

Facilitez vos développements J2EE avec JBoss Seam

Octobre 07

Résumé

Le framework JBoss Seam est actuellement un des frameworks de la communauté Java qui génère le plus de « buzz ».

Pour certains, Seam est encore un framework parmi tant d'autres et, pour d'autres, c'est la solution technique ultime pour le développement d'applications web sur JEE5.

A première vue, JBoss Seam fait plutôt penser à une sorte de vitrine technologique du savoir faire de la communauté JBoss (Hibernate, JBPM, Drools, JBoss Cache, RichFaces, JBoss Ejb3 ...).

Mais quand on y regarde de plus près, le framework JBoss Seam permet réellement de faciliter et d'augmenter la productivité des développements web basés sur JEE5 grâce notamment à certains concepts novateurs (modèle de composant unifié) et aussi aux améliorations apportées à JEE5.

Seam nous réconcilie avec la technologie J2EE et surtout avec les développements JSF, il est donc tout à fait possible d'utiliser Seam en dehors du contexte JEE5.

Table des matières

1. SEAM : LE FRAMEWORK D'INTEGRATION JEE5	4
1.1 Pourquoi choisir Seam ?	4
1.2 Historique	4
1.3 Concepts techniques.....	4
1.3.1 Le modèle de composant Seam	4
1.3.2 Contextes Seam	6
1.3.3 Injection des dépendances	7
1.3.4 Gestion des conversations.....	7
1.3.5 Intégration avec JPA.....	8
1.3.6 Simplification de JSF	9
1.3.7 Support Web 2.0	9
1.4 Utilisation de Seam en dehors de JEE5.....	10
2. CONCLUSION	10

Liste des Figures

Figure 1: Intégration JSF/EJB 3 sans le framework Seam..... 5



1. Seam : le framework d'intégration JEE5

1.1 Pourquoi choisir Seam ?

Le développement d'applications web basées sur J2EE se résume souvent à assembler un ou plusieurs frameworks, parfois pour chaque couche applicative, ces frameworks étant basés ou non sur des standards et bien souvent en provenance de la communauté open source (Struts, JSF, Spring, Hibernate, iBATIS, Axis 2, Dozer ...).

Il n'est pas rare de se retrouver avec une dizaine de composants techniques à intégrer avec chacun leur propre modèle de développement, leur propre mécanisme de configuration et les éventuelles problématiques de compatibilité. Par exemple le choix des bibliothèques JSF demande actuellement pas mal de réflexion !

Avec le framework JBoss Seam, plus besoin de se poser de questions sur le choix des composants techniques à intégrer ; Seam est livré avec tous les composants nécessaires au développement d'une application web et, en plus, il permet de simplifier grandement les développements grâce à son modèle de composants unifié.

1.2 Historique

Le framework JBoss Seam est un framework Open Source à l'initiative de Gavin King (fondateur du framework Hibernate). A l'origine, le but de Seam était de permettre l'implémentation d'applications web *stateful* à partir de JSF 1.2 et des EJB3 (principaux concepts de JEE5), tout en s'affranchissant des problématiques liées au framework Hibernate.

Et comme JEE5 ne permet pas de couvrir tous les besoins techniques d'une application web, le framework Seam propose une extension de JEE5 avec des concepts novateurs et intègre également d'autres frameworks en provenance de la communauté Open Source.

Au final Seam est devenu un véritable framework d'intégration de JEE5 à la manière de Spring pour J2EE 1.4.

La version 1.0 de Seam est sortie en Juin 2006, la version 1.2.1 GA est la dernière version stable et la version 2.0 est attendue avant la fin 2007.

1.3 Concepts techniques

1.3.1 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.

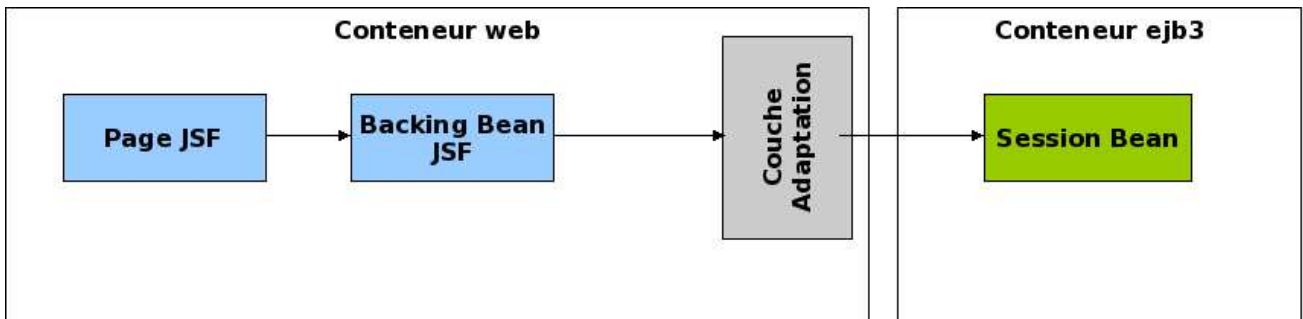


Figure 1: Intégration JSF/EJB 3 sans le framework Seam

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*.

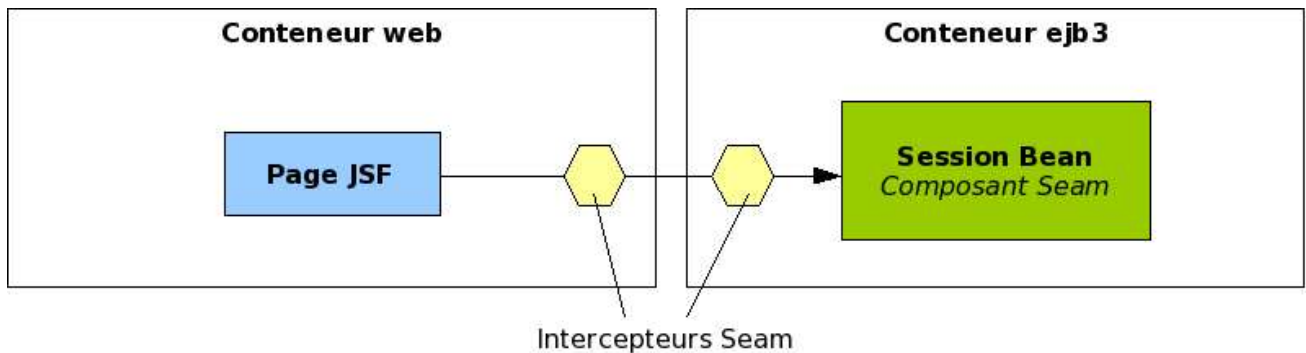


Figure 2: Intégration JSF/EJB3 avec le framework Seam

Un composant Seam est une classe java du type POJO ou EJB 3.0 qui contient l'annotation `@name`.

Exemple : déclaration d'un composant Seam du type *EJB3 Session Bean Stateful* :

```
@Stateful
@Name("manager")
public class ManagerAction {
}
```

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

```
<h:commandButton id="save" action="#{manager.save}" value="Save" />
```

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.

1.3.2 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,
- **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 est le contexte par défaut des composants du type EJB Stateful,
- **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).

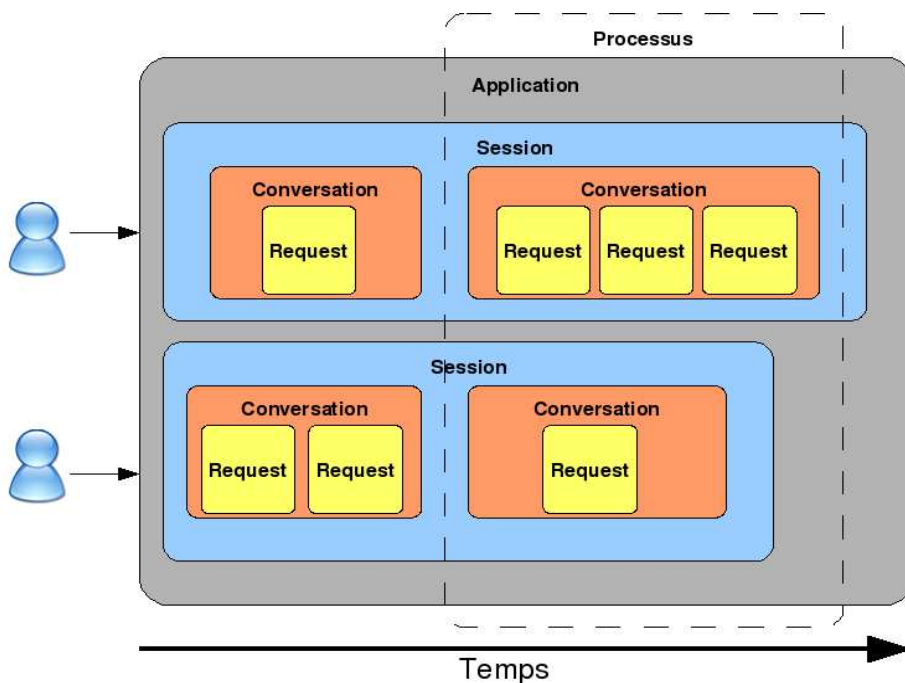


Figure 3: Les contextes Seam

La déclaration du contexte s'effectue à partir de l'annotation Java 5 `@scope`.

Dans l'exemple ci dessous le JavaBean `myBean` est déclaré dans le contexte de conversation :

```
@Name ("myBean")
@Scope (ScopeType.CONVERSATION)
public class MyBean {
}
}
```

1.3.3 Injection des 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.

Exemple :

```
@Name ("myBean")
public class MyBean {
    //Composant injecté par seam
    @In
    private ComponentA componentA;

    // Composant projeté par Seam
    // dans le contexte de Session
    @Out (scope=ScopeType.SESSION)
    private ComponentB componentB;
}
}
```

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.

1.3.4 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 déclarant 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.



De la même façon une conversation est terminée à partir de l'annotation `@End` sur une méthode ou bien dès qu'elle tombe en *timeout*.

Exemple :

```
@Name("myBean")
@Stateful
public class MyBean {

    // Resultats de recherche
    // stocké en conversation
    @Out
    private List<ComponentB> results;

    @Begin
    public void search(){
        results = doSearch();
    }

    @End
    public void cancel(){
    }

    private List doSearch() {
        //Business Code
    }

    @Destroy
    @Remove
    public void destroy() {
    }
}
```

Dans l'exemple ci-dessus, Seam nous réconcilie avec les EJB Stateful pourtant si décriés dans les précédentes versions de J2EE : l'annotation `@Destroy` qui, superposée à l'annotation EJB3 `@Remove`, permet de supprimer l'EJB stateful du conteneur EJB dès que la conversation est terminée ou bien en *timeout*.

Pour résumer, les conversations Seam permettent de développer des applications web stateful très robustes, en utilisant enfin des composants techniques prévus à cet effet (EJB Stateful) au lieu des sessions HTTP.

1.3.5 Intégration avec JPA

JPA (Java Persistence API) est l'API de persistance fournie par JEE5. Cette API se base en grande partie sur les concepts apportés par Hibernate, le framework incontournable de mapping relationnel objet (ORM).

JPA et Hibernate ont un mécanisme de chargement des données à la demande (Lazy Loading), pour éviter par exemple qu'un seul select sur une table parent récupère toutes les données des tables filles. Ce chargement des données à la demande n'est possible qu'à partir d'un contexte de persistance (ou session hibernate) et généralement celui-ci n'est plus accessible depuis une page JSF de l'application web, ce qui peut parfois poser des problèmes pour l'affichage des données.

Pour contourner ce problème, il y a deux solutions :



- implémenter une couche d'objets (DTO pour Data Transfert Object) permettant de charger les objets de persistance avec toutes les données nécessaires pour l'affichage,
- utiliser le contournement *OpenSessionInView* qui consiste à maintenir le contexte de persistance ouvert pour chaque requête HTTP.

Ces deux solutions, si elles fonctionnent, ont leurs limites. Dans le cas des DTO, on duplique le code et, avec la deuxième solution, on maintient le contexte de persistance ouvert pour toutes les requêtes HTTP et ce mécanisme ne convient pas vraiment aux applications JSF.

Avec Seam, la solution est beaucoup plus simple : le contexte de persistance sera réutilisé si la page web a besoin d'afficher des données supplémentaires. On a donc plus besoin d'implémenter de couche DTO.

1.3.6 Simplification de JSF

Développer des applications web avec JSF nécessite une bonne connaissance du mode de fonctionnement de l'API et surtout des moyens de contournements permettant de combler les lacunes actuelles de cette API.

Pour cela, Seam étend certaines phases du cycle de vie JSF de façon à pouvoir :

- garder les données en *request* même après un *redirect*,
- gérer simplement les événements utilisateurs sur un tableau de données à partir des composants *DataModel*,
- améliorer la gestion des erreurs qui se déclenchent lors d'une phase du cycle de vie JSF,
- enrichir le mécanisme de validation JSF à partir des annotations Hibernate Validator,
- mettre en place des *pageflows* robustes (possibilité de détecter le retour arrière sur un navigateur...)
- ...

Il faut noter que cette extension de JSF est compatible avec la plupart des implémentations open source JSF 1.1 ou 1.2 (Sun, MyFaces, IceFaces, RichFaces...).

D'autre part, Seam fournit également une bibliothèque de tags JSF (*Seam Controls*) qui étend l'API JSF de référence. Cette extension permet notamment de développer des applications web du type ReStFull (c'est à dire accessible en GET HTTP). Il faut savoir que, par défaut, une application web basée sur JSF utilise abusivement des requêtes HTTP en mode POST, il est donc par exemple très difficile de mettre un lien sur certaines pages d'une telle application.

1.3.7 Support Web 2.0

Le développement d'applications web 2.0 dites « riches » passe par l'utilisation massive de code Javascript embarqué sur le client (technologies Ajax). Actuellement, force est de constater qu'il y a pléthore de solutions techniques, plus ou moins exotiques, pour mettre en oeuvre ce type de technologie...

Seam nous simplifie la vie encore une fois, puisqu'il est livré avec des bibliothèques capables d'implémenter des applications Web 2.0 :

- la bibliothèque *Ajax4JSF* permet d'embarquer des technologies Ajax au niveau des composants JSF,

- le mécanisme de *Seam Remoting* permet à du code Javascript coté client d'invoquer de manière asynchrone un composant Seam Java déployé coté serveur,
- enfin Seam est compatible avec les bibliothèques JSF *RichFaces* et *IceFaces*, deux bibliothèques JSF open source permettant le développement d'applications web 2.0.

1.4 Utilisation de Seam en dehors de JEE5

JEE5 se diffusant encore trop lentement, les concepteurs de Seam ont donc prévu de pouvoir utiliser Seam en dehors de JEE5 à condition d'utiliser Java 5.

Une architecture possible est d'intégrer Seam avec le framework Spring :

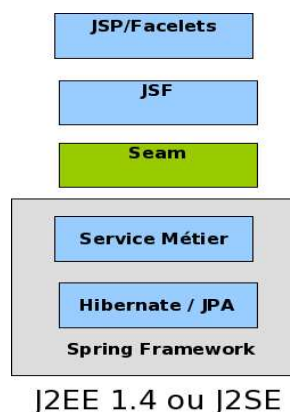


Figure 4: Architecture Seam avec Spring

Dans ce type d'architecture, Seam est utilisé avant tout pour simplifier les développements JSF, ce qui parfois dans le cas d'applications web complexes peut être un choix très intéressant et facilitera plus tard la migration vers JEE5.

Encore une fois, Seam nous facilite la tâche en s'intégrant parfaitement avec le framework Spring :

- chaque *bean* Spring peut être déclaré comme composant Seam et donc être injecté dans un autre composant Seam avec l'annotation `@In`,
- il est possible de partager des composants entre Seam et Spring, utile par exemple pour la gestion de la persistance.

De plus, avec la version 2.0 de Seam, l'intégration avec Spring sera encore améliorée (possibilité de piloter des transactions déclarées au niveau de Spring...)

2. Conclusion

Seam met à notre disposition un modèle de composants ouvert qui va bientôt devenir un standard (JSR 299), toutes les améliorations et extensions apportées par Seam permettant de simplifier les développements avec JEE5.

D'autre part, dans sa version actuelle, Seam couvre la plupart des besoins techniques et fonctionnels liés au développement d'une application web ce qui permet d'éviter les habituelles questions sur le choix des composants à intégrer...



Dans cette présentation nous n'avons abordé que les grands concepts mais Seam fournit également de nombreuses autres fonctionnalités :

- gestion de processus métiers et des *pageflows web* avec JBPM,
- gestion des évènements au niveau des composants,
- gestion des règles métiers avec Drools,
- généralisation de l'Expression Language (EL) pour le code Java et les fichiers de configuration,
- support PDF, internationalisation, gestion des thèmes,
- gestion de la sécurité et des autorisations,
- support CRUD,
- tests unitaires avec TestNG,
- génération de code avec l'outil *seam gen* (ressemble au *scaffold* de Ruby On Rails).

En ce qui concerne l'avenir :

- la version 2.0 de Seam (sortie prévue avant fin 2007) va apporter encore plus de fonctionnalités et plus de simplifications (support GWT, Hibernate Search...),
- le prochain IDE Red Hat Studio Developer basé sur Eclipse (en versions beta actuellement <http://www.redhat.com/developers/rhds/>) semble très prometteur et devrait permettre d'avoir une plate-forme de développement complètement intégrée avec Seam (génération de code, wizards...).

Avec tous ces atouts, le framework Seam semble bien placé pour devenir le framework incontournable pour le développement d'applications JEE5 et risque même d'en pousser d'autres vers la porte de sortie (Spring...).