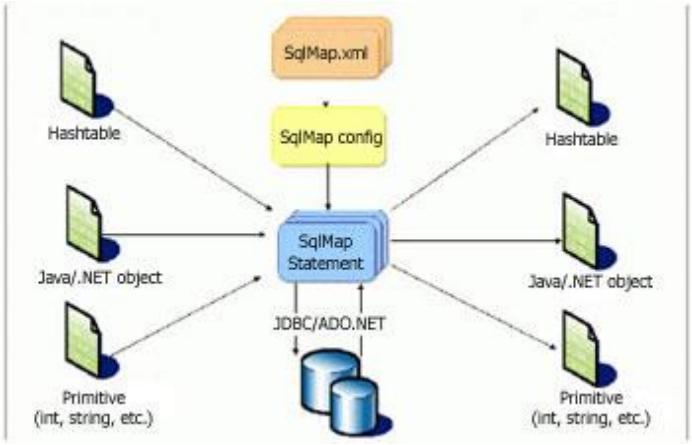
3.6 Partie Integration

3.6.1 Architecture

Le framework MyBatis est composé de deux modules : MyBatis-Data Mapper et MyBatis–Dao.

Cependant l'architecture étant fortement basée sur Spring, c'est SpringDao avec support MyBatis qui sera utilisé en remplacement de MyBatis-DAO, afin notamment de bénéficier de la gestion des transactions.

Data-Mapper est le module de mapping entre les requêtes SQL et les javabeans (VO ou BO). Il permet de mapper les types de base du langage (types primitifs, wrapper, String, Date,…) sur les types JDBC, mais également de mapper un resultset sur un graphe d'objets.



3.6.2 Générateur Ibator

Les DAO et VO peuvent être générés avec Ibator, afin de disposer d'un jeu de méthodes CRUD de départ. Le code pourra ensuite être personnalisé et régénéré sans perdre ses modifications. Attention : les méthodes CRUD générées n'opèrent que sur une table.

Pré-requis :

- Disposer du plugin Ibator for eclipse (faire un update sur le serveur d'indus A3, si besoin)
- Avoir créé le schéma de la base.

Etapes :

- Créer un fichier abatorConfig.xml à la racine du projet contenant au minimum :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE abatorConfiguration PUBLIC "-//Apache Software Foundation//DTD Abator for MyBatis Configuration
1.0//EN"
  "http://ibatis.apache.org/dtd/abator-config 1 0.dtd">

<abatorConfiguration>
  <abatorContext generatorSet="Java5">    <!-- TODO: Add Database Connection Information -->
    <jdbcConnection driverClass="com.mysql.jdbc.Driver"
        connectionURL="jdbc:mysql://130.177.222.200:3308/gca3 dev"
        userId="gca3 dev dbo"
        password="gca3_dev_dbo">
      <classPathEntry location="D:\Documents and Settings\pzh8z1\workspace\GCA3\Serveur\WEB-INF\lib\mysql-
connector-java-3.1.7-bin.jar" />
    </jdbcConnection>

    <javaModelGenerator targetPackage="com.eds.gca3.integration.vo" targetProject="GCA3\Serveur\src" />
    <sqlMapGenerator targetPackage="com.eds.gca3.integration.dao.map" targetProject="GCA3\Serveur\src" />
    <daoGenerator type="SPRING" targetPackage="com.eds.gca3.integration.dao"
targetProject="GCA3\Serveur\src" />

    <table schema="dbo" tableName="%">
    </table>

  </abatorContext>
</abatorConfiguration>
```

- Configurer la connexion JDBC (le chemin du driver est absolu)
- Dans les balises javaModelGenerator, sqlMapGenerator, daoGenerator , attribut targetPackage, remplacer le package « com.eds.gca3 » par celui du projet. Dans l'attribut targetProject, remplacer « GCA3 » par le nom du projet Eclipse
- Enfin pour lancer la génération : clic droit sur abatorConfig.xml / Generate IBATIS Artifacts

Fichiers générés :

Liste des fichiers générés pour une table « MA_TABLE » du schéma :

Artefact généré	fonction
integration.dao.MaTableDAO	interface du DAO
integration.dao.MaTableDAOImpl	implémentation du DAO pour Spring/MyBatis
integration.dao.map.ma_table_sqlMap.xml	Requêtes SQL
integration.vo.MaTable	Classe java permettant de mapper chaque colonne d'une table, excepté les blobs. La conversion de noms et de types SGBD vers Java est définies par défaut dans Abator, mais surchargeable.
integration.vo.MaTableWithBLOBS	Classe java permettant de mapper chaque colonne d'une table, blob ou clob inclus.
integration.vo.MaTableExample	Classe java permettant de créer des filtres de recherche/maj via le DAO (pattern « query by example »)
Integration.vo.MaTableKeys	Classe représentant la clé de la table si la clé est composite

3.6.2.1 Gestion des CLOB et BLOB

Les types SGBD CLOB et BLOB sont mappés respectivement sur les types « java.lang.String » et « byte[] »

Pour les tables contenant un champ CLOB ou un BLOB, certaines méthodes DAO générées seront suffixées par «withBLOB » et « withoutBLOB » et utiliseront des vo également suffixés. Le code permet donc d'être optimisé dans le cas où on ne souhaite pas accéder aux BLOBS.

3.6.3 Intégration dans Spring

Les classes générées par Ibator utilisent l'API de Spring-DAO (Template de code, injection de la dataSource, …)

Procédure :

- Créer un fichier spring-appContext-common-datasource.xml.xml dans le source-folder (src) et ayant cette structure :

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Configuration de  Spring, spécialement pour les couches DAO :
  - précise la configuration MyBatis,
  - précise le mode de connexion utilisé (ici jdbc direct)
  - et indique à Spring d'utiliser le mode "transaction par annotation"

le fichier spring-appContext-common-datasource.xml est importé dans  le fichier spring-appContext-
common.xml
-->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="
          http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd
                http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.0.xsd
                http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-
tx-3.0.xsd"
    >

  <!-- ========================= RESOURCE DEFINITIONS ========================= -->

  <!-- Mode NORMAL : DataSource JNDI, à définir comme ressource du serveur JEE -->
        <jee:jndi-lookup id="dataSource" jndi-name="java:comp/env/jdbc/ProtoDS"/>

  <!-- Mode TEST UNITAIRE : Datasource locale gérée par Spring, sans pool de connexion -->
  <!--bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource" destroy-
method="close">
```

```
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/sample" />
        <property name="username" value="sample"/>
        <property name="password" value="sample"/>
  </bean-->

  <!-- SqlMap setup for MyBatis Database Layer -->
  <bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
       <property name="configLocation">
           <value>classpath:ibatis.xml</value>
       </property>
       <property name="useTransactionAwareDataSource">
           <value>true</value>
       </property>
  </bean>

  <bean id="sqlMapClientTemplate" class="org.springframework.orm.ibatis.SqlMapClientTemplate">
        <property name="sqlMapClient" ref="sqlMapClient" />
        <property name="dataSource" ref="dataSource" />
  </bean>

  <!-- Transaction manager pour une unique DataSource JDBC -->
  <bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
  </bean>

  <!--
        Instruct Spring to perform declarative transaction management
        automatically on annotated classes.
  -->
  <tx:annotation-driven transaction-manager="txManager"/>

  <!-- ========================= couche DAO MYBATIS ========================= -->
</beans>
```

- Paramétrer la dataSource
- A la fin du fichier, déclarer les beans DAO générés par Ibator et qui seront utilisé par l'application :

```
<bean id="ProductDAO" class="hornet.projet.integration.dao.ProductDAOImpl">
  <property name="sqlMapClientTemplate" ref="sqlMapClientTemplate" />
</bean>
```

- Créer un fichier ibatis.xml dans le source-folder (src), afin de déclarer les fichiers sqlMap.xml générés par Ibator:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig
PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN" "http://ibatis.apache.org/dtd/sql-map-config-
2.dtd">

<!-- Configuration MyBatis, chargé par SpringDAO dans spring-appContext-common-datasource.xml -->
<sqlMapConfig>
  <settings
        useStatementNamespaces="true"
  />
  <sqlMap resource="hornet/projet/integration/dao/maps/PRODUCT_SqlMap.xml"/>
  <sqlMap resource="hornet/projet/integration/dao/maps/procedures_SqlMap.xml"/>
</sqlMapConfig>
```

- Ne pas oublier de charger « spring-appContext-common-datasource.xml » à l'initialisation de Spring. Dans une application Web, il faudra l'importer dans le fichier « spring-appContext-common.xml »:

```
<!-- Import des fichiers de configuration spring -->
<import resource="classpath:/spring-appContext-common-service.xml"/>
<import resource="classpath:/spring-appContext-common-datasource.xml"/>
<import resource="classpath:/spring-appContext-common-dao.xml"/>
```

3.6.4 Développement spécifique MyBatis

Si les DAO, VO et SQLMaps.xml générés par Ibator ne suffisent pas, il est possible d'ajouter du code directement dans les fichiers générés, ou dans des fichiers séparés pour plus de lisibilité.

Règles à respecter :

- Ecrire du SQL standard tant que c'est possible (i.e. : éviter les appels de fonction ou procédures stockées spécifiques au SGBD), afin d'améliorer la portabilité et ainsi permettre l'utilisation de HSQLDB pour les tests unitaires.
- Les requêtes avec jointure doivent utiliser la syntaxe (LEFT OUTER) JOIN … ON et non la forme « FROM A,B »
- **On pourra mapper les jointures sur un graphe d'objet BO uniquement si l'isolation par les VO n'est pas nécessaire (gain de temps).**
- Si une requête utilise du SQL spécifique SGBD, ajouter à son id un suffixe correspondant au nom du sgbd :

```
<insert id="insertProduct-ORACLE" parameterClass="com.domain.Product">
  insert into PRODUCT (PRD_ID,PRD_DESCRIPTION)
        values (#id#,#description#)
  <selectKey resultClass="int" type="pre" >
        SELECT STOCKIDSEQUENCE.NEXTVAL AS ID FROM DUAL
  </selectKey>
</insert>
```

3.6.4.1 Ajout d'une requête

Règle : Vérifier que la requête ne peut pas être réalisée via une des méthodes générées, notamment **selectByExample, insertByExample, updateByExample, deleteByExample**

3.6.4.1.1select (avec jointure)

Règle : si une requête de sélection comporte des jointures, elle doit figurer dans le fichier SqlMap.xml correspondant à la table principale de la requête.

Règle : Les données renvoyées par une requête de jointure peuvent être mappées sur un VO spécifique à cette jointure, ou directement un BO.

On pourra soit utiliser les alias « as » SQL pour le mapping des colonnes sur les propriétés de la classe ou définir un ResultMap. L'intérêt du ResultMap est d'être réutilisable et pouvoir gérer les graphes d'objet.

Exemple mapping sur un VO spécifique :

- Ajouter dans le fichier person_SqlMap.xml :

```
<select id="getPersonWithRole" parameterClass=”int” resultClass="hornet.projet.integration.vo.PersonWithRole">
SELECT   PER_ID as id,
  PER_FIRST_NAME as firstName,
  PER_LAST_NAME as lastName,
  PER_BIRTH_DATE as birthDate,
  PER_WEIGHT_KG as weightInKilograms,
  PER_HEIGHT_M as heightInMeters,
  ROL_NAME as roleName
FROM PERSON
JOIN ROLE on PERSON.ROL_ID = ROLE.ROL_ID
WHERE PER_ID = #value#
</select>
```

- Créer une classe java **hornet.projet.integration.vo.PersonWithRole** avec les propriétés nécessaires.

3.6.4.1.2insert/update/delete

Règle : Les données à mettre à jour doivent être modélisées par une classe Java, dont le nom est suffisamment explicite. La mise à jour portant sur une table à la fois, on pourra réutiliser la classe VO générée appropriée.

Chaque paramètre de requête #param# correspond au nom d'une propriété Javabean.

Exemple :

```
<insert id="insertPerson" parameterClass="projet.integration.vo.Person">
INSERT INTO
  PERSON (PER_ID, PER_FIRST_NAME, PER_LAST_NAME,
          PER_BIRTH_DATE, PER_WEIGHT_KG, PER_HEIGHT_M)
  VALUES (#id#, #firstName#, #lastName#,
          #birthDate#, #weightInKilograms#, #heightInMeters#)
</insert>

<update id="updatePerson" parameterClass="projet.integration.vo.Person">
UPDATE PERSON
  SET PER_FIRST_NAME = #firstName#,
  PER_LAST_NAME = #lastName#, PER_BIRTH_DATE = #birthDate#,
  PER_WEIGHT_KG = #weightInKilograms#,
  PER_HEIGHT_M = #heightInMeters#
WHERE PER_ID = #id#
</update>

<delete id="deletePersonById" parameterClass="projet.integration.vo.Person">
DELETE PERSON
WHERE PER_ID = #id#
</delete>
```

3.6.4.1.3Récupération de la PK après insertion

Lorsque est elle supportée (ceci dépend de la version du SGBD et de la version des pilote JDBC), on privilegiera la syntaxe standard SQL suivante :

```
<insert id="insertProduct-GENERIC" parameterClass="product">
    INSERT INTO PRODUCT (PRD_DESCRIPTION) VALUES (#description#) RETURNING PRD_ID
</insert>
```

Dans le cas contraire, on utilisera la balise <selectKey> de MyBatis qui contiendra l'instruction SQL spécifique à chaque SGBD pour récupérer la valeur de la clé primaire utilisée à l'insertion.

- Cas oracle :

```
<insert id="insertProduct-ORACLE" parameterClass="product">
    <selectKey resultClass="int" type="pre" keyProperty="id" >
        SELECT STOCKIDSEQUENCE.NEXTVAL AS VALUE FROM DUAL
    </selectKey>
    INSERT INTO PRODUCT (PRD_ID, PRD_DESCRIPTION) VALUES (#id#, #description#)
</insert>
```

- Cas ms sql serveur 2005 :

```
<insert id="insertProduct-MS-SQL" parameterClass="product">
    INSERT INTO PRODUCT (PRD DESCRIPTION) VALUES (#description#)
    <selectKey resultClass="int" type="post" keyProperty="id" >
        SELECT @@IDENTITY AS VALUE
    </selectKey>
</insert>
```

- Cas mySQL :

```
<insert id="insertProduct-MYSQL" parameterClass="product">
    INSERT INTO PRODUCT (PRD DESCRIPTION) VALUES (#description#)
    <selectKey resultClass="int" type="post" keyProperty="id" >
        SELECT LAST_INSERT_ID() AS VALUE
    </selectKey>
</insert>
```

- Cas postgreSQL :

```
<insert id="insertProduct-POSTGRESQL" parameterClass="product">
    INSERT INTO PRODUCT (PRD_DESCRIPTION) VALUES (#description#)
    <selectKey resultClass="int" type="post" keyProperty="id" >
        SELECT currval('PRODUCT PRD ID SEQ')
    </selectKey>
</insert>
```

Note : Une séquence est implicitement créée lorsque l'on déclare la création d'une table avec une colonne de type SERIAL. La colonne se transformera en type INTEGER mais aura une valeur par défaut à nextval('ma_sequence');

Exemple avec une table nommé ‘secteur’ et une colonne pk nommée ‘id_secteur’, la séquence implicitement crée portera le nom ‘secteur_id_secteur_seq’.

3.6.4.1.4Construction dynamique de requête SQL

On pourra s’inspirer du code selectByExample généré par MyBatis.

3.6.4.2 Appel d’une procédure ou fonction stockée

Exemples de procedures stockées (MySQL) :

```
DELIMITER $$

CREATE PROCEDURE sp swap(INOUT a DECIMAL,INOUT b DECIMAL)
BEGIN
  DECLARE t DECIMAL;
  set t = b;
  set b = a;
  set a = t;
END $$

CREATE PROCEDURE sp_productsMaxPrice(price DECIMAL)
BEGIN
  SELECT * from PRODUCT where PRODUCT PRICE < price;
END $$

CREATE FUNCTION sf_exp(val FLOAT) RETURNS FLOAT
BEGIN
  RETURN 1 + val*(1 + val/2*(1 + val/3*(1 + val/4)));
END $$

CREATE PROCEDURE sp_productsMaxPrice2(price DECIMAL)
BEGIN
  SELECT * from PRODUCT where PRODUCT_PRICE < price;

  return price;
END $$


DELIMITER ;
```

3.6.4.2.1Déclaration dans SqlMap.xml :

```
  <resultMap class="Product" id="ProductResult">
    <result column="PRODUCT_ID" jdbcType="INTEGER" property="productId" />
    <result column="PRODUCT_LABEL" jdbcType="VARCHAR" property="productLabel" />
```

```
    <result column="PRODUCT PRICE" jdbcType="DECIMAL" property="productPrice" />
    <result column="PRODUCT VALIDITY" jdbcType="TIMESTAMP" property="productValidity" />
  </resultMap>

  <parameterMap class="java.util.Map" id="map1">
    <parameter property="price" mode="IN"/>
  </parameterMap>

  <parameterMap class="java.util.Map" id="map2">
    <parameter property="a" mode="INOUT"/>
    <parameter property="b" mode="INOUT"/>
  </parameterMap>

  <parameterMap class="java.util.Map" id="map4">
    <parameter property="result" mode="OUT"/>
    <parameter property="value" mode="IN"/>
  </parameterMap>

  <parameterMap class="java.util.Map" id="map5">
    <parameter property="priceOut" mode="OUT"/>
    <parameter property="price" mode="IN"/>
  </parameterMap>


    <!--
      Procédure stockée avec paramètre IN retournant un resultSet
    -->
  <procedure id="sp productsMaxPrice" parameterMap="map1" resultMap="ProductResult">
        {call sp_productsMaxPrice(?)}
  </procedure>

    <!--
      Procédure stockée effectuant un échange de ses paramètres (INOUT)
    -->
  <procedure id="sp_swap" parameterMap="map2">
        {call sp_swap(?,?)}
  </procedure>

    <!--
      Fonction stockée effectuant une approximation (grossière) de l'exponentielle
    -->
  <procedure id="sf_exp" parameterMap="map4">
        {? = call sf_exp(?)}
  </procedure>

    <!--
      Fonction stockée en utilisant les paramètres nommés
    -->
  <procedure id="sf_exp2" parameterClass="java.util.Map">
        {#result,mode=OUT# = call sf exp(#value#)}
  </procedure>

    <!--
      Mélange de procédure et de fonction
    -->
  <procedure id="sp productsMaxPrice2" parameterMap="map5" resultMap="ProductResult">
        {? = call sp productsMaxPrice2(?)}
  </procedure>

    <!--
      Mélange de procédure et de fonction avec paramètres nommés
    -->
  <procedure id="sf exp3" parameterClass="java.util.Map" resultMap="ProductResult">
        {#priceOut,mode=OUT# = call sp_productsMaxPrice2(#price#)}
  </procedure>
```

--> Les modes de paramètre possibles sont IN (paramètre d'entrée), OUT (paramètre de sortie) ou INOUT (les deux). Le mode par défaut est IN.

3.6.4.2.2Classe DAO :

Règles :

- Si la procédure retourne un result-set interne (sp_productsMaxPrice),utiliser les methodes queryForList() ou queryForObject() pour le récupérer
- Si la procédure utilise des paramètres OUT ou INOUT sans rien renvoyer, utiliser la méthode update.

- Dans le cas d'une fonction, déclarer la valeur de retour comme premier paramètre de type OUT et utiliser la méthode update :

```
public class ProceduresDaoImpl extends SqlMapClientDaoSupport implements ProceduresDao {

    /** {@inheritDoc} */
    @SuppressWarnings("unchecked")
    public List<Product> selectProductsMaxPrice(double price) {
        Map<String, Object> maps = new TreeMap<String, Object>();
        maps.put("price", price);
        return getSqlMapClientTemplate().queryForList("procedures.sp_productsMaxPrice",maps);
    }

    public void swap(double[] a, double[] b) {
        Map<String, Object> maps = new TreeMap<String, Object>();
        maps.put("a", a[0]);
        maps.put("b", b[0]);
        getSqlMapClientTemplate().update("procedures.sp swap",maps);
        a[0] = new Double((String)maps.get("a"));
        b[0] = new Double((String)maps.get("b"));
    }

    public float exp(float v) {
        Map<String, Object> maps = new TreeMap<String, Object>();
        maps.put("value", v);
        getSqlMapClientTemplate().update("procedures.sf_exp",maps);
        return (Float) new Float((String)maps.get("result"));
    }
}
```

- Dans le cas d'une procédure qui renvoie et un result-set et une valeur de retour et/ou des paramètres OUT ou INOUT, utiliser les méthodes queryForList() ou queryForObject().

Remarques :
Les curseurs peuvent également être mappés sur le type JDBC « OTHER »

3.6.4.3 N+1 selects

Le Problème des N+1 selects apparait lorsqu'on souhaite charger un graphe d'objet. Par exemple quand on charge une liste de N objets avec 1 requête puis pour chaque objet, on effectue une nouvelle requête pour charger un objet lié. Pour résoudre ce problème on utilisera une jointure et un mapping approprié (ResultMap).

On pourra directement réutiliser le graphe d'objet des BO si l'isolation par les VO n'est pas nécessaire.

- Relation 1-1
  Le but est de Mapper sur un objet contenant une référence à un autre objet.

- Relation 1-N ou M-N
  Le but est de Mapper sur un objet contenant une collection.
  On utilise pour cela un ResultMap avec l'attribut groupBy.

```
<resultMap id="categoryResult" class="hornet.projet.business.bo.CategoryBO" groupBy="id">
  <result property="id" column="CAT_ID"/>
  <result property="description" column="CAT_DESCRIPTION"/>
  <result property="productList" resultMap="ProductCategory.productResult"/>
</resultMap>
<resultMap id="productResult" class=" hornet.projet.business.bo.ProductBO">
  <result property="id" column="PRD ID"/>
  <result property="description" column="PRD_DESCRIPTION"/>
</resultMap>
<select id="getCategory" parameterClass="int" resultMap="categoryResult">
select C.CAT ID, C.CAT DESCRIPTION, P.PRD ID, P.PRD DESCRIPTION
from CATEGORY C
left outer join PRODUCT P
on C.CAT_ID = P.PRD_CAT_ID
where CAT_ID = #value#
```

```
</select>
```

3.6.4.4 Gestion des batchs updates

3.6.4.4.1Principes

Les batchs updates sont des batchs JDBC. Le principe de ces batchs est d'éviter d'exécuter unitairement des ordres SQL de création, modification et suppression, surtout s'ils sont nombreux, afin d'optimiser les temps d'exécution. Pour ce faire on distingue les ordres de création, de modification et de suppression que l'on stocke dans trois listes différentes. Puis, on exécute dans un batch JDBC, par groupe de 50, les ordres de création ensuite ceux de modification et enfin ceux de suppression. Ainsi, on réussit à agréger les ordres SQL et à les exécuter par paquets de 50 et non unitairement ce qui permet de multiplier par dix le gain d'exécution.

3.6.4.4.2Exemple

Pour réaliser les batch updates il est nécessaire d'utiliser l'interface SqlMapClient afin d'extraire les ordres SQL de chaque requête pour les agréger.

```
this.getSqlMapClientTemplate().execute(
    new SqlMapClientCallback() {

        public Object doInSqlMapClient(
                final SqlMapExecutor executor) throws SQLException {

            // Nombre d'update par batch
            final int commitSize = 100;

            int totalRecordsUpdated = 0;
            int commitBatchUpdated = 0;
            int total = 0;

            // Démarrage du premier batch
            executor.startBatch();

            for (final PosteJeuneBO poste : listePostes) {

        // Ajout d'un ordre dans la liste
                executor.update(
                    "E_PJE_POSTE_JEUNE.updatePosteJeune", poste);

                commitBatchUpdated++;

        // Controle si on execute le batch
                if ((commitBatchUpdated == commitSize)
                        || (commitBatchUpdated == listePostes.size())) {
                    totalRecordsUpdated += commitBatchUpdated;

                    // Fin du batch
                    total += executor.executeBatch();
                    commitBatchUpdated = 0;

                    // Démarrage d'un autre batch si
                    // nécessaire
                    if (totalRecordsUpdated < listePostes.size()) {
                        executor.startBatch();
                    }

                }
            }

            return Integer.valueOf(total);
        }
    });
```

3.6.5 Cache de requêtes

3.6.5.1 Principes

MyBatis permet de charger en mémoire des requêtes dans un cache afin d'éviter de relancer celles-ci si ce n'est pas nécessaire. On peut ainsi diminuer le temps d'exécution et optimiser l'accès aux données.

Il préférable d'utiliser ce cache en lecture seules, uniquement sur les données qui sont majoritairement accédées en lecture et dont la fréquence de mise à jour est faible.

Il faut impérativement définir la taille du cache et son time out pour renouveler les données stockées en mémoires. En effet, il est important déterminer quels sont les requêtes à conserver dans le cache pour estimer la taille de celui-ci. Il faut noter qu'une requête qui comporte plusieurs clauses dynamiques différentes engendre autant de requête à stocker en cache qu'il existe de combinaison possible pour ces clauses. Par exemple pour les requêtes de type « ByExample », il y aura autant de requêtes stockées en cache que de combinaison possible entre tous les critères. Ainsi, s'il existe au maximum deux clauses sur la requête, le cache contiendra quatre requêtes, une pour chaque combinaison possible.

Pour optimiser la mémoire du cache plusieurs astuces sont possibles :

- Appliquer des algorithmes supplémentaires pour améliorer la gestion du cache, tel que l'algorithme LRU qui supprime du cache les requêtes les moins utilisées.
- Modifier le codes des requêtes de type « ByExample » afin que celles-ci ne retourne que les identifiants. Puis, ajouter un algorithme qui parcourt ce résultat et qui effectue une requête sur les clefs primaires. Ce principe est un peu plus couteux à la première exécution mais il permet de stocker moins de requête dans le cache et de stocker tout l'objet.

Enfin, l'utilisation d'un cache n'a pas d'intérêt si celui-ci n'est pas utilisé à plus de 50%.

Exemple d'utilisation :

```
<cacheModel id="product-cache" implementation="LRU" readOnly="true" serialize="false">
  <flushInterval hours="24"/>
  <flushOnExecute  statement="insertProduct"/>
  <flushOnExecute  statement="updateProduct"/>
  <flushOnExecute  statement="deleteProduct"/>
  <property name="CacheSize" value="100"/>
</cacheModel>


<statement id="getProductList" cacheModel="product-cache">
  select * from PRODUCT where PRD_CAT_ID = #value#
</statement>
```

Pour la requête de récuperation des produits, un modèle de cache est appliqué.

Ce modèle de cache comprend une taille, ainsi que des évènements et un intervalle de temps déclanchant la réinitialisation de celui-ci.

3.6.6 Limiter le nombre de résultat

3.6.6.1 Limiter sur le sgbd

- Cas oracle :
  - Les 100 premiers résultats d'une requête non triée :

```
SELECT *
FROM `t_client`
WHERE ROWNUM <= 100
```

  - Les 100 premiers résultats d'une requête triée :

```
SELECT *
FROM (SELECT * FROM `t_client` ORDER BY name)
WHERE ROWNUM <= 100
```

Le compteur ROWNUM, initialisé à 1, est définit pour chaque résultat d'une requête. La valeur de celui-ci est attribuée puis incrémenté de 1 à chaque ligne retournée par la requête. Lorsque les données retournées ne sont pas triées avec le mot-clé ORDER BY ce principe ne pose pas de problème. Par contre, si on souhaite trier les données et tenir compte de l'ordre alors il est nécessaire d'utiliser une sous-requête. En effet, la valeur de ROWNUM est attribuée pour chaque ligne **avant** le tri final. Ainsi, une fois le résultat ordonné, le critère des 100 premiers ne peut être respecté.

- Cas ms sql serveur 2005 :
  - Les 100 premiers résultats

```
SELECT TOP 100 *
FROM t_client
```

- Cas mySQL et postgreSQL :
  - Les 100 premiers résultats :

```
SELECT *
FROM `ARCHIVA_ARTIFACT`
LIMIT 100
```

3.6.7 Pagination de listes

Plusieurs solutions sont possibles pour la pagination. Le choix se fait en fonction des volumes des données et fréquences d'utilisation et de mise à jour.

Les trois variantes les plus communes sont présentées ici.

3.6.7.1 Paginer sur le sgbd

- Cas oracle :

Il est possible d'utiliser la fonction analytique ROW_NUMBER, pour filtrer le résultat d'une requête sur des intervalles consécutifs. Le principe est assez similaire à ROWNUM. En effet, la fonction ROW_NUMBER est appliquée sur le résultat final de la requête alors que ROWNUM est appliqué pendant l'exécution de celle-ci.

Exemple qui affiche 30 lignes à partir de l'enregistrement 10 :

- avec ROWNUM :

```
SELECT …
FROM (
  SELECT …, ROWNUM num FROM
         (SELECT … FROM t client ORDER BY name)
  )
WHERE num BETWEEN 10 and 40
```

- avec ROW_NUMBER :

```
SELECT …
FROM (
  SELECT …, ROW_NUMBER() OVER (ORDER BY name) num
  FROM t_client
  )
WHERE num BETWEEN 10 and 40
```

Référence : http://oracle.developpez.com/faq/?page=3-1#rownum

- Cas ms sql serveur 2005 :

Affiche 30 lignes à partir de l'enregistrement 10 :

```
SELECT * FROM (
```

```
         SELECT TOP 10 Field1, Field2 FROM (
         SELECT TOP 30 Field1, Field2
         FROM  matable
        ORDER BY monchamp asc
        ) AS tbl1 ORDER BY monchamp desc
        ) AS tbl2 ORDER BY monchamp asc
```

Référence : http://sqlserver.developpez.com/faq/?page=Jeu#Jeu1

Autre approche :

```
select * FROM (
  select *, row_number() over(order by mon_champ) as rownum from maTable
) orderedResults where rownum between 10 and 40
```

- Cas mySQL :

Utilisation de l'instruction :
LIMIT <indice de départ dans la liste de résultat >, <nombre de résultat à afficer >

Exemple, à partir de l'élément 10, afficher 30 résultats :

```
SELECT *
FROM `ARCHIVA_ARTIFACT`
LIMIT 10 , 30
```

- Cas postgreSQL :

La syntaxe s'approche de celle de MySQL mais elle est attention car elle est inversée !

```
SELECT select list
FROM table_expression
[LIMIT { number | ALL }] [OFFSET number]
```

Sur le même exemple : à partir de l'élément 10, afficher 30 résultats :

```
SELECT *
FROM `ARCHIVA_ARTIFACT`
LIMIT 30 OFFSET 10
```

Avantages	Inconvénient
Ne retourne que le resultset de la page souhaitée	-on exécute la requête à chaque fois (ou du moins à chaque changement de page) -Dépendant SGBD

3.6.7.2 Paginer sur le sgbd (2)

Si requête couteuse en temps d'exécution : utiliser une table partagée par les utilisateurs pour stocker le résultat complet de la requête, ensuite la pagination est faite sur cette table selon l'une des techniques précédentes.

Avantages	Inconvénient
on exécute la requête une seule fois	Espace disque plus important, et à priori une table par requête (mais on pourrait ne stocker que les ids du résultat)

3.6.7.3 Paginer sur le serveur

On récupère le resultset complet de la requête, et on pagine sur le serveur.

Avantages	Inconvénient

on exécute la requête une seule fois Indépendant SGBD	Tout le resultset est en session Pas de modif concurrente

3.6.8 Types Java Spécifiques

Il est parfois utile d'avoir des types spécifiques dans des BO. Par exemple on peut prendre le cas d'une date stockée en tant que varchar dans la base de données et que l'on souhaite traité de façon particulière en java.

Considérons que nous avons une classe DateIncomplète qui correspond à notre type. Les objets métier de notre application vont contenir des variables de type DateIncomplète.

Pour qu'MyBatis puissent gérer correctement ce type de champs indépendamment de la structure de l'objet DateIncomplete, il va falloir créer un handler particulier pour ce type de champs qui va implémenter l'interface TypeHandlerCallback.

Exemple :

```
public class DateIncompleteTypeHandler implements TypeHandlerCallback {

    /** {@inheritDoc} */
    public Object getResult(
            ResultGetter arg0) throws SQLException {

        String value = arg0.getString();

        DateIncomplete dateIncomplete = new DateIncomplete();
        dateIncomplete.setValue(value);

        return dateIncomplete;
    }

    /** {@inheritDoc} */
    public void setParameter(
            ParameterSetter arg0, Object arg1) throws SQLException {

        if (arg1 == null) {
            arg0.setNull(Types.VARCHAR);
        } else {
            DateIncomplete dateIncomplete = (DateIncomplete) arg1;
            arg0.setString(dateIncomplete.getValue());
        }
    }

    /** {@inheritDoc} */
    public Object valueOf(
            String arg0) {

        return arg0;
    }

}
```

Le handler comporte trois méthodes :

- getResult : qui permet de transformer les données de la base dans le type Java correspondant (DateIncomplete dans notre exemple)
- setParameter : qui permet de définir le paramètre de requête à partir du type Java (varchar (String) dans notre cas)
- valueOf : permet de tester les valeurs par le gestionnaire, pratique dans le cas du traitement des valeurs NULL ou bien de valeurs prédéfinies par le gestionnaire (valeurs fixes)..

Ensuite il suffit dans la configuration de MyBatis de lui préciser que pour tous les champs de type DateIncomplete il faut utiliser le DateIncompleteTypeHandler. Ceci s'effectue dans le fichier « ibatis.xml »

```
<typeHandler javaType="hornet.projet.utility.DateIncomplete"
        callback="hornet.projet.integration.dao.utility.DateIncompleteTypeHandler" />
```

3.6.1 Spécificités SQL des différents SGBD

Les fonctionnalités avancés d'SQL n'etant pas normalisés, il est nécessaire parfois d'utiliser les syntaxes spécifiques à chaque système de gestion de base de données.

3.6.1.1 Retour de la première expression rencontrée non nulle

Voici les équivalents à utiliser selon le SGBD cible :

- Cas oracle :
  Oracle ne dispose pas de fonction ISNULL(). Cependant on peut utiliser la fonction NVL() qui effectue la même chose. Exemple :

```
SELECT ProductName,UnitPrice*(UnitsInStock+NVL(UnitsOnOrder,0))
FROM Products
```

- Cas ms sql serveur 2005 :

```
SELECT ProductName,UnitPrice*(UnitsInStock+ISNULL(UnitsOnOrder,0))
FROM Products
```

- Cas mySQL :
  MySQL a bien une fonction ISNULL(), cependant elle fonctionne légèrement différemment de la fonction ISNULL() de Miscrosoft. Pour MySQL, on utilisera donc la function IFNULL() à la place. Exemple :

```
SELECT ProductName,UnitPrice*(UnitsInStock+IFNULL(UnitsOnOrder,0))
FROM Products
```

  ou bien la fonction COALESCE()

```
SELECT ProductName,UnitPrice*(UnitsInStock+COALESCE(UnitsOnOrder,0))
FROM Products
```

- Cas postgreSQL :
  La fonction ISNULL() n'existe pas sous PostgreSQL, on utilisera la fonction COALESCE() à la place :

```
SELECT ProductName,UnitPrice*(UnitsInStock+COALESCE(UnitsOnOrder,0))
FROM Products
```

3.6.1.2 Utilisation des LOBs

Les LOB sont les objets larges (de l'anglais « Large OBject) soit typiquement des types de données d'environ 4 ko ou plus, bien que certaines bases de données peuvent gérer plus de 32 ko avant que les données deviennent "larges".

3.6.1.2.1 Avec PostgreSQL

Pour l'utilisation des LOBs, on veillera à :

- Passer le mode autocommit (activé par défaut) à false.
- A ne pas faire de commit, ceci est géré par le commitOnClose de la méthode open de LargeObject

Attention, lors de la suppression, le fichier n'est pas supprimé c'est uniquement l'oid. Il faut donc supprimer le link. Pour cela, il est possible d'installer un package postgresql qui supprime le link par trigger : "create extension lo"

Exemple de code d'insertion/update :

Fichier sqlMap.xml (MyBatis)

```
<update id="updateContenuFichierStream" parameterClass="map" timeout="0">
UPDATE fichier SET
fi_contenu=#contenu#
WHERE gr_id=#idGroupe# AND fi_numero=#numero#
</update>
```

Code java

```
// enregistre
final Map<String, Object> mapParams = new HashMap<String, Object>();

mapParams.put("idGroupe", fichierVo.getIdGroupe());
mapParams.put("numero", fichierVo.getNumero());

Connection conn = null;
long oid = 0;
try {
    conn = this.getSqlMapClientTemplate().getDataSource().getConnection();
    Connection nativeConn = new CommonsDbcpNativeJdbcExtractor().getNativeConnection(conn);

    // All LargeObject API calls must be within a transaction block
    nativeConn.setAutoCommit(false);

    LargeObjectManager manlobj = ((PGConnection)nativeConn).getLargeObjectAPI();

    oid = manlobj.createLO(LargeObjectManager.READ | LargeObjectManager.WRITE);
    LargeObject obj = manlobj.open(oid,true);

    // Copy the data from the file to the large object
    byte buf[] = new byte[2048];
    int s, tl = 0;
    while ((s = fichierVo.getContenu().read(buf, 0, 2048)) > 0) {
        obj.write(buf, 0, s);
        tl += s;
    }
} catch (SQLException e) {
    …
} catch (IOException e) {
    …
} finally {
    // Close the large object
    obj.close();
}

mapParams.put("contenu", oid);
this.getSqlMapClientTemplate().update("updateContenuFichierStream", mapParams);
```

Exemple de code de récupération/lecture :

Code java

```
Connection conn = null;
PreparedStatement ps = null;
ResultSet rs = null;
try {
    conn = this.getSqlMapClientTemplate().getDataSource().getConnection();

    Connection nativeConn = new CommonsDbcpNativeJdbcExtractor().getNativeConnection(conn);

    // All LargeObject API calls must be within a transaction block
    nativeConn.setAutoCommit(false);

    LargeObjectManager manlobj = ((PGConnection)nativeConn).getLargeObjectAPI();

    ps = conn.prepareStatement("SELECT fi_contenu FROM fichier WHERE gr_id=? AND fi_numero=?");
    ps.setInt(1, idGroupe);
    ps.setInt(2, numeroFichier);
    rs = ps.executeQuery();

    while (rs.next()) {
        // Open the large object for reading
        long oid = rs.getLong(1);
        LargeObject obj = manlobj.open(oid,LargeObjectManager.READ, true);

        // Read the data
        byte buf[] = new byte[obj.size()];
        obj.read(buf, 0, obj.size());
        // Do something with the data read here
        …

        // Close the object
```

```
        obj.close();
    }
} catch (SQLException e) {
    …
} catch (IOException e) {
    …
} finally {
    if (rs != null) rs.close();
    If (ps != null) ps.close();
}
```

3.7 Socle technique

3.7.1 Déclaration des ressources JNDI

JNDI signifie Java Naming and Directory Interface, cette API permet :

- d'accéder à différents services de nommage ou de répertoire de façon uniforme ;
- d'organiser et rechercher des informations ou des objets par nommage (java naming and directory interface) ;
- de faire des opérations sur des annuaires (java naming and directory interface)

JNDI est très utilisée dans l'univers des serveurs d'applications Java et fait partie de l'ensemble des APIs Java EE où il permet de lier un nom (par exemple 'base/sql/login') à une information.

Les ressources peuvent être de deux natures différentes :

- Ressource locale
  - <Resource> enfant de l'élément <Context>
  - name : permet de ranger la ressource dans l'arbre JNDI
  - auth : prend pour valeur « Container » ou « Application » pour désigner qui de Tomcat ou du programmeur doit fournir les accréditations pour accéder à la ressource
  - type : la classe Java qui détermine le reste de la configuration de la ressource

- Ressource globale
  - <Resource> enfant de l'élément <Server>/<GlobalNamingResources>
  - La configuration se fait comme pour une ressource locale
  - Dans le contexte : <ResourceLink>
  - type : reprise de celui de la ressource globale
  - name : nom JNDI accessible par l'application
  - global : référence au nom utilisé dans la partie globale

Exemple de ressources :

- JDBC Datasource
- JavaMail Session
- Custom Object Factory
- …

Les sources de données sont gérées comme des sources locales et sont à définir dans META-INF/context.xml

```
<Context path="/PROJET" reloadable="true">
        ...
  <Resource name="jdbc/demoGlobal" type="javax.sql.DataSource"
        auth="Container" username="root" passeword="root"
        driverClassName="net.sourceforge.jtds.jdbc.Driver"
        url="jdbc:jtds:sqlserver://localhost:1433/DEMO" initialSize="2"
        maxActive="9" maxIdle="4" minIdle="2" maxWait="5000" />
```

```
</Context>
```

Le nœud Context contient les caractéristiques suivantes :

- path : discriminant dans l'url qui identifie l'application
- reloadable : scrutation des répertoires classes et lib pour recharger le contexte dynamiquement en cas de modifications.

3.7.2 Gestion de la sécurité

- Utilisation du standard JEE (Tomcat Realm), couplé avec l'intercepteur RolesInterceptor (restriction d'accès aux flux XML, selon profil utilisateur)

- Utilisation de JAAS-Realm

- Gestion de Single-Sign-On (SSO), via des serveurs de type CAS.

3.7.3 Gestion des transactions

Configuration Spring src/spring-appContext-common-datasource.xml:

```
<bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
       <property name="configLocation">
           <value>classpath:sqlConfig.xml</value>
       </property>
       <property name="useTransactionAwareDataSource">
           <value>true</value>
       </property>
</bean>
<!-- Transaction manager pour une unique DataSource JDBC -->
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
</bean>

<!--
       Instruct Spring to perform declarative transaction management
       automatically on annotated classes.
-->
<tx:annotation-driven transaction-manager="txManager"/>
```

- démarcation des transactions au niveau des services en utilisant les annotations.
- annotation de niveau classe : toutes les méthodes sont transactionnelles

```
@Transactional
// exécution de chaque méthode dans une transaction
public class ProductServiceImpl implements ProductService {
```

- annotation de niveau méthode

```
    @Transactional
    // optimisation : transaction en lecture seule
    public List<Product> list() {
```

Cela permet de surcharger la définition de niveau classe. Notamment pour redéfinir la propagation (création d'une nouvelle transaction, réutilisation, …) ou le niveau d'isolation (READ_UNCOMMITED, …)

3.7.4 Gestion des erreurs

3.7.4.1 Erreurs techniques

Concerne les erreurs non fonctionnelles, non prévues et souvent irrécupérables du type rupture de communication réseau (Apache, Navigateur Web, Base de de donnée), perte de session, file not found, out of memory, null pointer, et les Runtime Exception en général.

Le framework génère automatiquement un rapport d'erreur détaillé dans les logs, et un message simplifié destiné à l'utilisateur incluant un identifiant unique de l'erreur produite (au format « instance de serveur »-« date système »-« numéro aléatoire »).

3.7.4.2 Erreurs fonctionnelles

Les erreurs fonctionnelles concernent le métier de l'application, elles doivent donc remonter à l'utilisateur avec un niveau de détail élevé.

Il est possible de lever une erreur fonctionnelle soit dans la couche Service, soit Action. Pour cela il suffit d'ajouter le code :

```
throw new BusinessException(« monCodeErreur », new String[]{argument1, argument2, …}) ;
```

Cette exception remonte la pile d'exécution pour être interceptée par la couche Web, qui se charge de transmettre un message clair pour l'utilisateur (via le result « error »). Le message d'erreur sera recherché dans les fichiers de propriétés (I18N) dans la portée de l'action en cours (cf. section 3.7.5.1)

Remarque : Afin de ne pas surcharger les logs, la pile d'exécution Java de l'exception fonctionnelle n'est affichée que si le niveau de trace est positionné à DEBUG.

3.7.5 Gestion de l'encoding et de l'internationalisation

3.7.5.1 Internationalisation (I18N)

- Couche Web :

Les messages (erreurs, information, etc…) sont stockés dans des fichiers de propriétés I18N appelés resource-bundle (RS).

Algorithme de recherche du RS de Struts :

Recherche des RS dans l'ordre suivant :

- le RS associé à l'action : situé dans le même package que la classe action et nommé MonAction_fr.properties
- le RS associé à l'une des classes action parente
- le RS associé au package : package_fr.properties
- le RS associé à l'un des package ancêtre.
- le RS globale de l'application

Si plusieurs RS sont présents, utilisation du RS le plus proche de celui de la localisation de l'utilisateur (Pays + langue).

Utilisation :

- Dans une classe Action, il suffit d'appeler la méthode getText (« mon.code.message »), qui utilise la localisation par défaut de l'utilisateur (envoyée par la requête http).
- Dans un template JSP, on utilise le tag Struts :

```
<s:text name="mon.code.message" />
```

- Couche Service

Déclaration du préfix des fichiers de propriétés contenant les messages dans le spring-appContext-common-service.xml :

```
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="messages"/>
</bean>
```

Tous les messages seront donc stockés dans des fichiers messages_fr.properties, messages.properties, … .

Ensuite dans le service il suffit de récupérer les messages à partir du context Spring comme ceci :

```
applicationContext.getMessage(
            "Code de la propriété", new Object[]{"parametre1"}, Locale.FRENCH);
```

Pour récupérer l'applicationContext il faut utiliser l'injection Spring (Interface ApplicationContextAware).

Remarque : Les apostrophes doivent être doublées dans tous les libellés des fichiers de propriétés.

Pour utiliser des messages dans du code JavaScript, il est nécessaire d'échapper les libellés :

```
<s:set name="monCodeMessage" value="%{getText('mon.code.message')}"/>
<script type="text/javascript">
            var monCodeMessage = '<s:property value="#monCodeMessage" escape="true" />';
</script>
```

3.7.6 Gestion des logs

- **Utilisation de Log4J. Le code peut référencer directement l'API Log4J** (notamment le Logger)

- Paramétrage dans /src/log4j.properties ou /src/log4j.xml.

3.7.7 Package sun.*

3.7.7.1 Enlever accent

La méthode du framework "StringUtils.enleverAccent" permet de remplacer la classe sun.text.Normalizer du jdk 5 qui est devenu obsolète dans la version 6.

Exemple d'utilisation :

```
String chaine = "abàâäàâcdefghijklmnopqrstuvwxyz0123456789éèêàâ";
String chaineResultat = "abaaaaacdefghijklmnopqrstuvwxyz0123456789eeeaa";

String result = StringUtils.enleverAccent(chaine);
```

3.7.8 Gestion des dépendances

3.7.8.1 Principe

Les dépendances sont gérées par le framework ivy qui s'articule autour de plusieurs fichiers de configuration :

- ivysettings.xml (fichier de paramètrage des repositories)
- ivysettings.properties
- ivy-template.xml (modèle de fichier de configuration des dépendances et plan de configuration, utilisé pour la génération du fichier ivy.xml en fonction d'un numéro de révision : utilisation principal pour les projets multimodules devant utiliser une revision commune @REVISION@)
- ivy.xml (fichier de configuration des dépendances et plan de configuration configuré pour une révision particulière du projet)

Ces fichiers sont présents à la racine du projet.

3.7.8.2 Les repositories

3.7.8.2.1Arborescence

Créer l'arborescence dans le repository, ex :
-org.monorganisation.projet
    -monmodule-projet
        -version (ex : 1.1.0)
            -ivys
            -jars
            -wars
            -javadocs
            -sources
            -…

3.7.8.2.2Format

Les fichiers doivent respecter le format défini.

3.7.8.2.2.1 Jar-artefact

[jar-artefact] = [module]-[revision].jar

3.7.8.2.2.2 War-artefact

[war-artefact] = [module]-[revision].war

3.7.8.2.2.3 Ivy-artefact

[ivy-artefact] = [module]-[revision] -ivy.xml

3.7.8.2.2.4 Source-artefact

[source-artefact] = [module]-[revision] -sources.zip

3.7.8.2.2.5 Javadoc-artefact

[javadoc-artefact] = [module]-[revision] -javadoc.zip
…

3.7.8.3 Fichier de paramètrage : Ivysettings.xml

3.7.8.3.1.1 Développement

Le développement hornet est basé sur ivy. Il est donc nécessaire de configurer les fichiers qui permettent de requêter sur des repositories.

Deux possibilités pour le développement :

- Développement local : repositories sur le filesystem du développeur
- Développement distant : repositories sur une gestionnaire d'artefact

Les fichiers de configuration sont à déposer dans le répertoire de l'utilisateur, ex :

- Windows 7 : C:\Users\NomUtilisateur\.ivy2
- Unix : ~\ .ivy2

3.7.8.3.1.2 Configuration

Voici un exemple de configuration en distant ;

Ivysettings.properties

```
default.artifact.pattern=[organisation]/[module]/[revision]/[type]s/[module]-[revision](-[classifier]).[ext]
default.ivy.pattern=[organisation]/[module]/[revision]/ivys/[module]-[revision]-ivy.xml
```

```
repository.host=URL_DEPOT
repository.url=http://${repository.host}/artifactory


# dépot en ligne pour les fichiers du framework et ses dépendances
repository.integration.url=${repository.url}/repository-integration
repository.integration.artifact.pattern=${default.artifact.pattern}
repository.integration.ivy.pattern=${default.ivy.pattern}

repository.metier.url=${repository.url}/repository-metier
repository.metier.artifact.pattern=${default.artifact.pattern}
repository.metier.ivy.pattern=${default.ivy.pattern}

repository.technique.local.url=${repository.url}/repository-technique-local
repository.technique.local.artifact.pattern=${default.artifact.pattern}
repository.technique.local.ivy.pattern=${default.ivy.pattern}

repository.technique.remote.url=${repository.url}/repository-technique-remote
repository.technique.remote.artifact.pattern=${default.artifact.pattern}
repository.technique.remote.ivy.pattern=${default.ivy.pattern}

repository.technique.tiercesparties.url=${repository.url}/repository-tiercesparties
repository.technique.tiercesparties.artifact.pattern=${default.artifact.pattern}
repository.technique.tiercesparties.ivy.pattern=${default.ivy.pattern}
```

Ivysettings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ivy-settings>
  <properties file="ivysettings.properties" />

  <settings defaultResolver="main" />

  <credentials host="${repository.host}" realm="Artifactory Realm" username="LOGIN"
passwd="MOT_DE_PASS" />

  <resolvers>
    <chain name="main" returnFirst="true" >

<url name="repository-integration">
                  <ivy
pattern="${repository.integration.url}/${repository.integration.ivy.pattern}" />
                  <artifact
pattern="${repository.integration.url}/${repository.integration.artifact.pattern}" />
            </url>

            <url name="repository-technique-local">
                  <ivy
pattern="${repository.technique.local.url}/${repository.technique.local.ivy.pattern}" />
                  <artifact
pattern="${repository.technique.local.url}/${repository.technique.local.artifact.pattern}" />
            </url>

            <url name="repository-technique-remote">
                  <ivy
pattern="${repository.technique.remote.url}/${repository.technique.remote.ivy.pattern}" />
                  <artifact
pattern="${repository.technique.remote.url}/${repository.technique.remote.artifact.pattern}" />
            </url>
```

```
            <url name="repository-technique-tiercesparties">
                  <ivy
pattern="${repository.technique.tiercesparties.url}/${repository.technique.tiercesparties.ivy.pat
tern}" />
                  <artifact
pattern="${repository.technique.tiercesparties.url}/${repository.technique.tiercesparties.artifac
t.pattern}" />
            </url>

            <url name="repository-metier">
                  <ivy pattern="${repository.metier.url}/${repository.metier.ivy.pattern}" />
                  <artifact
pattern="${repository.metier.url}/${repository.metier.artifact.pattern}" />
            </url>
```

3.7.8.3.1.3 Cache

ivy gère deux niveaux de cache : cache de dépendance et un autre pour la résolution de dépendances. Son emplacement par défaut est celui de l'utilisateur de l'OS : **~/.ivy2/cache**.

3.7.8.3.1.4 Resolvers

Les resolvers sont classés selon un ordre bien précis. En effet, cela permet à Ivy de définir l'ordre d'appel des *repositories* pour la recherche d'une dépendance. De plus, cet ordre est couplé à la propriété « *returnFirst* » qui retourne la première dépendance trouvée.

Ne pas oublier que le cache est interrogé en premier par rapport à cette liste.

3.7.8.4 Fichier de configuration : ivy.xml

Une configuration avec le framework Hornet :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ivy-module version="2.0" xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd"
xmlns:m="http://ant.apache.org/ivy/extra" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<info organisation="com.effitic.nantes.hornetTemplate" module="hornetTemplate"
revision="latest.integration" status="SNAPSHOT"/>
<configurations>
<conf name="default" visibility="public" description="runtime dependencies and master artifact can be used
with this conf" extends="runtime,master"/>
<conf name="master" visibility="public" description="contains only the artifact published by this module
itself, with no transitive dependencies"/>
<conf name="compile" visibility="public" description="this is the default scope, used if none is
specified. Compile dependencies are available in all classpaths."/>
<conf name="provided" visibility="public" description="this is much like compile, but indicates you expect
the JDK or a container to provide it. It is only available on the compilation classpath, and is not
transitive."/>
<conf name="runtime" visibility="public" description="this scope indicates that the dependency is not
required for compilation, but is for execution. It is in the runtime and test classpaths, but not the
compile classpath." extends="compile"/>
<conf name="test" visibility="private" description="this scope indicates that the dependency is not
required for normal use of the application, and is only available for the test compilation and execution
phases." extends="runtime"/>
<conf name="system" visibility="public" description="this scope is similar to provided except that you
have to provide the JAR which contains it explicitly. The artifact is always available and is not looked
up in a repository."/>
<conf name="sources" visibility="public" description="this configuration contains the source artifact of
this module, if any."/>
<conf name="javadoc" visibility="public" description="this configuration contains the javadoc artifact of
this module, if any."/>
<conf name="reports" visibility="public" description="this configuration contains the report artifact of
this module, if any."/>
<conf name="optional" visibility="public" description="contains all optional dependencies"/>
<conf name="sonar" visibility="private" description="dependance for sonar analysis"/>
<conf name="cobertura" visibility="private" description="dependance for code coverage"/>
</configurations>
<publications>
<artifact name="hornetTemplate" type="war" ext="war" conf="master"/>
<artifact name="hornetTemplate" type="source" ext="zip" conf="sources" m:classifier="sources"/>
```

```
<artifact name="hornetTemplate" type="javadoc" ext="zip" conf="javadoc" m:classifier="javadoc"/>
<artifact name="hornetTemplate" type="conf" ext="zip" conf="master" m:classifier="config"/>
<artifact name="hornetTemplate" type="conf" ext="zip" conf="master" m:classifier="context"/>
<artifact name="hornetTemplate" type="source" ext="zip" conf="sources" m:classifier="testsources"/>
<artifact name="hornetTemplate" type="report" ext="zip" conf="reports" m:classifier="ivyreport"/>
<artifact name="hornetTemplate" type="report" ext="zip" conf="reports" m:classifier="testreport"/>
<artifact name="hornetTemplate" type="war" ext="zip" conf="master" m:classifier="statique"/>
</publications>
<dependencies>
<dependency org="fr.gouv.diplomatie.hornet" name="hornetserver-web" rev="3.3.0" conf="compile-
&gt;master,libraries;provided;runtime-&gt;master,libraries;test" transitive="true">
<artifact name="hornetserver-web" type="jar"/>
</dependency>
<dependency org="org.codehaus.sonar-plugins" name="sonar-ant-task" rev="1.3" conf="sonar-&gt;default">
<artifact name="sonar-ant-task" type="jar"/>
</dependency>
<dependency org="net.sourceforge.cobertura" name="cobertura" rev="1.9.4.1" conf="cobertura-&gt;default">
<artifact name="cobertura" type="jar"/>
</dependency>
</dependencies>
</ivy-module>
```

Les dépendances présentes peuvent être modifié, tout comme il est possible d'en ajouter :

```
<dependency org="org.apache.commons" name="oro"
                rev="2.0.8" conf=" default" transitive="false">
                <artifact name="oro" type="jar" />
</dependency>
```

Attention, le framework fourni déjà un certain nombre de dépendances. En cas d'ajout de nouvelle dépendance, il faudra veiller à ne pas ajouter de doublon ou de framework identique avec des numéros de version différents, ce qui poserait des problèmes sur les développements.

3.7.8.5 Récupération des dépendances et construction

Afin de profiter de la gestion des dépendances par ivy, il est nécessaire d'ajouter des tâches ant au projet qui permettront de récupérer les artefacts dont dépend le projet.

La récupération des dépendances s'effectue en lançant la tâche Ant « importHornetLibs ».
Le déploiement dans le repository est assuré par la tâche « publish-project-all-release » du fichier « build.xml ».

3.7.8.6 Gestion des dépendances par Eclipse

Eclipse propose une gestion des librairies assez rigide. Or, afin de rendre l'utilisation d'Ivy possible dans Eclipse il existe deux moyens :

- Une tâche Ant, couplé avec des propriétés de *build* des projets
- Un plugin Eclipse

Le plugin Eclipse a pour le moment été écarté pour des problèmes de versionning.
En effet, le plugin ivyDE ne support pour le moment que la version 2.1.0 d'Ivy. Or la version utilisée est la 2.3.0.

L'écriture d'une tâche Ant diffère quelque peu suivant le type de projet. Un projet web ne stock pas ses librairies dans le même répertoire qu'un projet java classique. De même que la définition dans Eclipse au niveau du fichier de configuration (*.classpath*) est différente.

3.7.8.6.1 Projet web

3.7.8.6.1.1 Tâche Ant

Lancer la tâche ant « *importHornetLibs* ».

3.7.8.6.1.2 Classpath Eclipse

Vérifier que votre projet contient bien ces deux lignes dans le fichier *.classpath* :

```
    <classpathentry kind="con"
path="org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/Ja
vaSE-1.6"/>
```

```
 <classpathentry kind="con"
path="org.eclipse.jst.server.core.container/org.eclipse.jst.server.tomcat.runtimeTarget/Apache Tomcat
v6.0"/>
    <classpathentry kind="con" path="org.eclipse.jst.j2ee.internal.web.container"/>
```

3.7.8.6.1.3 Launch Eclipse

Ensuite, il faut créer le fichier de launch Eclipse qui s'appuiera sur la tâche Ant à générer.
Clique droit sur le projet Eclipse

- Properties
- Menu builders
  - Cliquer sur « new… » -> « Ant builder »
  - Dans l'onglet « Main », sélectionner le fichier *build.xml* du projet
  - Dans l'onglet « target » :
    - Partie « after a clean », sélectionner la tâche Ant « *importHornetLibs* »
    - Partie « Manual Build », sélectionner la tâche Ant « *importHornetLibs* »
  - Valider

Réaliser un *clean* du projet : la tâche Ant doit se lancer, les logs sont présents dans la console.

3.7.8.6.2Projet Java

3.7.8.6.2.1 Tâche Ant

La tâche « *generation-classpath* » créé le répertoire de librairies s'il n'est pas présent, puis rapatrie celles-ci dans ce répertoire.
Pour finir, elle met à jour le classpath du projet Eclipse.

3.7.8.6.2.2 Le fichier de transformation

Il s'agit d'un fichier xsl sur lequel s'appuie la tâche Ant pour créer un nouveau fichier « *.classpath* » pour eclipse.
Il doit être déposé dans le répertoire « devlib » du projet.

Voici une configuration par défaut, qu'il est possible d'adapter.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
  <!ENTITY tab    "&#009;">
]>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" encoding="UTF-8" standalone="yes"
        indent="yes" />
 <xsl:output method="xml" />
 <xsl:strip-space elements="*"/>
 <xsl:template match="/">
        <classpath>
                <classpathentry kind="src" path="src"/>
                <classpathentry kind="src" path="tst"/>
                <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE CONTAINER"/>
                <classpathentry kind="output" path="bin"/>

                <xsl:apply-templates />
        </classpath>
 </xsl:template>
 <xsl:template match="dependency">
        <xsl:if test="contains(@conf,'compile') ">
        <classpathentry kind="lib">
                <xsl:attribute name="path">lib/compile/<xsl:value-of select="@name" />-<xsl:value-of
select="@rev" />.jar</xsl:attribute>
        </classpathentry>
        </xsl:if>
 </xsl:template>

</xsl:stylesheet>
```

Si le projet Java utilise les librairies du framework, celle-ci devront être ajouté manuellement dans le fichier xsl.

3.7.8.6.2.3 Launch Eclipse

Ensuite, il faut créer le fichier de launch Eclipse qui s'appuiera sur la tâche Ant à générer.
Clique droit sur le projet Eclipse

- Properties
- Menu builders
  - Cliquer sur « new… » -> « Ant builder »
  - Dans l'onglet « Main », sélectionner le fichier *build.xml* du projet
  - Dans l'onglet « target » :
    - Partie « after a clean », sélectionner la tâche Ant « *generation-classpath* »
    - Partie « Manual Build », sélectionner la tâche Ant « *generation-classpath* »
  - Valider

Réaliser un *clean* du projet : la tâche Ant doit se lancer, les logs sont présents dans la console.

3.8 Gestion des Tests

3.8.1 Principes Généraux

- Créer un *source-folder* « tst » à la racine du projet.
- Créer une classe de test par classe Action ou Service testée. La classe est suffixée par Test.
- Créer une méthode par scénario de test envisagé (toujours prévoir au moins un cas normal et un cas d'erreur)
- En cas de correction d'anomalie, toujours créer un test unitaire reproduisant l'anomalie avant de la corriger.

3.8.2 Test de classe Action

```
public class ListTest extends BaseStrutsTestCase<List> {

    @Before public void initDB() {
        DataSource ds = (DataSource) applicationContext.getBean("dataSource");
        new JdbcTemplate(ds).execute("DELETE FROM PRODUCT");
    }

    @Test  public void testScenario() throws Exception {

        tstInitAction( "/dyn/protected/products", "add");
        tstRequestParameter(
            "product.productLabel", "aaa");
        tstRequestParameter(
            "product.productPrice", "30.55");
        tstRequestParameter(
            "product.productValidity", "20/01/2010");
        tstExecute();
        Assert.assertEquals(Action.SUCCESS, tstGetResult());

        tstInitAction("/dyn/protected/products", "list");
        tstExecute();
        Assert.assertEquals(Action.SUCCESS, tstGetResult());
        Assert.assertEquals(1, tstGetAction().getList().size());

        Product p = tstGetAction().getList().get(tstGetAction().getList().size()-1);
        tstInitAction("/dyn/protected/products", "remove");
        tstRequestParameter(
            "product.productId", ""+p.getProductId());
        tstExecute();
        Assert.assertEquals(Action.SUCCESS, tstGetResult());
    }

    @Test  public void testErreurs() throws Exception {
```

```
        tstInitAction("/dyn/protected/products", "remove");
        tstRequestParameter("product.productId", "-1");
        tstExecute();
        Assert.assertEquals("errorReport", tstGetResult());//se traduit par une erreur technique


        tstInitAction("/dyn/protected/products", "add");
        tstRequestParameter(
            "product.productLabel", "aaa");
        tstExecute();
        Assert.assertEquals(Action.INPUT, tstGetResult());
        Assert.assertEquals(2,
            tstGetAction().getFieldErrors().size());

        tstInitAction("/dyn/protected/products", "add");
        tstRequestParameter(
            "product.productLabel", "aaa");
        tstRequestParameter(
            "product.productPrice", "30.55");
        tstRequestParameter(
            "product.productValidity", "20/01/2000");
        tstExecute();
        Assert.assertEquals(Action.INPUT, tstGetResult());
        Assert.assertEquals(1,
            tstGetAction().getFieldErrors().size());

    }

}
```

3.8.3 Test de service

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"spring-appContext-common.xml"})
public final class ProductServiceTest {

    /** auto-injection par Spring */
    @Autowired
    private ProductService service;

    @org.junit.Test
    public void totoList() throws Exception {
        List<Product> utils = service.list();
        Assert.assertEquals(1000, utils.size());
    }
}
```

3.9 Fonctionnalités

3.9.1 Menu, fil d'Ariane et plan du site dynamiques

Il est possible de charger dynamiquement le menu, le fil d'Ariane ainsi que le plan du site en renseignant un fichier *menu.xml*.

En voici un exemple :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<pagelet>
    <root
        id="root"
        href="/dyn/protected/accueil.html">
    </root>
    <menu>
        <menu-item
            id="menu.accueil"
            href="/dyn/protected/accueil.html"
            visibleDansFilAriane="false">
        </menu-item>
        <menu-item
            id="menu.partenaires">
            <menu-item
                id="menu.partenaires.recherche"
                href="/dyn/protected/partenaire/formRecherche.html">
                <menu-item
                    id="menu.partenaires.fichePartenaire"
                    visibleDansMenu="false" visibleDansPlan="false">
```

```
                </menu-item>
                <menu-item
                    id="menu.partenaires.ajoutPartenaire"
                    visibleDansMenu="false" visibleDansPlan="false">
                </menu-item>
                <menu-item
                    id="menu.partenaires.editionPartenaire"
                    visibleDansMenu="false" visibleDansPlan="false">
                </menu-item>
            </menu-item>
        </menu-item>
        <menu-item
            id="menu.administration">
            <menu-item
               id="menu.administration.secteurs"
               visibleDansFilAriane="false">
               <menu-item
                   id="menu.administration.secteurs.liste"
                   href="/dyn/protected/administration/secteur/liste.html">
                   <menu-item
                       id="menu.administration.secteurs.ajoutSecteur"
                       visibleDansMenu="false" visibleDansPlan="false">
                   </menu-item>
                   <menu-item
                       id="menu.administration.secteurs.editionSecteur"
                       visibleDansMenu="false" visibleDansPlan="false">
                   </menu-item>
               </menu-item>
            </menu-item>
        </menu-item>
        <menu-item
            id="menu.accessibilite"
            href="/dyn/protected/accessibilite.html"
            visibleDansMenu="false" visibleDansPlan="true">
        </menu-item>
        <menu-item
            id="menu.planSite"
            href="/dyn/protected/planSite.html"
            visibleDansMenu="false" visibleDansPlan="false">
        </menu-item>
    </menu>
</pagelet>
```

Ce fichier est constitué de différentes balises :

- root : contient les informations propres à la page racine du site
- menu : contient l'arborescence du site
- menu-item : contient les informations d'un nœud du site

La balise root contient deux attributs obligatoires qui sont :

- id : identifiant du nœud
- href : chemin vers la page liée au nœud

La balise menu-item peut avoir ces deux attributs ainsi que :

- visibleDansMenu : booléen indiquant si ce nœud est visible ou non dans le menu
- visibleDansPlan : booléen indiquant si ce nœud est visible ou non dans le plan du site
- visibleDansFilAriane : booléen indiquant si ce nœud est visible ou non dans le fil d'Ariane

Pour cette balise, seul l'attribut « id » est obligatoire, les booléens prennant la valeur « true » par défaut.

3.9.2 Upload / Download de fichiers

3.9.2.1 Upload

Coté client : le formulaire doit être de type *multipart* et comporter au moins un champ *input* de type *file*

```
<form enctype="multipart/form-data" action="monaction.xml" method="post">
  …
  Fichier 1 : <input type="file" name="myDoc" />
  Fichier 2 : <input type="file" name="myDoc" />
  …
</form>
```

Côté serveur : l'action doit implémenter ces méthodes :

```
public void setMyDoc(File[] myDocs)
public void setMyDocContentType(String[] contentTypes)
public void setMyDocFileName(String[] fileNames)
```

NB : les crochets [] sont facultatifs si un seul fichier est uploadé

Explication : Struts 2 utilise *commons-fileupload* pour *parser* les requêtes *multipart*.
Les fichiers uploadés sont enregistrés temporairement sur disque (nom uniques par session). L'accès au flux de données se fait via les objets de type java.io.File. Les *contentType* et *fileName* correspondent au type et au nom de fichier coté client.
Pour accéder au flux de donnée en Java :

```
InputStream is = new BufferedInputStream(new FileInputStream(myDocs[0]));
try{
   //… traitement…
} finally {
   is.close(); // et surtout on n'oublie pas de le fermer!
}
```

Pour récupérer le flux dans d'un tableau de *byte* :

```
byte[] tab = new byte[is.available()] ;
is.read(tab);
```

Paramétrage par défaut (modifiable dans *struts.xml*):

Propriété	Description	Valeur par défaut
struts.multipart.parser	*Parser* de requêtes *multipart*	Jakarta (Commons FileUpload)
struts.multipart.saveDir	Répertoire temporaire pour les fichiers uploadés	javax.servlet.context.tempdir tel que défini par Tomcat
struts.multipart.maxSize	Quantité uploadée en octet. Si plusieurs fichiers, s'applique au total des fichiers	2 Mo

3.9.2.2 Intégration ClamAV

Créer et mettre le fichier de propriétés suivant dans les sources du projet : /src
Fichier de propriété : clamav.properties

Propriété	Description	Exemple
antivirus.serverAddress	Adresse du serveur ClamAV	10.2.0.233
antivirus.serverPort	Port du service ClamAV	3310
antivirus.timeout	Temps maximum de réponse du service	1000
antivirus.connectTimeout	Temps maximum de connexion au service	1000

Exemple :

```
antivirus.serverAddress=10.2.0.233
antivirus.serverPort=3310
antivirus.timeout=1000
antivirus.connectTimeout=1000
```

Modifier la configuration Spring du projet afin de créer le service ClamAV pour les *upload*.

Configuration Spring : déclaration du *PropertyPlaceholderConfigurer* et du service *ClamAVCheckService*.

```
<bean id="clamavPropertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location" value="classpath:clamav.properties" />
        <property name="placeholderPrefix" value="$clamav{" />
</bean>

<bean id="clamServiceCheck" class="hornet.framework.clamav.service.ClamAVCheckService">
        <property name="clamAVServer" value="$clamav{antivirus.serverAddress}" />
        <property name="clamAVPort" value="$clamav{antivirus.serverPort}" />
        <property name="timeout" value="$clamav{antivirus.timeout}" />
        <property name="connectTimeout" value="$clamav{antivirus.connectTimeout}" />
</bean>
```

Ensuite il faut injecter le service dans la classe :

```
<bean id="MyUploadService" class="fr.gouv.diplomatie.appli.business.service.MyUploadServiceImpl" >
            <constructor-arg ref="clamServiceCheck" index="0"/>
</bean>
```

Le service est injecté dans le constructeur de la classe *MyUploadServiceImpl*.

Appel et traitement de la réponse au service anti-virus :

```
ClamAvResponse response = this.clamAVCheckService.checkByStream(myFileToCheck);
switch (response.getCategorieReponse()) {
                case NO_VIRUS:
                    //traitement
  break;
                case VIRUS:
                    //traitement
  break;
                case TIMEOUT:
                  //traitement
  break;

                case NO_SERVICE:
                    //traitement
  break;
                default:
                    return false;
                  break;
}
```

ClamAvResponse peut renvoyer d'autres valeurs :

- STATS, VERSION

La réponse peut être récupérée ainsi :

- `response.getResponse();`

Les réponses possibles sont les suivantes :

- 1: stream: OK\n
- 1: stream: KO FOUND XXX\n
- Null

Le cas *null* intervient lors d'un retour de type SERVICE_KO.

3.9.2.3 Simulateur ClamAV

Il est possible que des applications soient déployées avec le contrôle antivirus sans pour autant que celui-ci soit actif.

Ceci peut être notamment le cas pour des applications intranet.

Les applications utilisant le mécanisme de paramètrage des nomenclatures devront mettre dans leur référentiel deux indicateurs :

- *serviceClamavActifIntranet*
- *serviceClamavActifInternet*

Les valeurs de chaque clé étant *true/false*.

Pour cela, il faut alimenter le fichier *configuration.xml* des projets de type PARAM :

```
<configuration>
    <nomenclature
        idNomenclature="CLAM_AV" idTableSQL="CLA_ID" nomTableSQL="R_CLA_CLAMAV"
        libelle="Service ClamAV" acces="111" visible="true">
        <colonne libelle="Identifiant" nomColonneSQL="CLA_ID" acces="R" />
        <colonne libelle="Libellé" nomColonneSQL="CLA CODE" />
        <colonne libelle="Actif" nomColonneSQL="CLA_ACTIF" />
    </nomenclature>
</configuration>
```

Dans l'application :

- Créer le BO
- Créer le DAO
- Créer le SqlMap
- Créer l'interface
- Ecrire le script SQL

Exemple :

BO :

```
public class ClamAVBO {

    /**
     * idendifiant
     */
    private Long identifiant;

    /**
     * libelle
     */
    private String code;

    /**
     * libelle
     */
    private boolean actif;

…


SqlMap :

<sqlMap namespace="R DOM DOMAINE">

  <resultMap class="ClamAVBO"
         id="clamavResult">
         <result property="identifiant" column="CLA_ID" />
         <result property="code" column="CLA CODE" />
         <result property="actif" column="CLA_ACTIF" />

  </resultMap>
```

```
    <select id="getClamAVActifByCode" parameterClass="java.lang.String" resultMap="clamavResult">
    SELECT
        CLA_ID,
        CLA_CODE,
   CLA_ACTIF
    FROM R_CLA_CLAMAV
    WHERE CLA_CODE = #value#
    </select>
</sqlMap>
```

Exemple d'appel :

```
ClamAVBO clamav = getClamAVActifByCode("serviceClamavActifIntranet");
if(clamav.isActif()){
   ClamAvResponse response = this.clamAVCheckService.checkByStream(myFileToCheck);
   …

}
```

3.9.2.4 Download

Classe Action :

```
byte[] data;
…
this.setContentDisposition("filename=\"monFichier.pdf"\"");
this.setContentType("application/pdf");
this.setInputStream(new ByteArrayInputStream(data));

return SUCCESS;
```

Struts.xml :

```
<action name="export" class="hornet.projet.web.action.Exporter">
        <result name="success" type="stream">
               <param name="contentType">${contentType}</param>
               <param name="inputName">inputStream</param>
               <param name="contentDisposition">
                     ${contentDiposition}
               </param>
               <param name="bufferSize">1024</param>
        </result>
</action>
```

3.9.3 Détection de Type MIME

3.9.3.1 Définition du Type Mime

Un *Internet media type*, à l'origine appelée Type MIME (MIME est l'abréviation de « *Multi-purpose Internet Mail Extensions* ») ou juste MIME ou encore *Content-type*, est un identifiant de format de données sur internet en deux parties.

Les identifiants étaient à l'origine définis dans la RFC 2046 pour leur utilisation dans les courriels à travers du SMTP mais ils ont été étendus à d'autres protocoles comme le HTTP ou le SIP.

En clair les types MIME constituent une façon normalisée de classer les différents types de fichiers sur Internet. Les programmes Internet, comme les serveurs et les navigateurs Web, comprennent tous une liste de types MIME afin de pouvoir transférer les mêmes types de fichiers de la même façon, quel que soit le système d'exploitation qu'ils utilisent.

3.9.3.2 Interface TypeMimeParseur

Une interface est crée dans le Framework Hornet.

Cette interface contient une méthode « *parse()* » qui prend en paramètre un fichier de type *File* et retourne un objet de type *TypeMime*. Cet objet contient un attribut « *nom* » qui contient la valeur du *TypeMime* détecté pour le fichier *File*.

Cette interface permet à Hornet de ne pas être dépendant d'une solution de détermination de *TypeMime*. Pour utiliser un framework de détection de *TypeMime* il suffit de créer une classe implémentant cette interface.

La solution retenue pour Hornet sera l'utilisation du framework Aperture.

3.9.3.3 Utilisation d'Aperture

Aperture est un framework pour faciliter l'obtention d'une version texte d'un fichier quelquesoit son format, il propose également une solution de détermination du type MIME grâce à la classe *MagicMimeTypeIdentifier*.

Aperture nécessite les jars suivants pour fonctionner :

- aperture-core-1.5.0.jar
- aperture-tools-demork-1.0.0.jar
- rdf2go.api-4.7.3.jar
- slf4j-api-1.5.6.jar
- slf4j-log4j12-1.5.6.jar

3.9.3.4 Liste des TypeMime gérés par le framework

Voici la liste des fichiers dont le type Mime est géré par le framework :

- Fichier Image : .jpeg, .png, .gif (animé ou non)
- Document Texte : .txt, .rtf, .pdf
- Document Microsoft Office : Excel (Excel 2K, Excel 2007), Word (Word 95, Word 2K, Word 2007), OpenDocument Word (.docx), OpenDocument Excel (.xlsx)
- Archive : .tar, .tar.gz, .gz, .zip

3.9.3.5 Intégration dans Hornet

L'objet « *TypeMime* » propose aussi différentes méthodes qui retournent vrai ou faux si le type Mime du fichier correspond à une catégorie donnée.

Par exemple la méthode « *isImage()* » retourne *true* si le fichier est de type Image (.gif, .jpeg, .png).

Certaines méthodes utilisent des expressions régulières pour déterminer la catégorie du fichier.

La classe « *ConstantesTypeMime* » contient l'ensemble des types MIME supportés par le framework ainsi que les expressions régulières utilisées pour déterminer la catégorie du fichier par la classe « *TypeMime* ».

A noter que si le fichier analysé par la classe « *ApertureParseur* » contient un type MIME non répertorié par le framework, il est quand même possible de le récupérer par l'attribut « *nom* » de l'objet *TypeMime* retourné par la méthode « *parse()* ». La méthode « *isNotClassified()* » retournera d'ailleurs vrai pour un type MIME non répertorié par le framework.

Une classe de test « *ApertureTest* » contient un test Junit sur l'ensemble des fichiers dont le type MIME est supporté par le framework avec pour chaque fichier un test sur la valeur du type MIME et un test sur la catégorie du fichier.

Exemple de test :

```
/**
 * testJPG
 */
@Test
public void testJPG() {
    System.out.print("testing JPG image...");
    // Recuperation dans typeMime du TypeMime du fichier
    this.genericParseAperture(jpgFile);
    // Comparaison avec Type Mime attendu
```

```
        Assert.assertEquals(typeMime.getNom(), ConstantesTypeMime.IMG_JPEG);
        // Comparaison avec Catégorie attendue
        Assert.assertTrue(typeMime.isImage());
        System.out.print("\n");
    }
```

L'ensemble des classes permettant la gestion est la détection du type MIME sont inclus dans le jar « hornetserver-typemime ».

3.9.3.6 Utilisation dans un projet

Pour utiliser la détection de type MIME dans un projet basé sur le Framework Hornet, il suffit d'intégrer la librairie « hornetserver-typemime ».

L'exemple de code suivant permet de détecter la détection du type MIME :

```
// Declaration du fichier a tester
File fichier = new File();
// Declaration du Parseur
ApertureParseur parser = new ApertureParseur();
// Detection du TypeMime par le Parseur
TypeMime typeMime = parser.parse(fichier);
// Recuperation du typeMime
String nomTypeMime = typeMime.getNom();
```

3.9.4 Reporting & éditique

3.9.4.1 JasperReport

JasperReport est un framework permettant de générer des rapports sous différents formats:

- PDF
- HTML
- XLS
- RTF (et non DOC)
- CSV
- XML
- TXT

3.9.4.1.1Cycle de vie d'un rapport

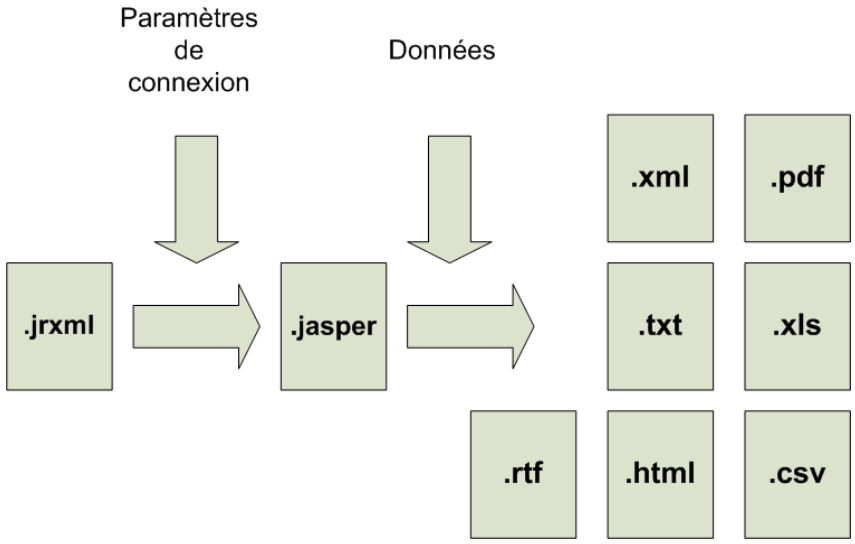


Figure 14 : Cycle de vie d'un rapport Jasper

3.9.4.1.2Architecture

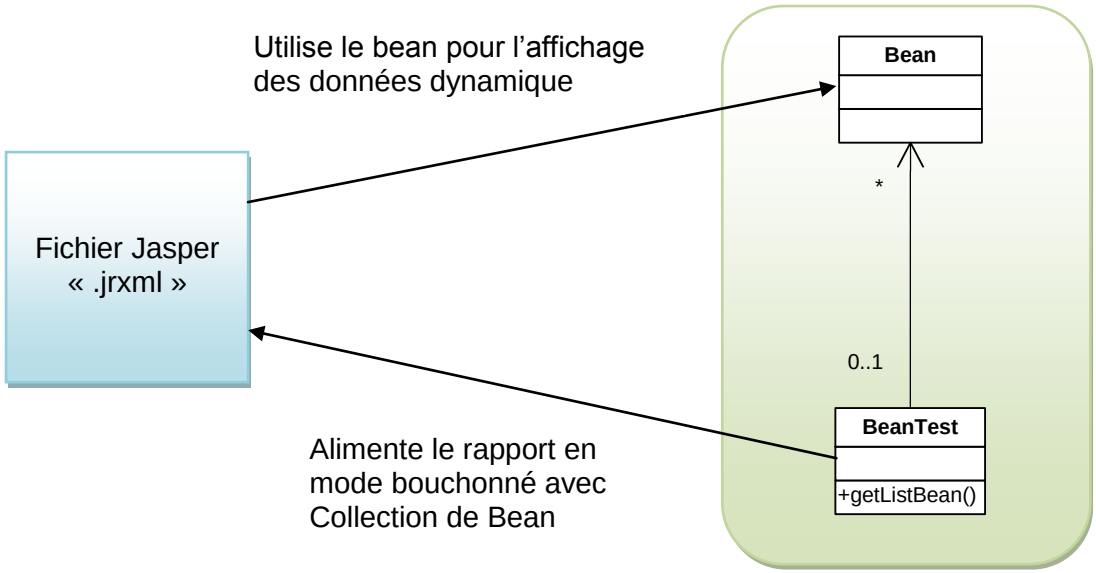


Figure 15 : Bloc création rapport Jasper

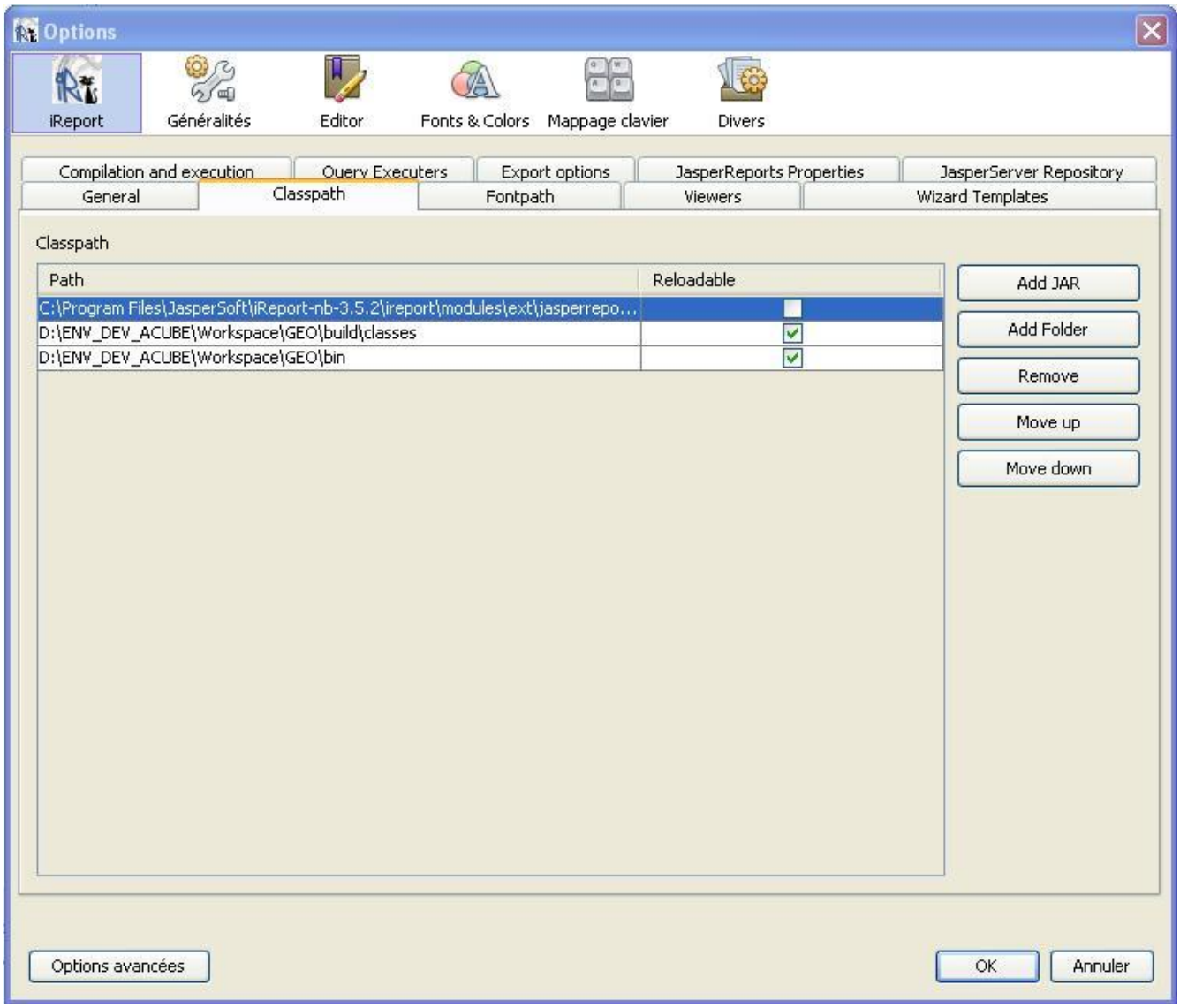
3.9.4.1.3 Création d'un rapport avec iReports

IReport est un outil de type wysiwyg (une interface utilisateur qui permet de composer visuellement le résultat voulu) permettant de créer des fichiers (.jrxml) utilisables ensuite par la librairie JasperReport.

3.9.4.1.3.1 Paramétrage

3.9.4.1.3.1.1 Classpath

Aller dans le menu « Outils/Options », puis allez dans l'onglet « Classpath » :



Il faut ajouter deux répertoires au *classpath*, le premier servira pour pointer sur les *Beans*, et le second pour accéder aux *BeanTest* afin de pouvoir tester le document en mode bouchonné :

- WORKSPACE_ECLIPSE\Projet\build\classes
- WORKSPACE_ECLIPSE\Projet\bin

3.9.4.1.3.2 Création des Beans

Avant de commencer à créer le rapport, il faut tout d'abord créer le *Bean* principal qui alimentera le rapport.

Pour chaque fichier .jasper, devra correspondre un *BeanTest* et *Bean*.

Dans le cas d'un rapport simple (pas de sous-rapport), un seul *Bean* suffira. En revanche, si le rapport à créer est plus complexe et nécessite un découpage et une utilisation de sous-rapport, alors le *Bean* principal contiendra autant de *List* de « BeanSousRapport » qu'il y a de sous-rapport dans le fichier à générer.

3.9.4.1.3.2.1 Bean

Le Bean contient les données qui vont alimenter le rapport que l'on devra créer. Les variables définies dans cet objet seront utilisées au sein du rapport.

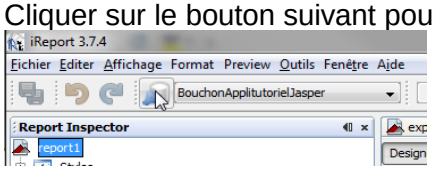
3.9.4.1.3.2.2 Bean Test

Le BeanTest est une classe contenant une méthode statique retournant une *Collection* de *Bean*. Cette classe permet de tester le rapport en mode bouchonné.

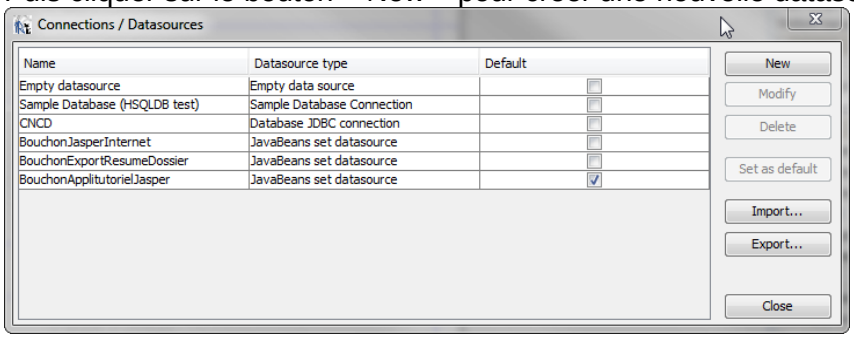
```
public class LigneElectionBOTest {

    public static List<LigneElectionBean> getBeans() {

        List<LigneElectionBean> beans = new ArrayList<LigneElectionBean>();

        LigneElectionBean ligneElectionBO;
        for (int i = 0; i < 100; i++) {
            ligneElectionBO = new LigneElectionBean();
            ligneElectionBO.setSigleOrgane("Sigle Organe");
            ligneElectionBO.setLibelleElection("Libellé Election");
            ligneElectionBO.setNomCandidat("Nom Candidat");
            ligneElectionBO.setEtatElection("Elu");
            ligneElectionBO.setDateElection(new Date());

            beans.add(ligneElectionBO);
        }
        return beans;
    }
}
```

3.9.4.1.3.3 *Création de la datasource (source de données)*

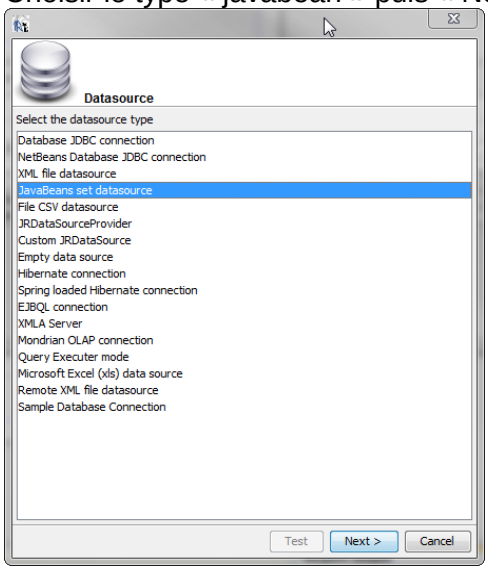
Cliquer sur le bouton suivant pour ouvrir la fenêtre de gestion des datasources :



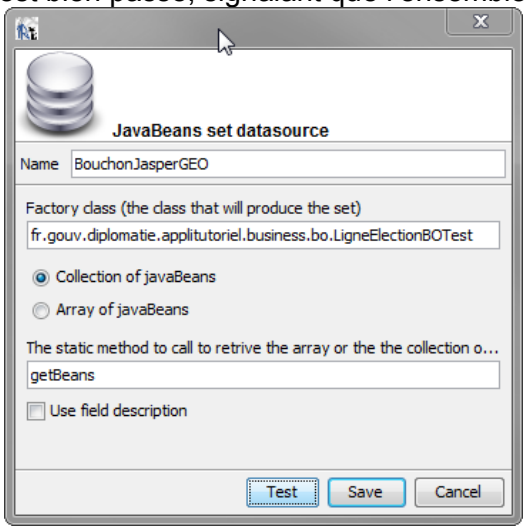
Puis cliquer sur le bouton « New » pour créer une nouvelle datasource :



Choisir le type « javabean » puis « Next » :



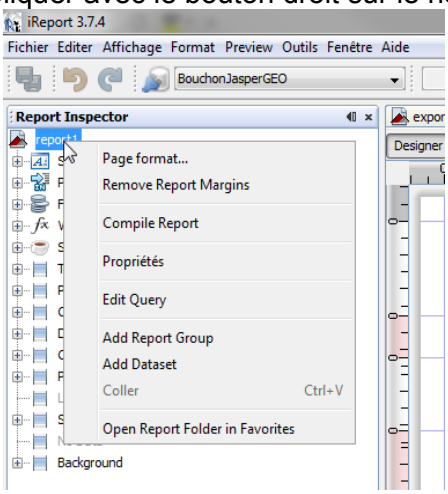
Dans la fenêtre de paramétrage de la datasource mettre **le nom complet de la classe « BeanTest »** (dans notre exemple la classe « `LigneElectionBOTest` ») et indiquer aussi **le nom de la méthode** « static » retournant la collection de beans puis cliquer sur le bouton « Test » pour contrôler le tout et enfin « Save » si le test s'est bien passé, signalant que l'ensemble des paramètres sont bien corrects :



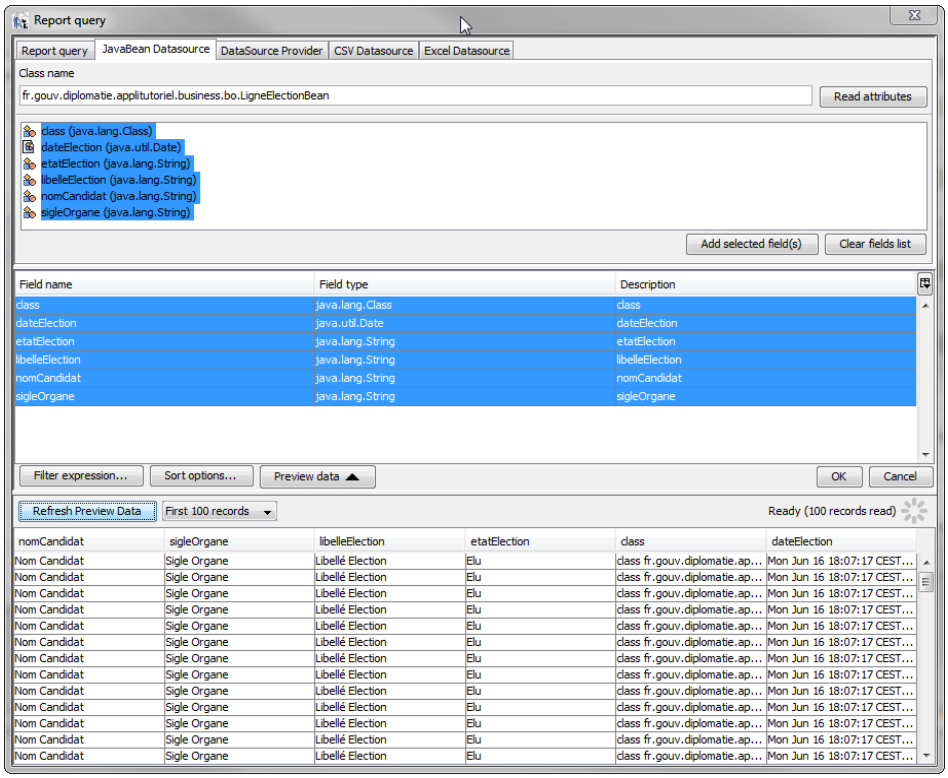
3.9.4.1.3.4 Création de la dataset principale (jeu de données)

La dataset est créée de cette façon :

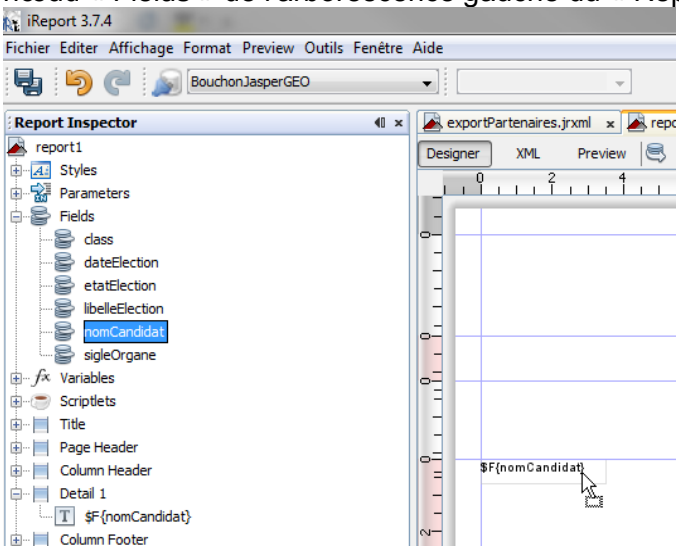
- Cliquer avec le bouton droit sur le nœud racine du report, puis choisir « Edit Query »



- Dans la fenêtre qui s'ouvre nommée « Report query » :
  - Sélectionner l'onglet « JavaBean DataSource »
  - Entrer **le nom complet de la classe « Bean »**
    - « `fr.gouv.diplomatie.applitutoriel.business.bo.LigneElectionBean` » dans notre exemple
  - Cliquer sur le bouton « Read attributes »
  - Sélectionner tous les attributs puis cliquer sur le bouton « Add selected field(s) »
  - (optionnel) il est possible en cliquant sur « Preview data » d'afficher les données de test produites par le bouchon.
  - Enfin cliquer sur « OK »



- On peut alors ajouter les champs dans le report par simple drag&drop (en les prenant depuis le nœud « Fields » de l'arborescence gauche du « Report Inspector »)



3.9.4.2 PDF/FDF

3.9.4.2.1Description

Forms Data Format ou FDF est un format de fichier développé par Adobe basé sur le format PDF et qui permet l'insertion de champs de saisie dans un fichier PDF. Ces champs peuvent être valorisés par un humain ou par un programme afin de produire un rapport final. FDF est plus rapide à mettre en œuvre que Jasper ou XSLT pour générer des PDF à partir de formulaires préexistant ayant une complexité élevée (type Cerfa).

Cependant, FDF impose certaines limites :

- Les tableaux dynamiques ne sont pas gérés
- Impossible de masquer ou d'insérer des parties du document (concaténation à gérer à part)

3.9.4.2.2Mise en oeuvre

1. Générer un fichier PDF/FDF avec un outil Adobe ou avec OpenOffice :

   - Guide Pour OpenOffice : http://documentation.openoffice.org/manuals/userguide3/0215WG3-UsingFormsInWriter.pdf. (Il suffit ensuite d'exporter au format PDF)

2. Créer une action Struts pour la génération, qui implémente l'interface « ***FDFAware*** » :

   - Méthode ***getFdfBean()*** : renvoie un *bean* Java ou une *Map* servant à valoriser les champs du formulaire FDF. Le bean peut très bien être l'action elle-même (« *return this;* »).
     Le nom d'un champ de formulaire doit correspondre à un attribut du *bean* ou a une clé de la *Map*.
     Le nom d'un champ peut être « composé » avec le séparateur « : », par exemple « *personne:nom* ». Dans ce cas le *bean* devra contenir une propriété « *personne* » contenant une propriété « *nom* », accessibles via les « *getter* » associés.
     Tous les types Java peuvent être utilisés. La conversion en chaine de caractère lors de la valorisation utilise la méthode *toString()* de l'objet (donc généralement la *Locale* et un format par défaut).
     Par défaut le type *Boolean* est converti en « Yes » ou « No » pour pouvoir fonctionner avec la case à cocher.
     Par défaut le type *Date* est formaté en « dd/MM/yy ».

   - Méthode ***setUnmergedFields()*** : appelée en fin de génération, pour indiquer à l'action la liste des champs non fusionnés.
     Remarque : les champs non fusionnés sont aussi tracés avec un niveau « WARN »

3. Dans « ***struts.xml*** », pour cette action, utiliser le *result type* « **fdf** ».

   - Renseigner le chemin du modèle PDF/FDF à utiliser dans l'attribut « ***location*** », ou directement dans le nœud.
   - Si le chemin commence par « ***classpath:*** », le modèle doit être situé dans le *classpath* de l'application avec comme emplacement relatif l'action en cours, sinon le modèle doit être situé dans « *WebContent* » ou un sous-répertoire :

```
<action name="cerfa"
    class="hornet.projet.web.action.export.CerfaAction">
    <result type="fdf">classpath:cerfa-cp.pdf</result>
</action>
```

ou

```
<action name="cerfa"
    class="hornet.projet.web.action.export.CerfaAction">
    <result type="fdf">/templates/export/fdf/cerfa-cp.pdf</result>
</action>
```

Remarque : Il est aussi possible d'utiliser la *value-stack* et les variables Struts.

3.9.4.2.3Fusion des images

Pour insérer dynamiquement une image dans le PDF final :

1. créer un champ quelconque dans le formulaire PDF/FDF pour définir la position et la taille de l'image. L'image aura par défaut la taille du rectangle.
2. Dans l'objet de données, créer une propriété liée à ce champ ayant le type « ***FDFImage*** » (classe du framework). Un objet *FDFImage* prend en entrée l'URI de l'image (qui peut être une url http, filesystem ou une ressource Java situé dans le *classpath*), ou directement un tableau de *byte* (cas du stockage en base de données par exemple).

Les formats d'image reconnus sont : GIF, PNG, BMP, TIFF

3.9.4.3 XSLT

3.9.4.3.1Word/Excel

1. Création d'un modèle de document (au format html compatible office 2000/2003/2007, ou le nouveau format xml office) dans une feuille XSLT :
2. On commence par créer le document dans Office ou OpenOffice, puis on exporte au format html/xml
3. On transforme ce document en feuille XSLT (on isole les parties structurantes dans des templates xslt).
4. On modifie la feuille de manière à ajouter les parties dynamiques (qui seront valorisées à partir des données de l'action Struts)
5. Dans *struts.xml*, pour l'action qui génère le document, on utilisera le *result type* « **xslt** ».

3.9.4.3.2PDF

1. Création d'un modèle de document XSL/FO « from scratch ».
2. Prévoir les parties dynamiques pendant la construction du modèle (qui seront valorisées à partir des données de l'action Struts)
3. Dans *struts.xml*, pour l'action qui génère le document, on utilisera le *result type* « **xsltFop** ».

3.9.4.4 CSV

A utiliser uniquement pour des données tabulaires brutes (sans style).

1. Création d'une action Struts qui implémente l'interface *CSVAware* pour la génération
2. Dans *struts.xml*, pour cette action, utiliser le *result type* « **csv** ».

3.9.4.5 Template d'email

3.9.4.5.1Description

Le framework propose une solution de *template* pour les emails basée sur Velocity et Spring.
Velocity utilise le *template* et fusionne celui-ci avec les données métiers (objets Java).

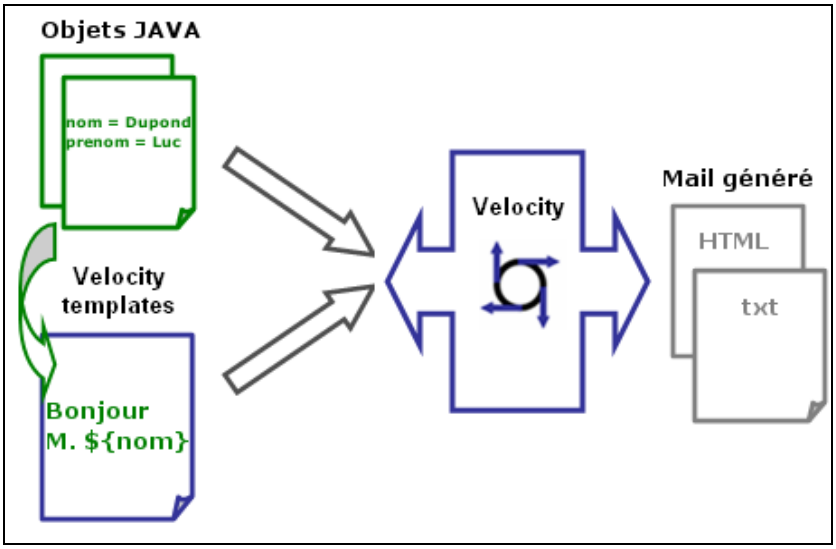


Schéma de fonctionnement de Velocity

3.9.4.5.2Mise en œuvre

1. Créer un fichier nommé « *spring-appContext-mail.xml* » avec au minimum le contenu suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Configuration de Spring pour l'envoi de mail -->
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd
            http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.0.xsd
            http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-
tx-3.0.xsd">

    <bean id="mailPropertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location" value="classpath:mail.properties" />
        <property name="placeholderPrefix" value="$mail{" />
    </bean>

    <bean id="mailServicePropertyConfigurer"
        class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location" value="classpath:mailService.properties" />
        <property name="placeholderPrefix" value="$mailService{" />
    </bean>

    <!-- Service Hornet Mail -->
    <bean id="hornetMailService" class="hornet.framework.mail.MailServiceImpl">
        <constructor-arg ref="mailSender" index="0" />
        <constructor-arg value="$mailService{mail.application.name}" index="1" />
        <constructor-arg value="$mailService{mail.messageid.domain}" index="2" />
    </bean>

    <!-- Classe utilisée pour envoyer les mails -->
    <bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
        <property name="host" value="$mail{mail.smtp.host}" />
        <property name="username" value="$mail{mail.smtp.username}" />
        <property name="password" value="$mail{mail.smtp.password}" />
        <property name="javaMailProperties">
            <props>
                <prop key="mail.debug">$mail{mail.debug}</prop>
            </props>
        </property>
    </bean>

    <!-- Service Mail pour le cas d'utilisation "Usecase" -->
    <bean id="mailUsecaseService"
        class="fr.gouv.diplomatie.monapplication.business.service.MailUsecaseServiceImpl">
        <constructor-arg ref="hornetMailService" index="0" />
        <constructor-arg value="$mailService{mail.usecase.object}" index="1" />
        <constructor-arg value="$mailService{mail.usecase.body}" index="2" />
        <constructor-arg value="$mailService{mail.usecase.to}" index="3" />
    </bean>

</beans>
```

Avec la configuration ci-dessus, le projet dispose d'un service « *mailUsecaseService* » contenant :

- un objet *hornetMailService*, permettant l'envoi de mail tout en respectant les bonnes pratiques pour combattre le spam.
- L'objet du message
- Le modèle Velocity pour le corps du message
- Les destinataires du message

2. Créer le service Java qui doit envoyer des emails.
    a. Créer l'interface correspondant à la configuration Spring avec une méthode métier « envoyerMail » :

```
public interface MailUsecaseService {

    void envoyerMail(String nom, String prenom, String fromAddress, String message);
}
```

Pour l'implémentation, la seule méthode à exécuter pour fusionner le template avec les données métier, puis pour envoyer le mail proprement dit est la méthode « envoyerDepuisModele » de la classe « hornet.framework.mail.MailService » :

```
void envoyerDepuisModele(final String expediteur, final String sujet, final String messageTemplate, final
Map<String, Object> paramMap, final String... destinataires)
```

Techniquement l'utilisation de cette classe permet d'implémenter notamment le point suivant pour le respect des bonnes pratiques pour combattre le spam :

- Ajout d'un MessageID au format :
  - UTC + '.' + UUID + '.' + Nom de l'application @diffusion.diplomatie.gouv.fr
  - Exemple : MY3JZE.52296379D1344BB1A9CCCE012410F1C7.CEF@diffusion.diplomatie.gouv.fr

b. Créer le service implémentant l'interface ci-dessus :

```
package fr.gouv.diplomatie.monapplication.business.service;

/**
 * Implementation pour l'envoi d'un mail pour le cas d'utilisation « Usecase ».
 */
public class MailUsecaseServiceImpl implements MailUsecaseService {

    /**
     * JavaMailSender
     */
    private final MailService mailService;

    /**
     * L'Objet du message
     */
    private final String objetMail;

    /**
     * Le Template du Corps du message
     */
    private final String corpsMail;

    /**
     * Les destinataires du mail.
     */
    private final String[] destinataires;

    /**
     * Constante NoReply
     */
    private static final String NO_REPLY = "NoReply";

    /**
     * Constructeur
     */
    public MailContactServiceImpl(
                final MailService mailService, final String objetContactMail,
                final String corpsContactMail, final String[] destinataires) {

        this.mailService = mailService;
        this.objetContactMail = objetContactMail;
        this.corpsContactMail = corpsContactMail;
        this.destinataires = destinataires.clone();
    }

    /** {@inheritDoc} */
    @Override
    public void envoyerMail(final String nom, final String prenom, final String fromAddress,
                final String message) {

        try {
            final Map<String, Object> params = new HashMap<String, Object>();
            params.put("nom", WordUtils.capitalize(nom));
            params.put("prenom", WordUtils.capitalize(prenom));
            params.put("corps", message);
```

```
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy 'à' HH:mm:ss");
            params.put("date", sdf.format(new Date()));

            params.put(MailServiceImpl.SMTP_HEADER_REPLYTO, NO_REPLY);
            params.put(MailServiceImpl.SMTP_HEADER_CC, "destinataire@mail.com");

            mailService.envoyerDepuisModele(fromAddress, this.objetMail,
                this.corpsMail, params, this.destinataires);

        } catch (final Exception e) {
            …
        }

    }

}
```

Dans cet exemple, les champs d'entete ReplyTo et Cc sont par exemple valorisés pour envoyer le message en copie à un destinataire et pour interdire au destinataire du mail de pouvoir répondre au message (cas d'un envoi par automate sur une boite non relevée).

3. Créer le Template Velocity du message proprement dit. Il est conseillé de sauvegarder ce template dans un fichier de ressource (sous src/config) avec l'extension « .vm » :

```
#set($corps = $corps.replaceAll("\n", "<br />"))
<p>Bonjour,</p>
<p>L'utilisateur $prenom $nom souhaite vous contacter.
Voici le contenu de sa demande :</p>
  <blockquote>
    <i>$corps</i>
  </blockquote>
<p>Envoyé le : $date<br/>
Par : <a href="$applicationUrl">$applicationName</a></p>
```

A noter qu'il n'est pas nécessaire d'indiquer les tags <html> ou <body>. Ceux-ci sont automatiquement ajoutés au moment de l'envoi du message. Pour en savoir plus sur la syntaxe, consulter le guide d'utilisation disponible en français sur le site de Velocity : http://velocity.apache.org/engine/releases/velocity-1.6.2/translations/user-guide_fr.html

Un exemple d'envoi complet de mail est également disponible dans l'applitutoriel. Elle implémente un classique formulaire de contact.

3.9.5 Ajout d'un CAPTCHA au projet

Un Captcha (**C**ompletely **A**utomated **P**ublic **T**est to tell **C**omputers and **H**umans **A**part) est un test permettant de différencier de manière automatisée un humain d'un ordinateur, lors du remplissage de données de formulaire d'inscription sur des sites par exemple.
C'est un moyen technique repondu pour lutter contre des robots malveillants.

Cependant l'utilisation de captcha n'est pas compatible avec les principes d'accessibilité. Si l'intégration d'un composant de ce type est nécessaire, consulter la documentation Hornet.

3.10 Performances et bonnes pratiques

3.10.1 Général

- **Ne jamais oublier le premier principe, aphorisme de Donald Knuth : « Premature Optimization is the root of all evil ».**
- **Ne pratiquer a priori que les optimisations peu couteuses en développement et les bonnes pratiques suivantes (qui deviendront des réflexes)**

- Ne pas utiliser les classes *Vector* et *Hashtable* qui utilisent de manière cachée la synchronisation Java. Utiliser *ArrayList*, *HashMap*, *TreeMap*, *HashSet*, … Si la collection DOIT être partagée entre plusieurs *Threads*, utiliser *Collection.synchronizedCollection(…)*
- Sauf si vous savez exactement à quoi ça sert, n'utilisez pas le mot clé « *synchronized* »
- Seuls les calculs de montant doivent se faire en *BigDecimal* (et non *Double* ou *Float* pour éviter les erreurs d'arrondi). Pour les identifiants de base de données, le type *Long* suffit.
- Tirer partie du « *Just In Time compiler* » de la JVM. Le JIT transforme le code Java appelé fréquemment en code natif.

```
public void maFonction() {
    for(int i=0 ; i<100000 ; i++) {
        //traitement
    }
}
```

en

```
public void maFonction() {
    for(int i=0 ; i<100000 ; i++) {
        maFonctionUnitaire(i) ;
    }
}

public void maFonctionUnitaire (int i) {
        //traitement
    }
```

3.10.2 Couche Web

1. Utiliser les JSP au lieu des *templates* XSL (conso mémoire et temps de réponse plus faible), surtout si les données à renvoyées au client sont importantes.
2. Toute requête http de mise à jour devrait également retourner les données permettant de rafraichir le formulaire (plutôt que de renvoyer le flux « success »). Cela évite au client de devoir appeler d'autres flux pour se rafraichir. Pour ce faire, coté serveur, on devra utiliser les techniques de réutilisation d'action et de JSP (chainage d'action, inclusion de JSP).
3. Limiter le flux XML d'un tableau à 500Ko non compressé. Penser à définir avec la MOA une limite du nombre de lignes retournée par requête (<1000). Cf. Pagination.
4. Faire attention à l'utilisation des tags Struts dans les itérations JSP pour les listes volumineuses.

   Ex : cas du tag s:a qui réduit très fortement les performances par rapport à l'utilisation du tag html a.

   Pour 300 éléments, 7s de temps de réponse sur la page avec tag s:a, moins d'une seconde en html.

3.10.3 Couche Service

- La signature (arguments et type de retour) doit être la plus abstraite possible. En particulier, les collections (*ArrayList*, *HashMap*) devront êtres manipulés sous forme de types abstraits *java.util.Collection*, *java.util.List* ou *java.util.Map*, et non en type concret.

3.10.4 Couche DAO

- Cibler les **requêtes optimisées sur les fonctionnalités les plus utilisées**… Le N+1 select sera souvent acceptable, et bien moins couteux à développer.
- On peut récupérer les associations 1-1 et 1-N par une seule requête avec jointure et mapper le résultat sur un BO (via plusieurs *resultMap* MyBatis).
- Ne rapporter qu'une collection rattachée à un objet par requête (*left join*). Si un objet possède plusieurs collections, il est plus optimal de récupérer chaque collection séparément.

- Pour les traitements par lots, on pensera à utiliser la pagination lors de la lecture des objets à traiter pour optimiser la mémoire, et les Batchs JDBC (via MyBatis) pour optimiser les écritures.

3.10.5 Base de données

- Pas de procédure stockée, sauf si problème de performance en exploitation qui impose d'en créer.

3.10.6 Horloges et horodatage

Il est fréquent d'avoir à horodater des étapes d'un traitement.

Certaines règles sont importantes à suivre pour éviter les incohérences, vu que les horloges des différents serveurs (application, base de données …) peuvent être légèrement décalées (bien que les serveurs de production soient synchronisés via le protocole NTP).

Les bonnes pratiques sont :

- Le serveur d'application est le serveur de référence pour les horodatages,
- Il ne faut pas utiliser une échelle de temps inférieure à la milliseconde,
- Il faut préférer un ordre d'étape à un ordre chronologique.

3.10.7 Externalisation des CSS et Javascript

Veiller à ce que les codes CSS et JavaScript ne soient pas embarqués dans le code de la page, à l'exception d'éventuelles variables de configuration pour les objets JavaScript ou messages d'erreur par exemple.

En effet, si vous incluez du code CSS ou JavaScript dans le corps de la page, alors il doit être transféré pour chaque (re)chargement de la page, ce qui augmente le volume de données transmises. En revanche, si les codes CSS et JavaScript sont inclus dans leurs propres fichiers, le navigateur peut les stocker dans son système de cache local afin de ne pas les redemander systématiquement.

Des exemples d'externalisation javascript sont disponibles dans l'applitutoriel hornet.

3.10.8 Mode Combine

3.10.8.1 Présentation

Ce mode permet de combiner et de minimiser les ressources JavaScript et CSS en paquetage moins finement découpés, augmentant ainsi la vitesse de chargement des pages.

L'objectif principal est de réduire le nombre de requêtes http produites par le navigateur pour le chargement d'une page donnée.

Ce type d'optimisation des performances fait partie des « best practices » inclues dans la catégorie « Minimizing round-trip times — reducing the number of serial request-response cycles » établie par Google pour l'obtention d'un web plus rapide.

3.10.8.2 Mode par défaut

L'utilisation du mode Combine est activée par défaut à partir de la version 3.6.1 d'Hornet. Ceci permet de faire bénéficier par défaut aux applications des gains de performance induits par l'utilisation de ce mode d'optimisation.

Le combo par défaut utilisé est le combo « normal ».

Toutefois, il reste possible de le désactiver explicitement (via comboType=none)

3.10.8.3 Choix du combo

Le choix du combo est réalisé par le paramètre « comboType » du fichier « hornet.properties ».

Les valeurs possibles pour ce paramètre sont :

- **basic** : choix du combo JS “basic” et activation du combo CSS du thème « themeName » (pour plus d’info cf § suivant)
- **normal** : choix du combo JS “normal” et activation du combo CSS du thème « themeName » (pour plus d’info cf § suivant)
- **full** : choix du combo JS “full” et activation du combo CSS du thème « themeName » (pour plus d’info cf § suivant)
- **none** : désactivation du mode combo (CSS et JS)

La valeur par défaut appliquée (quand le paramètre « comboType » est vide ou non présent) est « normal ».

3.10.8.3.1 Combos JS

Les combos JS sont au nombre de 3 : « basic », « normal, « full ». Techniquement, on trouve un module hornet par combo.

Combo JS	Nom module hornet	Description
Basic	hornet-xcombine-js-basic	Contient • Tous les fichiers JS Hornet • Tous les fichiers JS Gallery utilisés par hornet • **Les modules YUI de base**
Normal	hornet-xcombine-js-normal	Contient • Tous les fichiers JS Hornet • Tous les fichiers JS Gallery utilisés par hornet • **Les modules YUI permettant de faire fonctionner l’applitutoriel**
Full	hornet-xcombine-js-full	Contient • Tous les fichiers JS Hornet • Tous les fichiers JS Gallery utilisés par hornet • **Tous les modules YUI**

Note : Si l’application utilise un module YUI qui n’est pas présent initialement dans le combo, le module est chargé dynamiquement par le YUI Loader de la même manière que lorsque le mode Combine n’est pas activé.

3.10.8.3.2 Combos CSS

Les combos CSS sont disponibles sous forme de thèmes (1 combo par thème). Ils sont présents dans le répertoire `/hornetclient/<X.Y.Z>/themes/hornet-skin-<AAAA-x.y.z>` sur le serveur de framework. Leurs noms techniques sont simplement postfixés par « -xcombine ».

Thème	Nom skin « classique »	Nom skin « combo »
Addulact (defaut)	hornet-skin-defaut	hornet-skin-defaut**-xcombine**
MAEDI Intranet (diplonet)	hornet-skin-diplonet[-x.y.z]	hornet-skin-diplonet**-xcombine**[-x.y.z]

MAEDI Internet (francediplo)	hornet-skin-francediplo[-x.y.z]	hornet-skin-francediplo-**xcombine**[-x.y.z]

Le choix du skin est opérée automatiquement à partir des propriétés du fichier « hornet.properties » :

- themeName
- themeVersion
- comboType

3.10.9 Listener de journalisation d'évènements.

3.10.9.1 Présentation

Quatres listener permettant une journalisation d'évènements particuliers ont été ajoutés.
Ces listeners permettent de journaliser les évènements suivants :

- Démarrage/arret de l'application (**ApplicationContextListener**)
- Création/déstruction d'une session utilisateur (**ApplicationSessionListener**)
- Journalisation des erreurs d'authentification(**ApplicationAuthenticationFailureBadCredentialListener**)
- Journalisation des authentifications réussies (**ApplicationAuthenticationSuccessListener**)

Chacun de ces listener possède des éléments de journalisation par défaut, ainsi que des méthodes à implémenter afin d'adapter ces journalisation aux applications utilisant le framwork.

3.10.9.2 Ajout des classes d'applications

3.10.9.2.1 ApplicationSessionListener

Voici la classe de base à ajouter dans le package de listener pour la journalisation des création/destruction de session :

```
public final class ApplicationSessionListener extends HornetSessionListener {
    @Override
    protected void createSessionActions() {
        // Pas d'actions spécifiques
        Logger.getLogger(this.getClass()).info("Pas d'action spécifique");
    }
    @Override
    protected void destroySessionActions() {
        // Pas d'actions spécifiques
        Logger.getLogger(this.getClass()).info("Pas d'action spécifique");
    }
}
```

Des actions spécifiques peuvent être réalisées dans les méthodes **createSessionActions** et **destroySessionActions** afin de s'adapter aux applications.

3.10.9.2.2 ApplicationContextListener

Voici la classe de base à ajouter dans le package de listener pour la journalisation des démarrage/arrets de l'application :

```
public final class ApplicationContextListener extends HornetContextListener {
    @Override
    protected void initContextActions() {
        // Pas d'actions spécifiques
        Logger.getLogger(this.getClass()).info("Pas d'action spécifique");
    }
    @Override
    protected void destroyContextActions() {
        // Pas d'actions spécifiques
        Logger.getLogger(this.getClass()).info("Pas d'action spécifique");
    }
    @Override
    protected String getContext() {
        return "hornettemplate";
```

```
    }
}
```

Des actions spécifiques peuvent être réalisées dans les méthodes **initContextActions** et **destroyContextActions** afin de s'adapter aux applications.

Une méhode complémentaire **getContext**, permet de paramétrer le nom de votre application. L'utilisation d'une constante est préférable à une chaine comme déclarée dans l'exemple.

3.10.9.2.3 ApplicationAuthenticationFailureBadCredentialListener

Voici la classe de base à ajouter dans le package de listener pour la journalisation des erreur d'authentification :

```
public final class ApplicationAuthenticationFailureBadCredentialListener extends
            HornetAuthenticationFailureBadCredentialListener {
    @Override
    protected void execElementSpecifiqueApplication(final String user) {
        // Pas d'action spécifique
        Logger.getLogger(this.getClass()).info("Pas d'action spécifique");
    }
}
```

Des actions spécifiques peuvent être réalisées dans la méthode **execElementSpecifiqueApplication** afin de s'adapter aux applications.

3.10.9.2.4 ApplicationAuthenticationSuccessListener

Voici la classe de base à ajouter dans le package de listener pour la journalisation des éléments en cas d'authentification réussie :

```
public class ApplicationAuthenticationSuccessListener extends
            HornetAuthenticationSuccessListener<Utilisateur, UtilisateurService> {
    @Override
    protected void initElementSpecifiqueApplication(final Utilisateur user) {
        // Pas d'élément spécifique
        Logger.getLogger(this.getClass()).info("Pas d'action spécifique");
    }
    @Override
    protected String getSessionsetAttribut() {
        // utilisation de l'attribut par défaut.
        Logger.getLogger(this.getClass()).info("Action par défaut");
        return HornetAuthenticationSuccessListener.ATT_USER;
    }
    @Override
    protected void initElementSpecifiqueUtilisateur(final Utilisateur user) {
        // Pas d'élément spécifique
        Logger.getLogger(this.getClass()).info("Pas d'action spécifique");
    }
}
```

Ce listener nécéssite l'existance d'un service de gestion des utilisateurs (ici UtilisateurService) qui étant de **hornet.framework.web.service.UtilisateurWebService**.

Ce service contient une méthode **lireUtilisateur** à partir d'un identifiant qui permet la récupération d'un objet représentant l'utilisateur (ici Utilisateur).

A partir de ces éléments, la classe mère :

1. Charge l'utilisateur
2. Réalise les actions spécique sur l'utilisateur (**initElementSpecifiqueUtilisateur**)
3. Dépose l'oblet représentant l'utilisateur dans la session http dans l'attribut spécifié (**getSessionsetAttribut**)
4. Réalise les actions spéciques sur l'application (**initElementSpecifiqueApplication**)

3.10.9.3 Initialisation des listeners

3.10.9.3.1 Modification du web.xml

Ajouter dans le web.xml les éléments suivant, en fonciton des listeners implémenté au dessus :

```
<listener>
  <listener-class>
        ${package.listener}.ApplicationContextListener
  </listener-class>
</listener>
<listener>
  <listener-class>
        ${package.listener}.ApplicationSessionListener
  </listener-class>
</listener>
```

Remplacer ${package.listener} par le package dans lequel les classes de listener sont implémentées.

Il n'est pas possible d'ajouter les listener **ApplicationAuthenticationFailureBadCredentialListener et ApplicationAuthenticationSuccessListener** de la même manière car il s'agit de listener de type « Spring » (ils héritent de la classe org.springframework.context.ApplicationListener). Leur initialisation est décrite au § suivant.

3.10.9.3.2 Modification du spring-appContext-security.xml

Ajouter les lignes suivantes au début du fichier spring-appContext-security.xml :

```
<!--  listener de logging en succes et en échec -->
<context:component-scan base-package="hornet.framework.web.listener" />
<bean id="authenticationSuccessListener"
class="${package.listener}.ApplicationAuthenticationSuccessListener"/>
<bean id="authenticationFailureBadCredential"
class="${package.listener}.ApplicationAuthenticationFailureBadCredentialListener"/>
```

Remplacer ${package.listener} par le package dans lequel les classes de listener sont implémentées.

3.10.10 Spring AOP / Metrologie

3.10.10.1 Configuration Spring AOP

La configuration spring aop couplé au module metrologiefilter de HornetServer se fait via le fichier de configuration spring « spring-appContext-aopMetrologie.xml » suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"
	xmlns:tx="http://www.springframework.org/schema/tx" xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
	http://www.springframework.org/schema/aop
		http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
		http://www.springframework.org/schema/context
		http://www.springframework.org/schema/context/spring-context.xsd">

	<!-- ======================== METROLOGIE ================================= -->
	  <context:annotation-config />
	<context:component-scan base-package="fr.gouv.diplomatie"
		use-default-filters="false">
		<context:exclude-filter type="aspectj"
			expression="fr.gouv.diplomatie..*Action" />
	</context:component-scan>

	<!-- metrologieAspect -->
	<bean id="metrologieAction" class="hornet.framework.metrologie.aspect.MetrologieAspect">
		<constructor-arg index="0">
			<null />
		</constructor-arg>
		<constructor-arg index="1" value="CONTROLLEUR" />
	</bean>
	<bean id="metrologieService" class="hornet.framework.metrologie.aspect.MetrologieAspect">
		<constructor-arg index="0" value="CONTROLLEUR" />
		<constructor-arg index="1" value="SERVICE" />
	</bean>
	<bean id="metrologieDAO" class="hornet.framework.metrologie.aspect.MetrologieAspect">
		<constructor-arg index="0" value="SERVICE" />
		<constructor-arg index="1" value="DAO" />
	</bean>

	<aop:config proxy-target-class="true">
		<aop:aspect id="metrologieAspectAction" ref="metrologieAction">
			<aop:pointcut id="metrologieActionPointcut"
				expression="execution(public * fr.gouv.diplomatie..*Action.*(..))" />
			<aop:around pointcut-ref="metrologieActionPointcut"
				method="loggerTraceMetrologie" />
		</aop:aspect>
		<aop:aspect id="metrologieAspectService" ref="metrologieService">
			<aop:pointcut id="metrologieServicePointcut"
				expression="execution(public * fr.gouv.diplomatie..*ServiceImpl.*(..))" />
			<aop:around pointcut-ref="metrologieServicePointcut"
				method="loggerTraceMetrologie" />
		</aop:aspect>
		<aop:aspect id="metrologieAspectDAO" ref="metrologieDAO">
			<aop:pointcut id="metrologieDaoPointcut"
				expression="execution(public * fr.gouv.diplomatie..*DAOImpl.*(..))" />
			<aop:around pointcut-ref="metrologieDaoPointcut" method="loggerTraceMetrologie" />
		</aop:aspect>
	</aop:config>

</beans>
```

3.10.10.2 Modification du fichier spring-appContext-web.xml

Le fichier « **spring-appContext-web.xml** » de l'application doit être modifié pour importer le fichier de configuration spring « spring-appContext-aopMetrologie.xml ».

```
<import resource="classpath:/spring-appContext-aopMetrologie.xml"/>
```

3.10.11 Gestion de cache : EhCache / Spring

3.10.11.1 Présentation

EhCache est un framework libre pour résoudre les problèmes de cache. EhCache est principalement composé de 3 éléments :

- CacheManager : permet la gestion et le cycle de vie du cache
- Caches: est la classe centrale du framework. Elle gère les « elements » et est gérée par le « CacheManager ». On présente des configurations sur cache dans le fichier « **ehcache.xml** ».
- Elements: Un « element » est constitué par une clé et une valeur. La valeur peut-être n'importe quel objet qui implémente l'interface Serializable.

3.10.11.2 Configuration EhCache

La configuration de « EhCache » se trouve dans le fichier de configuration « **ehcache.xml** » dont voici un aperçu :

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="../src/config/ehcache.xsd" updateCheck="false"
        monitoring="autodetect" dynamicConfig="true" name="HORNET_CACHE_MANAGER_DEV">
        <diskStore path="java.io.tmpdir/ehcache" />
        <defaultCache maxEntriesLocalHeap="0" eternal="false" statistics="true"
                timeToIdleSeconds="1200" timeToLiveSeconds="1200">
                <persistence strategy="localTempSwap" />
        </defaultCache>
        <cache name="secteurCache" maxEntriesLocalHeap="1" maxEntriesLocalDisk="0"
                eternal="true" statistics="true">
                <persistence strategy="localTempSwap" />
        </cache>

</ehcache>
```

La plupart des noms des attributs ou balises ont un nom explicite. Néanmoins on peut faire le focus sur l'élément suivant :

- diskStore : cette balise contient un attribut « path » dans lequel on indique le chemin du répertoire ou seront stockés les éléments du cache.

3.10.11.3 Configuration Spring / EhCache

La configuration Spring de EhCache doit être ajoutée dans un fichier de contexte Spring ; par exemple dans le fichier « **spring-appContext-common-dao.xml** » on peut ajouter les lignes suivantes :

```
        <!-- #### DEBUT Gestion du cache  #### -->
        <ehcache:annotation-driven cache-manager="ehCacheManager" />

        <bean id="ehCacheManager"
                class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
                <property name="configLocation" value="classpath:ehcache.xml" />
                <property name="shared" value="true" />

        </bean>

        <bean id="mbeanServer"
                class="org.springframework.jmx.support.MBeanServerFactoryBean">
                <property name="locateExistingServerIfPossible" value="true" />
        </bean>

        <!-- JMX for ehcache -->
        <bean id="managementService"
                class="net.sf.ehcache.management.ManagementService"
                init-method="init"
                destroy-method="dispose">

                <constructor-arg ref="ehCacheManager"/>
                <constructor-arg ref="mbeanServer"/>
                <constructor-arg index="2" value="true"/>
```

```
                <constructor-arg index="3" value="true"/>
                <constructor-arg index="4" value="true"/>
                <constructor-arg index="5" value="true"/>
                </bean>
        <!-- #### FIN Gestion du cache  #### -->
```

3.10.11.4 Implémentation Spring / EhCache

En utilisant les annotations Spring, l'implémentation du cache sur un service donné se fait tout simplement via l'annotation **@Cacheable** en précisant le nom du cache définit dans la section « Configuration EhCache » en l'occurence « **secteurCache »** :

```
    /**
     * retrieveDataList
     *
     * @return List < Secteur >
     */
    @SuppressWarnings("unchecked")
    @Cacheable(cacheName = "secteurCache")
    public List<Secteur> retrieveDataList() {

        return this.getSqlMapClientTemplate().queryForList("secteur.select");

    }
```

3.10.11.5 Invalidation du cache

Pour invalider le cache, il suffit d'ajouter l'annotation « **@TriggersRemove** » au niveau du service en indiquant le nom du cache en l'ocurrence « **secteurCache** » dans le cadre de notre exemple. On considère que l'ajout ou la modification d'un secteur doit permettre d'invalider le cache pour que ce dernier soit renouvelé lors du prochain appel du service de récupération de la liste des secteurs.

```
/**
 * @param secteur
 *            Secteur
 * @return Secteur secteur
 */
@TriggersRemove(cacheName = "secteurCache", removeAll = true)
public Secteur ajouterSecteur(Secteur secteur) {

    secteur.setId((Long) this.getSqlMapClientTemplate().insert("secteur.insert", secteur));
    return secteur;
}

/**
 * @param secteur
 *            Secteur
 * @return Secteur secteur
 */
@TriggersRemove(cacheName = "secteurCache", removeAll = true)
public Secteur modifierSecteur(Secteur secteur) {

    this.getSqlMapClientTemplate().update("secteur.update", secteur);
    return secteur;

}
```

3.10.11.6 Monitoring JMX du cache

Il est possible de realiser une supervision du cache via l'utilisation de JMX sur le serveur Tomcat. Voici la procedure à suivre pour permettre ce monitoring.

3.10.11.6.1 Activation du monitoring à distance

Bien qu'il soit possible par défaut sur la plupart des systèmes (à l'exception d'Ubuntu ou CentOS) de faire du monitoring local (en développement), il est plus pratique d'activer le monitoring à distance pour permettre une vraie supervision des serveurs (en production par exemple). Pour cela, on doit démarrer le serveur Tomcat en ajoutant les paramètres suivants :

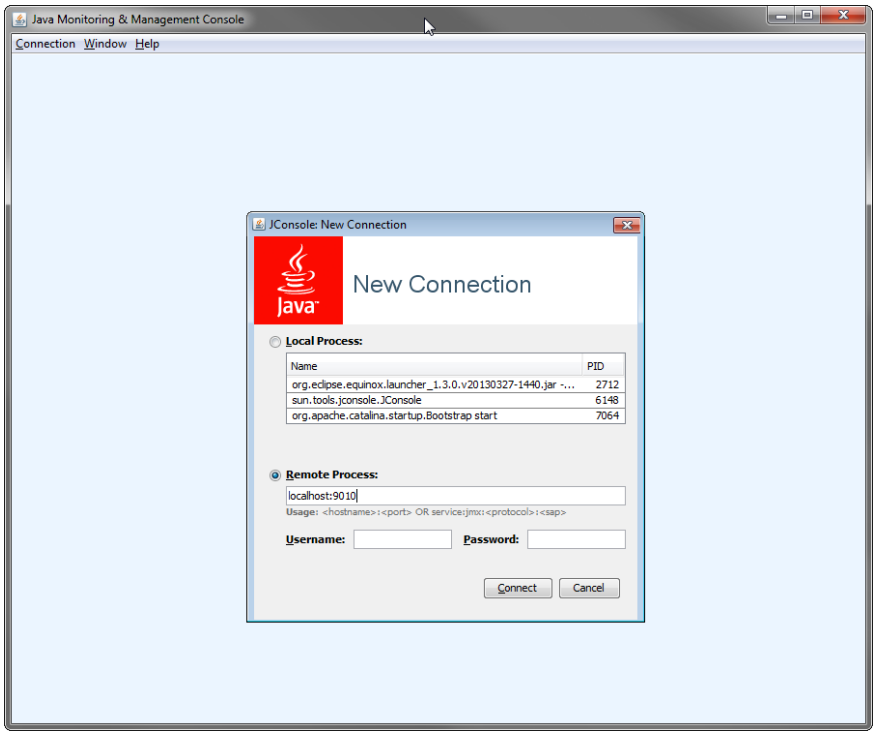
```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=9010
-Dcom.sun.management.jmxremote.local.only=false
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

A noter que la configuration du port peut être modifiée à loisir pour utiliser tout autre port à condition qu'il soit libre. On pourra sur une machine de type linux modifier le fichier « **setenv.sh** » de Tomcat et y ajouter les lignes suivantes :

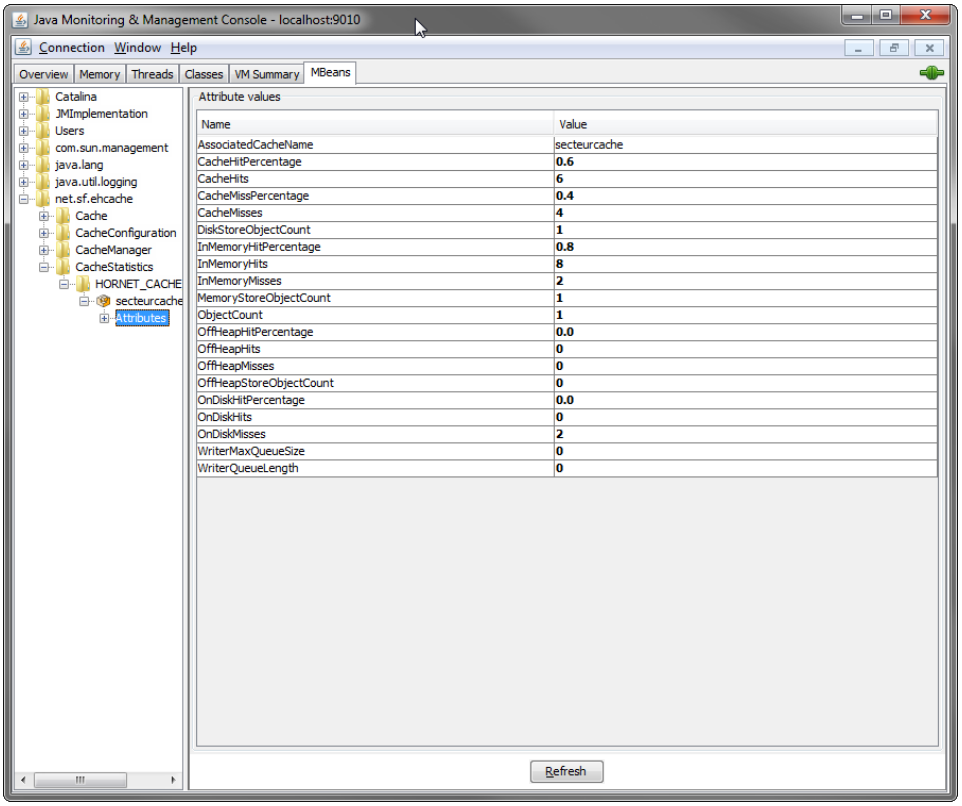
```
export JAVA_OPTS="$JAVA_OPTS \
-Dcom.sun.management.jmxremote \
-Dcom.sun.management.jmxremote.port=9010 \
-Dcom.sun.management.jmxremote.local.only=false \
-Dcom.sun.management.jmxremote.authenticate=false \
-Dcom.sun.management.jmxremote.ssl=false"
```

3.10.11.6.2 Monitoring via Jconsole

L'outil JConsole inclus dans le JDK peut permettre la supervision du cache, le lancer. On obtient l'écran suivant :



Choisir l'option Remote Process puis entrer le nom du serveur suivi du port en les séparant d'un « : ». Pour se connecter sur un serveur Tomcat en local, on entrera donc « **localhost:9010** ». Puis choisir le bouton « Connect »



Se positionner dans l'onglet « MBeans » et dérouler le nœud « net.sf.ehcache ». Différentes informations sont alors disponibles dont notamment pour les plus intéressantes les statistiques de CacheHits au niveau du nœud « CacheStatistics ». Dans la capture précédente, on peut voir que le cache « secteurcache » à pu être utilisé 6 fois sur 10. C'est-à-dire qu'il a permis une optimisation des performances dans 60% des cas.

3.11 Planification des traitements batchs

Les traitements batch à lancer de manière périodique doivent utiliser le Framework java d'ordonnancement de tâches « Quartz » (en remplacement des déclenchements par Cron/tab).

3.11.1 Présentation de Quartz

Il existe trois objets principaux pour Quartz. Le scheduler qui correspond à l'ordonnanceur des tâches, les jobs qui correspondent aux tâches à exécuter et les triggers qui déclenchent l'exécution d'une tâche.

3.11.1.1 Scheduler

Le scheduler correspond donc à l'ordonnanceur de tâches. Il est indispensable à la planification, c'est sur lui que sont fixés les jobs ainsi que les triggers. Pour nos exemples, nous utiliserons le scheduler par défaut qui ne gère pas la persistance des tâches.

3.11.1.2 Job

Il faut maintenant créer une tâche qui sera planifiée ultérieurement. Cette classe doit implémenter l'interface Job, qui lui confère la méthode « execute » à écrire et qui va lancer le job à proprement dit.

3.11.1.3 JobDetail

Pour instancier le Job, nous avons besoin d'une nouvelle classe appelée JobDetail. Cette classe est indispensable à l'activation du Job. Dans son principe le JobDetail est une sorte de « Factory » pour les objets Job à instancier et à exécuter.

3.11.1.4 Trigger

Il s'agit de créer des déclencheurs des tâches implémentées préalablement. Il est à noter qu'un *job* peut être déclenché par plusieurs *triggers*, mais de façon non réciproque, un *trigger* ne peut pas déclencher plusieurs *Jobs*.

Il faut savoir qu'il existe deux types de triggers :

- *SimpleTrigger* : ce type de *trigger* se base sur des intervalles réguliers et sur un nombre de répétitions précis
- *CronTrigger* : celui-ci offre plus de liberté sur la planification, puisqu'il se base sur les jours du calendrier

3.11.2 Configuration de Quartz avec Spring

La déclaration des jobs et des triggers permettant le déclenchement des jobs Quartz se fait par l'intermédiaire d'un fichier de configuration spring « spring-quartz.xml » dédié,

La déclaration d'un job se réalise par l'intermédiaire d'un objet de type JobDetail. Les objets JobDetail contiennent toutes les informations nécessaires pour lancer un job.

Spring permet la création d'un JobDetailBean.

3.11.2.1 Configuration du Job

Trois propriétés sont à renseigner lors de la déclaration du bean correspondant au Job dans le fichier spring-quartz.xml :

- targetObject : La classe contenant le service ou méthode que l'on souhaiterait exécuter
- targetMethod : Le nom de la méthode que l'on souhaite exécuter

Dans le fichier spring-quartz.xml :

```
<!-- Job -->
  <bean name="sendEmailToAdminJobDetail"
         class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
         <property name="targetObject" ref="sendEmailToAdminJob" />
         <property name="targetMethod">
                <value>sendMail</value>
         </property>
  </bean>

  <!-- My Email Job -->
  <bean name="sendEmailToAdminJob"
         class="fr.gouv.diplomatie.monprojet.quartz.SendEmailToAdminJob">
         <property name="mailContactService" ref="mailContactService" />
  </bean>
```

Dans le projet :

```
package fr.gouv.diplomatie.monprojet.quartz;

import org.springframework.beans.factory.annotation.Autowired;

import fr.gouv.diplomatie.monprojet.business.service.MailContactService;

/**
 * The job to send an email
 *
 * @author Hornet
 * @since 1.0 - 9 sept. 2014
 */
public class SendEmailToAdminJob {

    /**
     * Email Service
     */
    @Autowired
    private MailContactService mailContactService;

    /**
     * Send Mail
     */
    protected void sendMail() {

        this.mailContactService.envoyerMail("Quartz", "Quartz", "Quartz", "Hello admin");
    }

    /**
     * @return mailContactService
     */
    public MailContactService getMailContactService() {

        return this.mailContactService;
    }

    /**
     * @param mailContactService
     *            mailContactService
     */
    public void setMailContactService(MailContactService mailContactService) {

        this.mailContactService = mailContactService;
    }

}
```

3.11.2.2 Configuration du Trigger

Deux propriétés sont à renseigner lors de la déclaration du bean correspondant au Trigger qui déclenchera le Job, dans le fichier spring-appContext-quartz.xml:

- JobDetail : La référence vers le JobDetailBean décrivant le Job
- cronExpression : L'expression CRON décrivant la séquence de déclenchement du trigger

Dans le fichier spring-appContext-quartz.xml :

```
<!-- Trigger -->
  <bean id="cronTrigger"
        class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
        <property name="jobDetail" ref="sendEmailToAdminJobDetail" />
        <!-- Exécution tous les jours à 15 h -->
        <property name="cronExpression" value="0 0 15 * * ?" />
   </bean>
```

3.11.2.3 Configuration du Scheduler

La dernière étape est de déclarer dans le spring-appContext-quartz.xml une fabrique d'ordonnanceur, qui permettra à Spring de créer et gérer l'ordonnanceur de l'application, en lui indiquant les triggers déclarés précédemment.

L'initialisation de Quartz se fait par Spring. Spring ne fait pas automatiquement le lien entre le fichier "quartz.properties" et la configuration de Quartz. Il faut spécifiquement indiquer quel est le fichier properties à utiliser pour configurer Quartz.Exemple d'initialisation de Quartz avec Spring :

```
<!-- Scheduler -->
  <bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean"
        lazy-init="false">

        <property name="autoStartup" value="false" />

        <property name="triggers">
               <list>
                      <ref bean="cronTrigger" />
               </list>
        </property>

        <property name="quartzProperties">
               <bean
                      class="org.springframework.beans.factory.config.PropertiesFactoryBean">
                      <property name="location" value="classpath:quartz.properties" />
               </bean>
        </property>

  </bean>
```

3.11.3 Bonnes pratiques

3.11.3.1 Fichier de configuration quartz.properties

Une bonne pratique consiste à spécifier le fichier `quartz.properties` afin de bénéficier des configurations quartz pour les logs et par la suite pour l'interconnexion avec automator.

Les paramètres tels que les logs de succès ou d'erreur de job et le nombre de thread max seront alors pris en compte.

Exemple de fichier "quartz.properties" :

```
#============================================================================
# Configuration Main Scheduler Properties
#============================================================================
org.quartz.scheduler.instanceName = QuartzScheduler
org.quartz.scheduler.instanceId = AUTO
#============================================================================
# Configuration ThreadPool
#============================================================================
org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
org.quartz.threadPool.threadCount = 2
org.quartz.threadPool.threadPriority = 5
#============================================================================
# Configuration JobStore
#============================================================================
org.quartz.jobStore.misfireThreshold = 60000
org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
#============================================================================
# Configuration Plugins
#============================================================================
# configuration des logs
org.quartz.plugin.jobHistory.class = org.quartz.plugins.history.LoggingJobHistoryPlugin
org.quartz.plugin.jobHistory.jobSuccessMessage = Traitement {1}.{0} du {2, date, dd/MM/yyyy HH:mm:ss}
code retour=OK
org.quartz.plugin.jobHistory.jobFailedMessage = Traitement {1}.{0} du {2, date, dd/MM/yyyy HH:mm:ss}
code_retour=ERREUR
```

3.11.3.2 Utilisation de types de donnés primitifs

Pour transférer des informations à un Job via « JobDataMap », il faut utiliser des types de données primitifs (String par exemple).

3.11.3.3 Manipulation des Trigger

Pour manipuler les « Trigger » il faut utiliser la classe utilitaire « TriggerUtils » offrant notamment des « helpers » pour analyser les « Trigger ».

3.11.3.4 JDBC Jobstore

Il ne faut jamais écrire **directement** dans les tables de Quartz en base.

3.11.3.5 Clustered Scheduler

Dans une configuration non clustérisé de Quartz, il ne faut pas faire pointer un Scheduler non clustérisé vers une base de donnée en même temps qu'un autre Schudeler portant le même nom. Il y'a un risque de corruption en base.

3.11.3.6 Datasource connection size

Il est recommandé d'avoir un nombre maximum de connections à la Datasource supérieur au nombre de thread dédié au pool de connexion (de préférence plus 3).

3.11.3.7 Gestion de la famine

Lorsqu'un Job attend indéfiniment la satisfaction d'une condition et que tous les threads sont utilisés, il est préférable de libérer ce Job afin que son thread puisse servir à un autre Job qui est en attente.

3.11.3.8 Throwing exceptions

Il est préférable que le Job « catch » et gère les Exceptions pouvant intervenir au lieu de faire un « Throw ». En effet, si le Job Throw une « Exception », Quartz va immédiatement le ré exécuter.

3.11.3.9 Code in Listeners

Le code implémenté dans les « Listeners » TriggerListener, JobListener ou SchedulerListener doit être concis et efficient.

Toute méthode d'un listener soit avoir un try-catch pour traiter toute exception pouvant être générée.

3.11.4 Modification de spring-appContext-web.xml

Le fichier « spring-appContext-web.xml » de l'application doit être modifié pour importerle fichier de configuration spring « spring-appContext-quartz.xml».

```
<import resource="classpath:/spring-appContext-quartz.xml"/>
```

3.12 Appel de WebService en SSL

Les applications peuvent faire appel à des web services. Dans le cas où la communication doit être crypté et se faire en SSL, on utilisera au choix les classes java « **SSLSocketSpringFactory** » ou « **SSLSocketSpringFactoryWithTrustStore** » fournies dans le module « **hornetserver-webservicehelper** ».

Pour se connecter sur un serveur de manière sécurisé, on peut se reposer sur le « Keystore » et le « Truststore » :

- Le « Keystore » est un magasin de clé de certificat qui contient la clé privé et le certificat de la JVM du serveur local
- Le « Truststore » est un magasin de clé de certificat qui contient les certificats des serveurs distants, signés par une autorité de certification.

Selon le besoin, on peut générer une « Socket » de communication reposant soit uniquement sur le « Keystore » ou soit à la fois sur le « Keystore » et le « Truststore ».

3.12.1 Exemple d'appel d'un WS

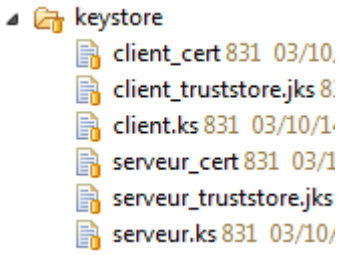
L'application tutoriel fourni un exemple d'appel vers un web service (nomme HelloService) en SSL et utilisant à la fois sur le « Keystore » et le « Truststore ».

Le WSDL du service appelé est disponible sous :

```
applitutoriel/src/config/fr/gouv/diplomatie/bouchonwebservice/wsdl/HelloService.wsdl
```

Les certificats client et serveur sont disponibles sous :

```
applitutoriel/keystore
```



De plus, le fichier webservices.properties doit référencer les chemins vers les fichiers de keystore et de truststore, ainsi que certains paramètres comme les mots de passe ou les algorithmes utilisés :

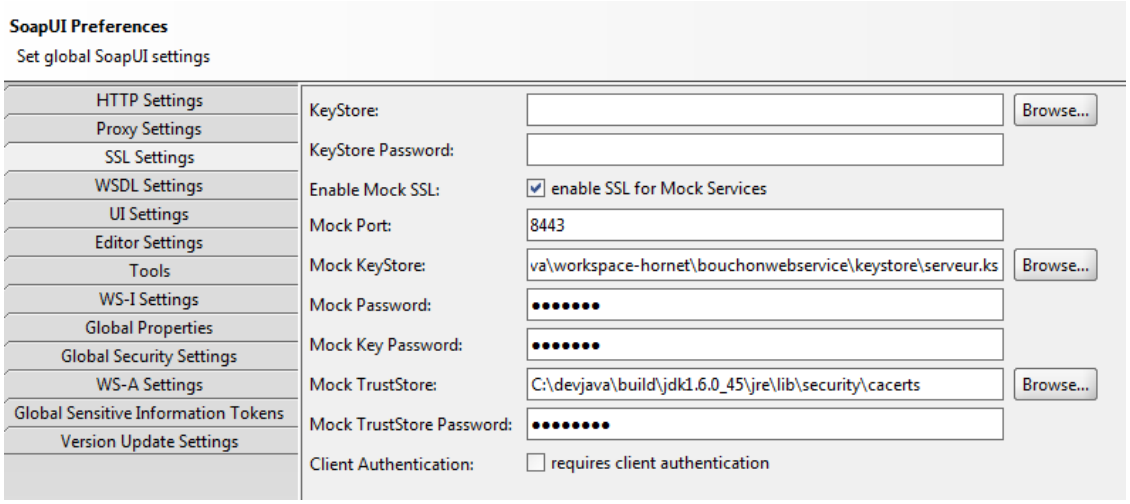
```
# propriétés pour le KeyStore
applitutoriel.keyStore=/etc/ssl/certs/client.ks
applitutoriel.keyStorePassword=client
applitutoriel.aliasCertificat=client
applitutoriel.algoKeyStore=SunX509
applitutoriel.typeKeyStore=JKS

# propriétés pour le TrustStore
applitutoriel.trustStore=/etc/ssl/certs/client truststore.jks
applitutoriel.trustStorePassword=client
applitutoriel.algoTrustStore=SunX509
applitutoriel.typeTrustStore=JKS

# endpoint du webservice
applitutoriel.HelloService.endpoint=https://localhost:8443/mockHelloServiceSoapBinding
```

Il est possible de mocker le WS HelloService via un outil comme SOAPUI qui permet de simuler des webservices. L'outil est disponbile en téléchargement à l'adresse suivante :

http://sourceforge.net/projects/soapui/files/soapui/

Après installation du logiciel, il faut configurer les préférences de SOAPUI afin de configurer l'authentification par SSL :

- Aller dans File > Preferences > SSL Settings
- Activer la case enable SSL for Mock Services
- Renseigner Mock Port n° 8443;
- Renseigner Mock KeyStore avec le chemin du fichier serveur ks situé dans le dossier keystore ; serveur.ks
- Renseigner Mock Password avec la valeur "serveur" ;
- Renseigner Mock Key Password avec la valeur "serveur" ;
- Renseigner le champ Mock Trust Store avec le fichier jssecacerts présent dans le dossier jre/lib/security de la JVM ;
- Renseigner le champ Mock Trust Store password avec "changeit"
- Cliquer sur Ok
- Aller dans File et cliquer sur Save Preferences
- Redémarrer SOAPUI

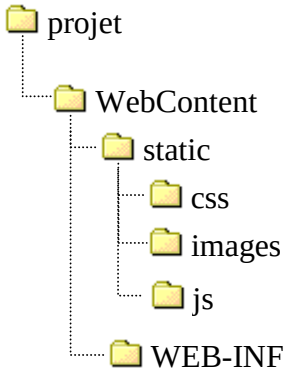


4 Nomenclature

4.1 Client riche

Elément	Nomenclature	Exemple	répertoire
Fichier Javascript	/js/<bigramme du cas d'utilisation>/<nom du cas d'utilisation>.js	/js/lp/recherche.js	WebContent/static

Exemple :



4.2 Serveur

La structuration des packages et répertoires se fait par couche applicative d'abord et (éventuellement) par cas d'utilisation ensuite.

Elément	Nomenclature	Exemple	Répertoire
Action	fr.gouv.diplomatie.<nom_projet>.web.action.<cas d'utilisation>.nomAction	fr.gouv.diplomatie.<nom_projet>.web.action.gestiondossier.ListerAction	src/java
Service (interface)	fr.gouv.diplomatie.<nom_projet>.business.service.<cas d'utilisation>.nomService	fr.gouv.diplomatie.<nom_projet>.business.service.DossierService	src/java
Service (implémentation)	fr.gouv.diplomatie.<nom_projet>.business.service.<cas d'utilisation>.nomServiceImpl	fr.gouv.diplomatie.<nom_projet>.business.service.DossierServiceImpl	src/java
BO	fr.gouv.diplomatie.<nom_projet>.business.bo.<cas d'utilisation>.nomBO	fr.gouv.diplomatie.<nom_projet>.business.bo.DossierBO	src/java
DAO (interface)	fr.gouv.diplomatie.<nom_projet>.integration.dao.nomTableDAO		src/java
DAO (implémentation)	fr.gouv.diplomatie.<nom_projet>.integration.dao.nomTableDAOImpl		src/java
DAO (mapping SQL)	fr.gouv.diplomatie.<nom_projet>.integration.dao.map.nomTable_sqlMap.xml		src/java

Classe de test d'action	fr.gouv.diplomatie.<nom_projet>.web.action.<cas d'utilisation>.nomActionTest		tst/java
Classe de test de service	fr.gouv.diplomatie.<nom_projet>.business.service.<cas d'utilisation>.nomServiceTest		tst/java
Page JSP	WEB-INF/tiles-jsp/<cas d'utilisation>/nomAction.jsp		WebContent
Flux XML (JSP)	WEB-INF/xml-jsp/<cas d'utilisation>/nomAction.jsp		WebContent
Template (JSP)	WEB-INF/templates/nomExport.jsp (.xsl)		WebContent
Log4J	log4j.properties		src/config
Struts Config	struts.xml		src/config
Spring général + business	spring-appContext-common-service.xml		src/config
Spring dao	spring-appContext-common-dao.xml		src/config
MyBatis SqlMapConfig	mybatis.xml		src/config
Tiles	tiles.xml		src/config

5 Configuration Applicative

5.1 Serveur d'application

Liste des paramètres et de leurs valeurs à mettre à jour dans les fichiers de configurations de l'application Web situés sous « /envconfig »[6] et « /livraison/webapp/config »[7] dans le projet :

hornet.properties :

Paramètre	Description	Valeur
fwkRoot	Chemin vers le framework hornetclient. Note : pas de « / » en fin d'url	Ex : http://serveur_mutualise/hornetclient/*<version_hornetclient>*/fwk
yui3Root	Chemin vers le framework Yahoo UI 3. Note : pas de « / » en fin d'url	Ex : http://serveur_mutualise/yui/yui/3.17.2
themeName	Référence du thème Hornet à utiliser	Ex : Diplonet
themeVersion	Version du thème Hornet à utiliser	*<version_hornetclient>*
combineType	Type de mode Combine Valeurs possibles : [basic, normal, full, none] Valeur par défaut : normal	basic
debugHornetClient	Active ou non le debug YUI et HornetClient Valeurs possible : [true, false] Valeur par défaut : false Attention, ne pas mettre à true pour une version de production !	false

Tableau 2 : Configuration hornet - Serveur d'application

log4j.properties :

Paramètre	Description	Valeur
repertoire	Répertoire de log	Ex : '/var/log/tomcat6' (debian) Ou 'D:/logs' (windows)
nomFichier	Nom du fichier où doivent se trouver les fichiers de log.	Ex : [PROJET].log
log4j.rootLogger	Niveau des logs et appender.	**INFO, FILE**

Tableau 3 : Configuration des logs - Serveur d'application

jdbc.properties :

Paramètre	Description	Valeur
jdbc.driver	Driver SQL	net.sourceforge.jtds.jdbc.Driver
jdbc.url	Url de connexion à la BDD	jdbc:jtds:sqlserver://10.104.19.27:50000/[PROJET]_DVLT_01; packetSize=16384;applicationName=[PROJET]
jdbc.username	Utilisateur pour la connexion	Ex : [projet]_dvlt_01_dbo

[6] Jeu de fichier de configuration pour le lancement de l'application en mode développement (sous Eclipse / WTP).

[7] Jeu de fichier de configuration pour la constitution d'une archive WAR dans le but d'un déploiement sur un environnement de recette, qualification ou production.

jdbc.password	Mot de passe de l'utilisateur	Ex : [projet]_dvlt_01_dbo

Tableau 4 : Configuration des logs - Serveur d'application

mail.properties :

Paramètre	Description	Valeur
mail.smtp.host	Serveur de mail	Ex : smtp.serveur.fr
mail.smtp.username	Nom du compte utilisateur	
mail.smtp.password	Mot de passe du compte utilisateur	
mail.debug	Activer ou non les traces de débogage pour l'envoi des mails	true

Tableau 5 : Configuration des mails - Serveur d'application

cas.properties :

Paramètre	Description	Valeur
cas.loginUrl	Url de connexion à cas	https://cas-server:8443/cas-server-webapp/login
cas.logoutUr	Url de déconnexion à cas	https://cas-server:8443/cas-server-webapp/logout
cas.securityCheckUrl	Url de vérification	https://localhost/[PROJET]/j_spring_cas_security_check
cas.ticketValidationUrl	Url de validation	https://cas-server:8443/cas-server-webapp/

Tableau 6 : Configuration cas - Serveur d'application

freemarker.properties :

Paramètre	Description	Valeur
template_update_delay	Temps de mise à jour du template	60000

Tableau 7 : Configuration cas - Serveur d'application

antivirus.properties :

Paramètre	Description	Valeur
antivirus.actif	Booléen indiquant si l'analyse doit être effectuée ou non.	**true**
antivirus.adresseServeur	Adresse du serveur ClamAV	Ex : 10.104.19.108
antivirus.portServeur	Port utilisé par le serveur ClamAV	Ex : 53310
antivirus.timeoutScan	Timeout de scan d'un fichier (en ms)	**1000**
antivirus.timeoutConnection	Timeout de connexion au serveur ClamAV (en ms)	**1000**

Tableau 8 : Configuration de l'antivirus - Serveur d'application

[NOM_APPLICATION].properties : Ce fichier de configuration correspond au besoin spécifique d'une application.

Paramètre	Description	Valeur
[NOM_APPLICATION].tailleMax	Taille maximum d'un fichier uploadé (en octets)	**209715200**
[NOM_APPLICATION].dureeVieContenu	Durée de stockage d'un fichier (en jours)	**7**
[NOM_APPLICATION].dureeVieInformations	Durée de stockage des informations d'un fichier (en jours)	**365**
[NOM_APPLICATION].extensionsInterdites	Extensions de fichier interdites	**exe,bat,dll,vbs,bas,vbx**

[NOM_APPLICATION].typesMimeInterdits	Type mime interdits	**application/octetstream,application/x-ms-dos-executable**

Tableau 9 : Configuration de l'application - Serveur d'application

Le contexte de l'application Web est décrit dans un fichier XML portant le nom de l'application déployée. Il défini :

- Le paramétrage de la connexion à la base de données.
- Le chemin absolu de stockage des fichiers properties sur le serveur.

Liste des paramètres et de leurs valeurs à mettre à jour dans le fichier de description du contexte de l'application Web sous /livraison/webapp/context/ dans le projet.

Ces éléments de configuration sont des paramètres d'infrastructures à charge de l'exploitation.

Paramètre	Description / Fichier	Valeur
Context		
Environnement		
conf/[NOM_APPLICATION]prop	Chemin de stockage des fichiers de configuration de l'application Web	/home/qualif/colimatic/config

Tableau 10 : Configuration – Contexte applicatif Internet - Serveur d'application

Exemple de fichier XML :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Context path="/monapplication" useHttpOnly="true" disableURLRewriting="true">
  <!-- Chemin vers le répertoire des fichiers de propriété -->
  <Environment name="conf/monapplicationprop" value="/home/monapplication/config" type="java.lang.String"
/>

</Context>
```

6 Outils de développement

- Eclipse (avec template de dev Java)
- Plugins : Checkstyle, Findbugs, PMD, iBator (abator)
- Firefox + FireBug pour le code html + javascript

FIN DU DOCUMENT