Plan

Remerciements

Introduction

Contexte du projet (bésoins pour nous et pour IOCEAN)

Objectifs détaillés (la spec de départ)

Planning prévisionnel

Méthodologie et environnement technique

Méthode de développement en V

Architecture MVC

M : entités

C : Composants Seam

V : Pages JSF, Facelets

Bibliothèque RichFaces

Framework Seam

Injection de dépendance

Configuration générale

Système de gestion d’exception dans Seam

…

Framework Hibernate

Mapping Objet/relationnel

Stratégie de mapping choisie

A l’aide des fichiers hbm.xml

A l’aide des annotations JPA (Java Persistence API)

Frameworks JSF, Facelets

Déploiement de l’application sous Tomcat 6.0

Base de données  : Mysql

Outil de collaboration et de versionning : SVN

Travaux réalisés

Diagrammes des cas d’utilisation

Back end

Scénarios

Front end

Diagramme de classes

Gestion du contenu

Structure de données

Design pattern composite

Gestion des droits d’utilisateurs

Design pattern … pour adaptation facile

Droit de rédacteur

Droit de gestionnaire

Droit d’administrateur

Templates de pages web

Back end

Front end

Gestion des exceptions

Tests unitaires JUnit

Installation

Fichiers de configuration

Déploiement de la solution

Conclusion

Glossaire

Bibliographie

Annexes

Back end

Screnshots

Front end

Screnshots

## Le modèle de composant Seam

A l'origine le modèle de composants de Seam a été conçu pour permettre l'intégration directe entre JSF et les EJB3.

Quand on développe une application web à partir de JSF, on implémente généralement deux types de composants :

* une page JSF qui va constituer la vue et contenir le code HTML,
* une classe Java appelée backing bean JSF qui va constituer le modèle et fournir les méthodes pour implémenter les actions utilisateurs (soumission formulaire ...).

Les méthodes du backing bean JSF liées aux actions utilisateurs accèdent en général à un composant métier. Si celui-ci est déployé sous la forme d'un EJB 3 de type Session Bean, il est alors nécessaire d'implémenter une couche d'adaptation entre ces deux types de composants.



Figure 1: Intégration JSF/EJB 3 sans le framework Seam

Avec Seam, il n'y a plus de distinction entre le composant backing bean et le composant métier ; ils forment un seul et unique composant qu'on appelle composant Seam. Une page JSF peut donc invoquer directement un composant métier déployé sous la forme d'un Session Bean.



Figure 2: Intégration JSF/EJB3 avec le framework Seam

Un composant Seam est une classe java du type POJO ou EJB 3.0 qui contient les annotations @name pour identifier le composant et @Scope pour le contexte applicatif.

Exemple : déclaration d'un composant Seam :

@Name("sessionUtilisateur")

@Scope(ScopeType.*SESSION*)

**public** **class** SessionUtilisateur {

…

}

Chaque composant Seam est déclaré sous un nom logique (ici sessionUtilisateur) et l'appel de ce composant depuis une page JSF s'effectue de la façon suivante :

<h:inputText id="login" value="#{sessionUtilisateur.utilisateur.login}" required="true"/>

Avec ce principe, les développements JSF sont simplifiés, puisque il n'y a plus besoin de déclarer les backing beans dans le fichier de configuration faces-config.xml.

La spécification Web Beans JSR 299 (http://jcp.org/en/jsr/detail?id=299) est basée sur les concepts de ce modèle de composant et sera très certainement intégrée dans la **spécification JEE6.**

## Contextes Seam

On accède à un composant Seam depuis son contexte. Chaque type de composant est déclaré par défaut dans un contexte mais il est facile d'étendre le contexte d'un composant. Pour cela, Seam met à notre disposition différents contextes :

* **stateless** qui est le contexte par défaut des composants EJB Stateless, sert pour les composants sans état.
* **request** (idem J2EE correspond à la requête HTTP) ou **event** qui est le contexte par défaut des composants du type POJO ou JavaBean,
* **page** (idem J2EE), le composant est accessible sur la durée de vie d'une page JSF,
* **conversation** qui correspond à un enchainement de pages.
* **session** (idem J2EE), le composant est accessible depuis la session HTTP,
* **process**, contexte utilisé pour la gestion des processus (par exemple JBPM)
* **application** (idem J2EE) est commun à toute l’application web et dure jusqu’à l’arret du serveur d’application.

## Gestion du contenu :

Pour mieux répondre aux spécifications d’IOCEAN, nous avons utilisé le design pattern « Composite ». Cela nous a permit d’hiérarchiser tout le contenu de l’application.

## Methode de développement

## Inversion de contrôle et injection de dépendances :

Pour que les composants puissent interagir entre eux, Seam propose un mécanisme d'injection des dépendances bijectif :

* un composant peut être injecté automatiquement dans un autre composant (équivalent au mécanisme d'injection des dépendances du framework Spring),
* un composant peut également créer et projeter un autre composant dans un contexte

(outjection) qui deviendra ainsi accessible par tous les autres composants.

L'injection ou la projection des dépendances s'effectue encore une fois à partir d'annotations Java, respectivement @In et @Out.

Il est également possible de forcer la création ou non des composants au moment de l'injection.

Avec Seam, nous avons donc un mécanisme d'injection des dépendances complètement dynamique, ne nécessitant pas de configuration XML.

Les composants Seam vont fréquemment avoir besoin d’utiliser d’autres composants Seam.

L’idée de l’inversion de contrôle en anglais Inversion Of Control (IoC) est de ne pas gérer manuellement dans chaque composant la récupération des autres composants dont il a besoin, mais de le laisser faire au framework. Les composants vont ainsi pouvoir être développés indépendamment les uns des autres, en laissant le soin au framework de faire le lien entre eux à l’exécution.

Chaque composant va signaler ses dépendances, et le framework va les lui injecter.

On indique une dépendance devant être injectée en ajoutant une annotation @In à une variable d’instance d’un composant.

Le framework va au-delà de l’injection de dépendance, puisqu’il va aussi gérer l’opération inverse, via l’annotation @Out, pour (re-)publier après l’exécution d’une méthode une dépendance si sa référence a été modifiée par la méthode.

L’annotation @Out est utile seulement si, à la sortie de la méthode, la variable fait référence à un nouvel objet, soit parce qu’elle était à null avant, soit parce que l’objet précédent a été remplacé par un autre objet.

Il est toujours possible de modifier dans une méthode l’état (les données) du composant injecté sans qu’il soit nécessaire de le republier, dans ce cas l’annotation @Out est inutile.

## Gestion des conversations

Actuellement, pour gérer l'état conversationnel dans une application web, la seule solution est de stocker les informations au niveau de la session HTTP, ce qui entraîne souvent des problèmes de réinitialisation ou de purge des informations.

Avec Seam, on utilise ce qu'on appelle les conversations longues soit de manière implicite à partir de composants du type EJB3 stateful ou bien avec des JavaBeans en les déclarants dans le contexte de conversation.

Une conversation Seam va donc englober plusieurs requêtes et un utilisateur pourra utiliser plusieurs conversations sachant que chaque conversation est complètement indépendante.

Le démarrage d'une conversation longue s'effectue de manière explicite en positionnant par exemple une annotation Java @Begin au niveau d'une méthode métier.

## Templates de pages web :

### Back end :

Grace à l’utilisation des facelets dans notre application, nous avons pu faire du templating, qui est non seulement utile pour éviter la recopie de code mais aussi pour la réutilisation des composants. Les deux tempates se trouve dans le répertoire /template.

Chaque page de l’application (.xhtml) est donc une composition d’une page template-admin.xhtml, qui affiche l’entête de la page, le menu de navigation, le pied de page, dont on redéfinie le contenu. Dans le cas de la partie « Backend » chaque page est une composition de template-admin.xhtml qui est une composition de header-admin.xhtml en y ajoutant le menu.xhmtl et footer.xhtml. La page admin-xhtml utilise ce template et rédéfinit le contenu afin d’acquérir l’élément courant grâce à un composant seam qui retourne la page courante à insérer.