

Programmation par composants avec les frameworks JSF et Seam

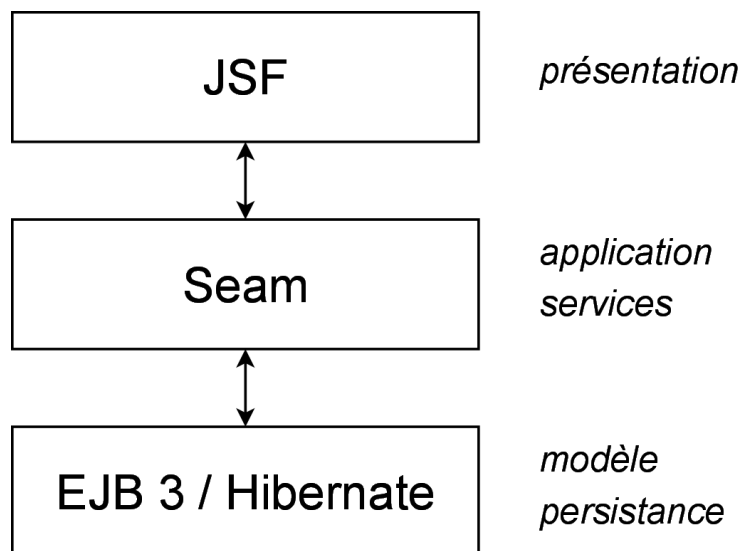
Thierry CHATEL, société IOCEAN

Architecture de la plateforme Seam

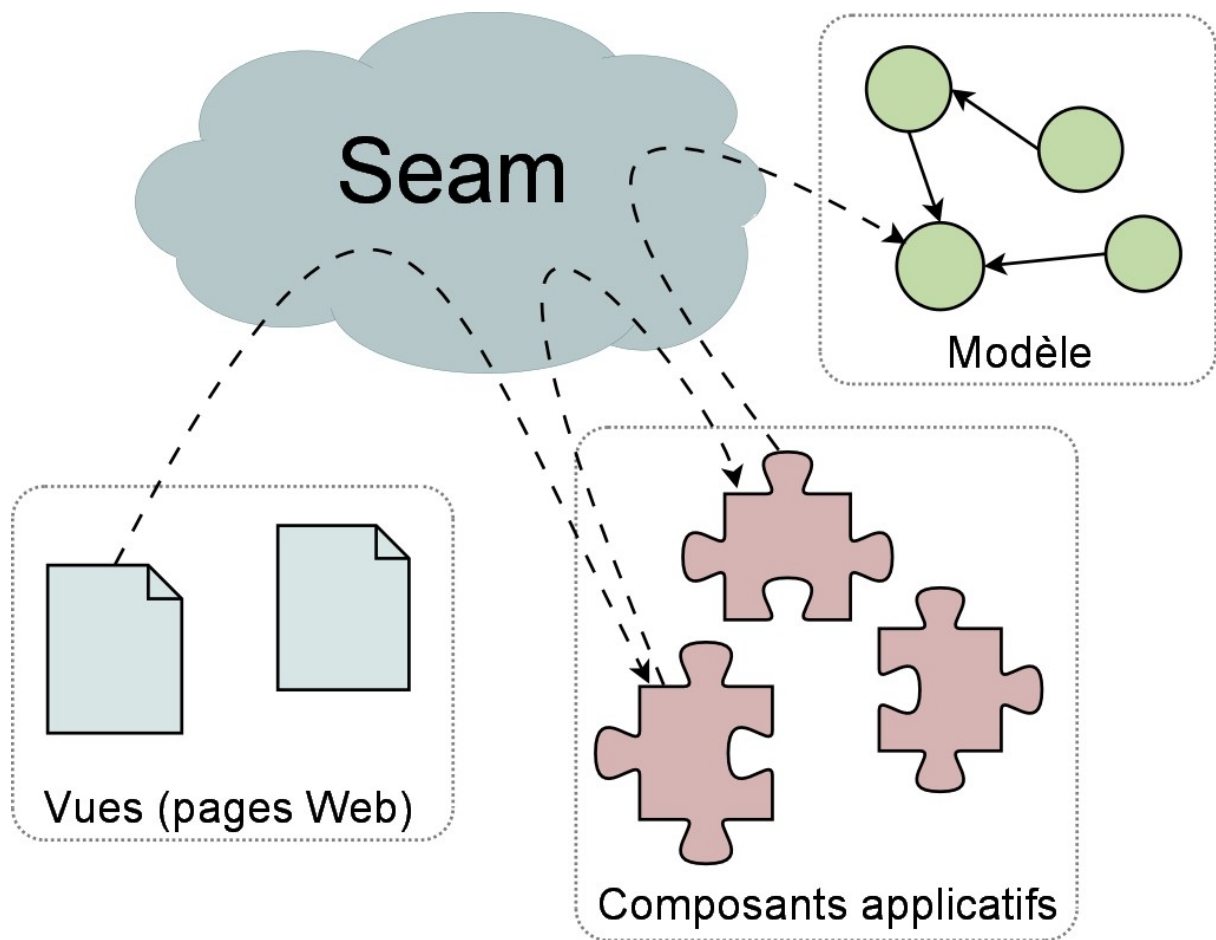
Seam est un framework open source de développement rapide d'application web, créé par JBoss. Le site web du framework se trouve à l'adresse : <http://www.seamframework.org/>



La plateforme Seam est un assemblage de trois frameworks :



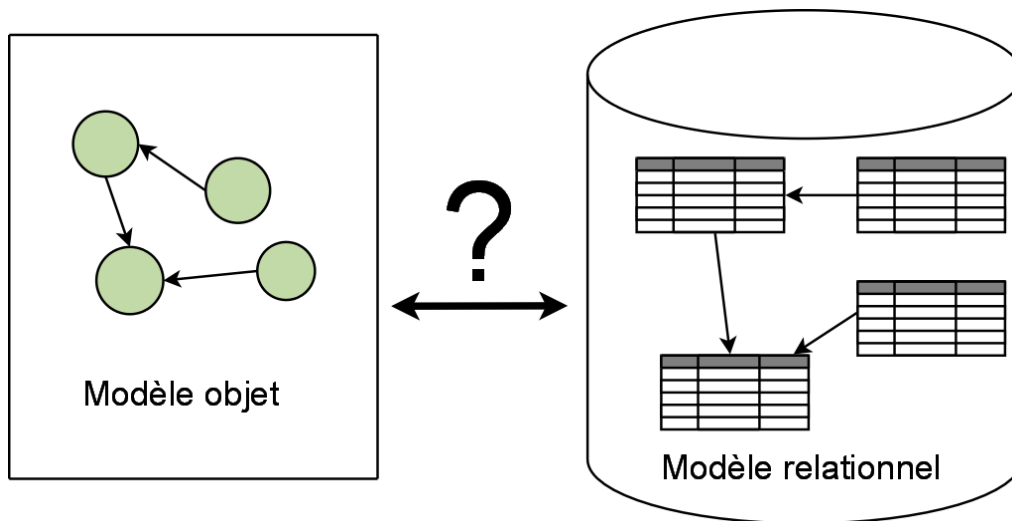
- JSF pour la couche de présentation
- EJB 3.0 (JPA) ou Hibernate pour la gestion de la persistance
- Seam lui-même pour la gestion des composants applicatifs et l'inversion de contrôle (IoC)



Seam agit comme un chef d'orchestre :

- les vues JSF (pages web ou portions de pages web) accèdent aux composants applicatifs par l'intermédiaire de Seam
- les composants applicatifs accèdent à d'autres composants applicatifs par l'intermédiaire de Seam
- les composants applicatifs accèdent aux entités persistantes du modèle via le contexte de persistance géré par Seam

Persistence avec EJB 3.0 (JPA) ou Hibernate



La plupart des applications écrites dans un langage objet doivent néanmoins sauvegarder leurs données dans une base relationnelle.

L'enregistrement d'un ensemble d'objets dans des tables relationnelles n'est pas quelque chose d'immédiat :

- correspondance une table pour une classe d'objet pas toujours optimale
- différences de paradigme entre un modèle objet et un modèle relationnel
 - modèle objet : classes, composition, encapsulation, héritage
 - modèle relationnel : tables et références

Un moteur d'**ORM** (Object-Relational Mapping) va gérer plus ou moins automatiquement la persistance (sauvegarde et relecture) d'un modèle objet dans une base relationnelle. Il va permettre :

- le mapping entre les classes d'objets et les tables
- l'écriture de requêtes portant sur les classes elles-mêmes

La plateforme JBoss Seam peut travailler avec l'une ou l'autre de ces deux solutions d'ORM :

- **JPA** (Java Persistence API), une norme qui est un sous-ensemble de la norme EJB 3.0
- **Hibernate**

Pour l'une ou l'autre de ces solutions, Seam fournit un ensemble de composants gérant automatiquement le contexte de persistance.

JSF et composants de présentation

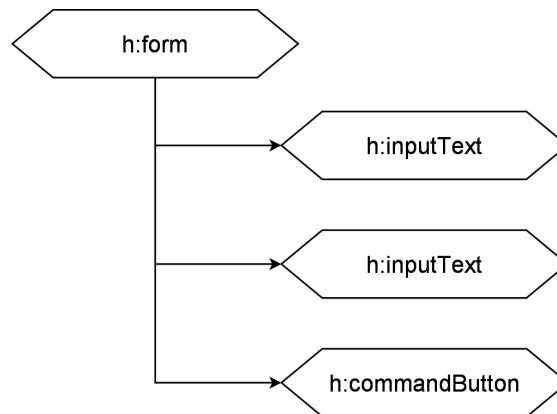
JSF est un framework de présentation Web, qui permet la création de vues (pages Web ou parties de pages Web) par assemblage de composants. C'est une norme faisant partie de la plateforme Java EE.

Il existe deux implémentations de la norme JSF :

- l'implémentation de référence de Sun
- le projet Apache MyFaces

Vue JSF = hiérarchie de composants

Une vue JSF est structurée sous la forme d'une hiérarchie de composants JSF.



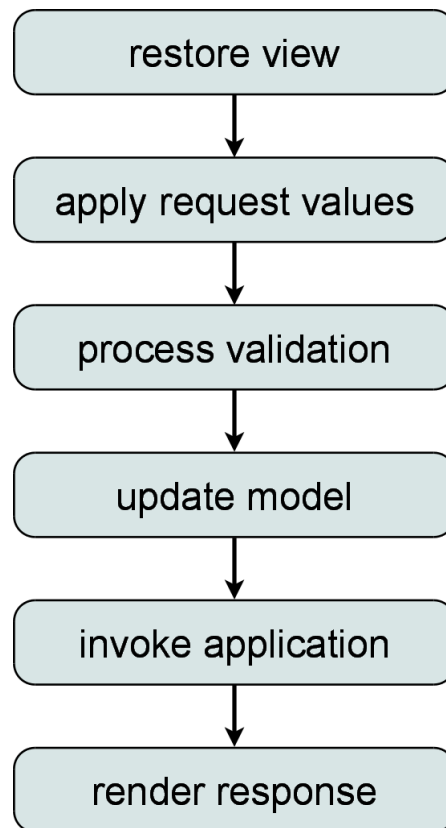
```
<h:form>
  <h:inputText value="#{userHome.name}" />
  <h:inputText value="#{userHome.firstname}" />
  <h:commandButton value="Save" action="#{userHome.save}" />
</h:form>
```

Plusieurs technologies peuvent être utilisées pour la création de vues JSF :

- JSP, dont la nature pas vraiment hiérarchique peut poser problème
- Facelets, qui est la technologie préconisée sur la plateforme Seam, et qui utilise des fichiers XHTML, strictement hiérarchiques, pour l'écriture des vues

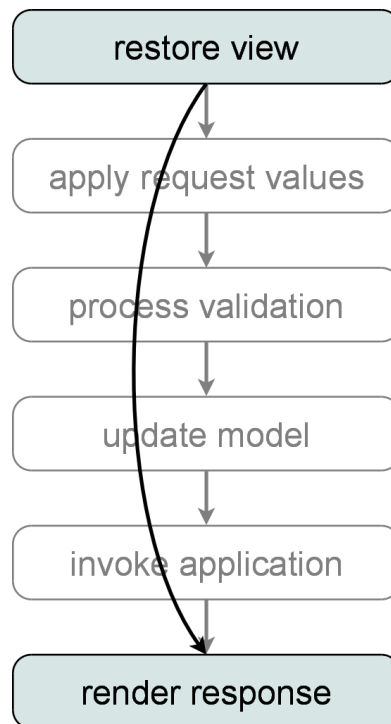
Cycle de vie des pages JSF

Le traitement d'une requête pour l'affichage d'une page Web par JSF va être composé des six étapes suivantes :



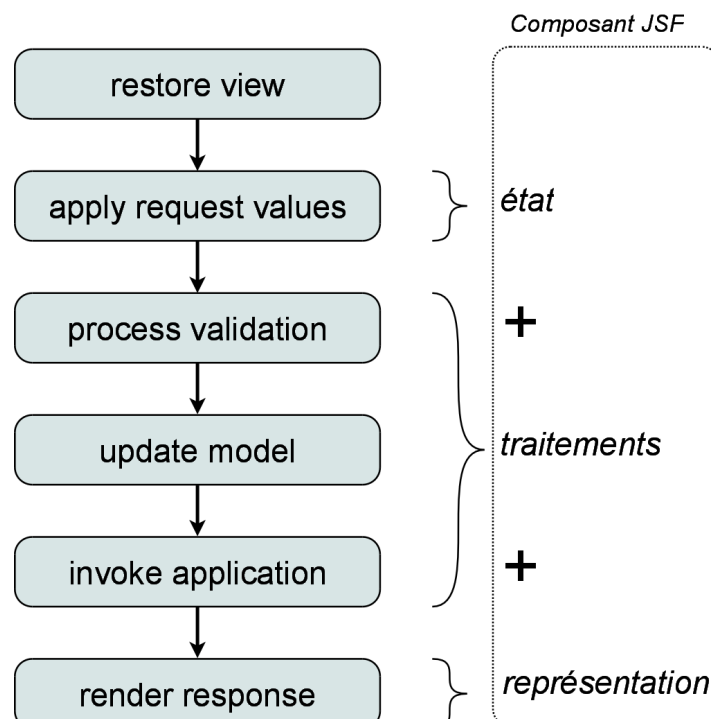
Ces six étapes ne sont toutes exécutées que dans le cas d'une requête correspondant à la soumission d'un formulaire (requête HTTP de type POST).

Dans le cas du premier affichage d'une page (requête HTTP de type GET), la plupart des étapes sont sautées :



Nature des composants JSF

Un composant JSF va avoir une représentation dans la page Web résultat, il va pouvoir conserver un état (données propres au composant) et effectuer des traitements.



Configuration des composants JSF dans les vues

Chaque composant JSF définit un ensemble de propriétés, qui vont permettre de le configurer, et peut recevoir aussi un contenu.

Les composants JSF sont placés dans les vues sous la forme de balises XML, avec le nom du composant, généralement un ou plusieurs attributs définissant les valeurs de certaines propriétés, et éventuellement un contenu.

Composant JSF avec deux attributs (*value* et *action*) et sans contenu :

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

Composant JSF avec contenu :

```
<h:form>
  ...contenu...
</h:form>
```

Les composants ayant un contenu vont en utiliser les divers éléments, souvent pour en créer une représentation particulière. C'est le cas du composant `<h:dataTable>` qui attend dans son contenu un entête facultatif, des colonnes pouvant elles-mêmes avoir un entête...

```
<h:dataTable value="#{tableBean.items}" var="item" >
  <f:facet name="header">
    <h:outputText value="dataTable demo" />
  </f:facet>
  <h:column>
    <f:facet name="header">
      <h:outputText value="id" />
    </f:facet>
    <h:outputText value="#{item.id}" />
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="name" />
    </f:facet>
    <h:outputText value="#{item.name}" />
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="phone" />
    </f:facet>
    <h:outputText value="#{item.phone}" />
  </h:column>
</h:dataTable>
```

Il y a peu d'interaction entre les composants JSF eux-mêmes. Quand un composant JSF doit agir sur un autre composant, en dehors de son contenu, il lui fait référence via son identifiant (attribut « id »).

C'est le cas de la bibliothèque de composants Ajax4Jsf, dont les composants ont une propriété **reRender** permettant d'indiquer l'identifiant d'un ou plusieurs composants devant être actualisés après une requête Ajax :

```
<a4j:form>
  <h:panelGrid columns="3">
    <h:outputText value="Name:" />
    <h:inputText value="#{userBean.name}" />
    <a4j:commandButton value="Say Hello" reRender="out" />
  </h:panelGrid>
</a4j:form>

<h:panelGroup id="out">
  <h:outputText value="Hello "
    rendered="#{not empty userBean.name}" />
  <h:outputText value="#{userBean.name}" />
  <h:outputText value="!"
    rendered="#{not empty userBean.name}" />
</h:panelGroup>
```

Bibliothèques de composants JSF

Il existe diverses bibliothèques, open sources ou commerciales, de composants JSF prêts à l'emploi, plus sophistiqués que les composants de base fournis avec le framework.

RichFaces :

RichFacesLiveDemo
Download Support Developer Guide

Column Group

- Developers Guide
- Tld Documentation
- Component Class Documentation

Ajax Support

Resources:Beans Handling

Ajax Validators

Ajax Output

Ajax Miscellaneous

Data Iteration

- Column
- Column Group
- Columns
- Data Definition List
- Data Filter Slider
- Data Grid
- Data List
- Data Ordered List
- Data Table
- Data Table Scroller
- Extended Data Table
- Repeat
- Scrollable Data Table
- Table Filtering

Drag-Drop Support

Rich Menu

Rich Trees

Rich Output

Rich Input

Rich Selects

Skins: BlueSky Laguna Glass-X Dark-X Classic more...

ColumnGroup

Usage Extended Data Model Tag Information

DataTable allows to show a tabular data. Additional to the standard <h:dataTable>, this component enables row and column spans for columns, a flexible layout for a header and a footer. DataTable supports "master-detail" pattern and allows to show the combination of a master table and detail sub-tables. The following example shows the dataTable component in use:

DataTable Column/ColumnGroup example

	Expenses			subtotals
	Meals	Hotels	Transport	
San Jose				
25-Aug-97	\$37.74	\$112.00	\$45.00	
26-Aug-97	\$27.28	\$112.00	\$45.00	
	\$65.02	\$224.00	\$90.00	\$379.02
Seattle				
27-Aug-97	\$96.25	\$109.00	\$36.00	
28-Aug-97	\$35.00	\$109.00	\$36.00	
	\$131.25	\$218.00	\$72.00	\$421.25
Totals	\$196.27	\$442.00	\$162.00	\$800.27

[View Source](#)

[RichFaces: Most Important Links](#) v.3.2.2.GA SVN \$Revision: 8947 \$

IceFaces :

Component Suite Showcase

Rime

- Extended Components
- Custom Components**
- Components
 - Autocomplete
 - Calendar
 - Charts
 - Connection Status
 - Drag & Drop
 - Effects
 - File Upload
 - Google Maps
 - Media
 - Menu Bar
 - Menu Popupp
 - Progress Bar
 - Rich Text
 - **Tree**
 - Table
- Layout Panels

Tree Component Description Source

The tree component displays hierarchical data as a "tree" of branches and leaf nodes.
Move nodes up and down using the commandButtons at the bottom of the page.

- Employees
 - Western
 - Smith, Ethan
 - **Smith, Jacob**
 - Williams, Daniel
 - Brown, Andrew
 - Central
 - Brown, Joshua
 - Brown, Christopher
 - Miller, Vincent
 - Davis, Matthew
 - Eastern
 - Garcia, Alexander
 - Garcia, Jacob
 - Garcia, Benjamin
 - Rodriguez, Joshua

Server-side Backing Bean Values:

Path: Employees Western Smith, Jacob

Employee ID: 15

Name: Jacob Smith

Phone: 555-4563

Copyright 2008 ICEsoft Technologies Inc.

Powered By ICEFaces

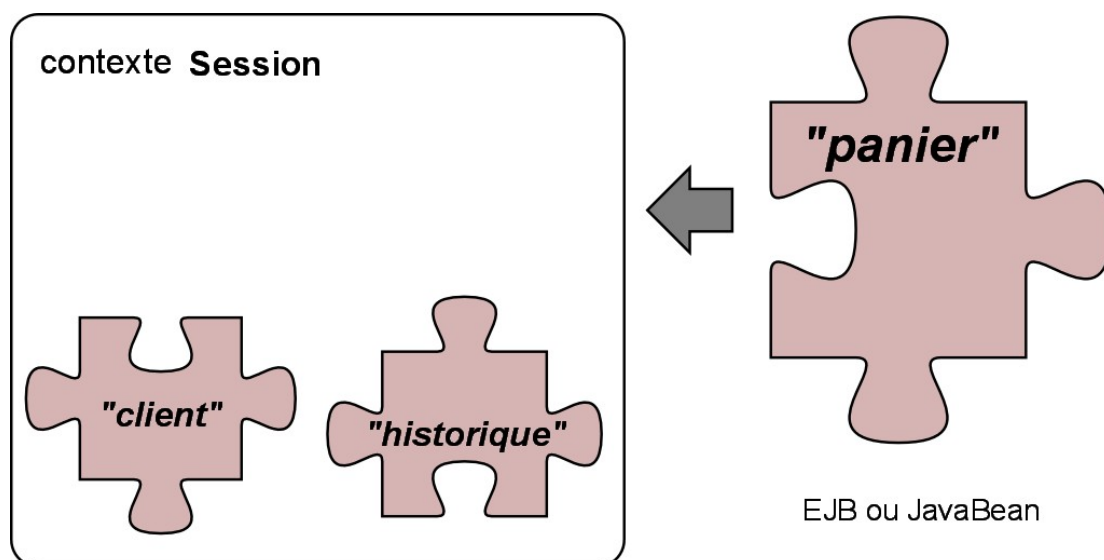
Seam et composants applicatifs

Notion de composant Seam

Les composants Seam sont des objets Java ordinaires (POJO = Plain Old Java Objects), qui généralement conservent un état, bien que ce ne soit pas obligatoire :

- JavaBeans
- EJB 3.0 Stateless Session Beans
- EJB 3.0 Statefull Session Beans

Ils sont publiés dans un **contexte**, en leur donnant un **nom** unique.



Le framework Seam gère lui-même les dépendances entre les composants.

Le nom est attribué au composant Seam en ajoutant une annotation `@Name` à la classe Java.

```
@Name ("panier")
public class Panier {
    ...
}
```

Contextes Seam

Chaque composant Seam est publié dans un contexte.

Il ne va exister qu'une seule instance d'un composant nommé dans chaque instance du contexte. Pour une instance donnée du contexte, Seam fournira le composant existant s'il s'y trouve déjà, ou en créera une nouvelle instance s'il n'y est pas encore. Le composant restera dans le contexte jusqu'à l'expiration de celui-ci.

Seam définit un ensemble de contextes correspondant aux divers niveaux de partage et de durée de vie des données dans une application Web.

Stateless

Event

Page

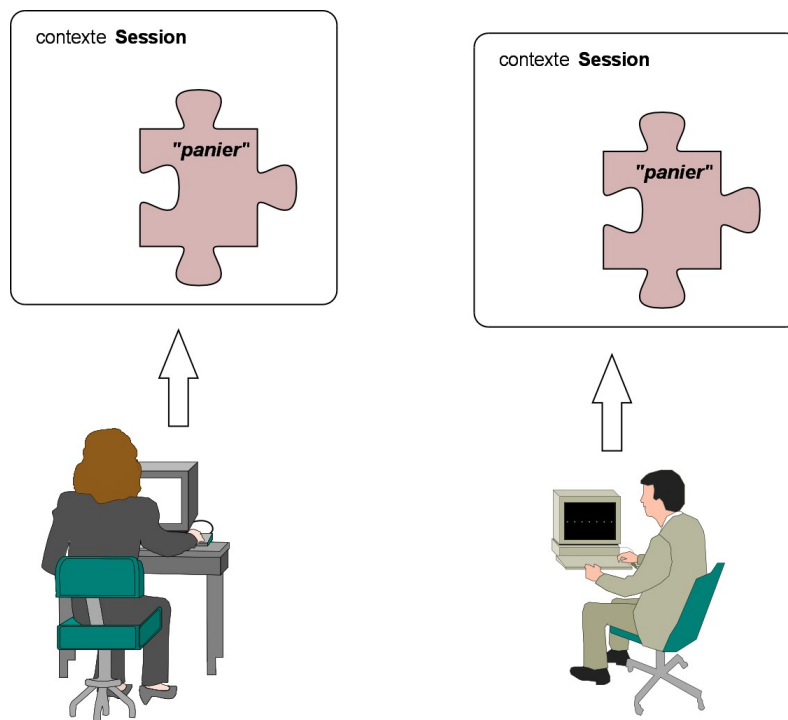
Conversation

Session

Business process

Application

- le contexte **Stateless** sert pour les composants sans état
- le contexte **Event** est valide le temps d'une requête
- le contexte **Page** est conservé tant qu'on effectue des actions sur la même page
- le contexte **Conversation** correspond à un enchaînement de pages, de type « wizard »
- le contexte **Session** est partagé par toute la session d'un utilisateur
- le contexte **Business Process** est utilisé par le moteur de workflow intégré à Seam
- le contexte **Application** est commun à toute l'application web, et dure jusqu'à l'arrêt du serveur d'application



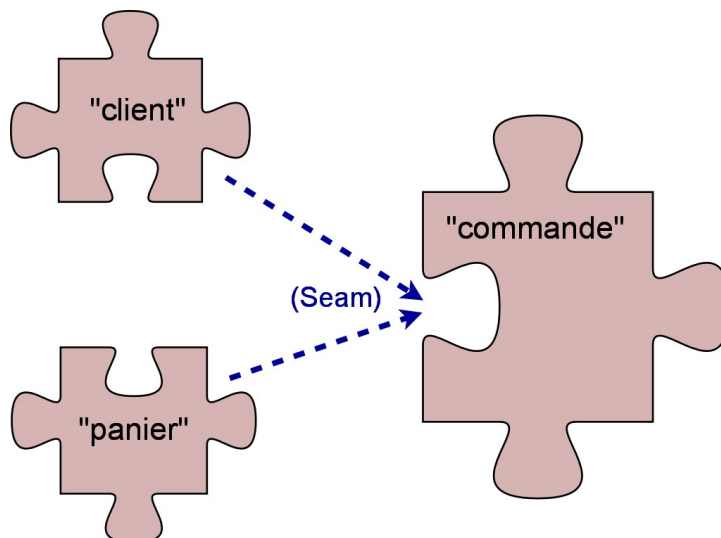
Le contexte Session, par exemple, est spécifique à chaque utilisateur connecté, donc chacun y verra sa propre instance d'un composant qui y est publié.

Le contexte est spécifié en ajoutant une annotation `@Scope` à la classe Java définissant le composant.

```
@Name ("panier")
@Scope (ScopeType.SESSIION)
public class Panier {
    ...
}
```

Inversion de contrôle et injection de dépendances

Les composants applicatifs Seam vont fréquemment avoir besoin d'utiliser d'autres composants Seam. L'idée de l'inversion de contrôle (IoC en anglais) 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 quelles sont ses *dépendances*, c'est-à-dire de quels composants il a besoin, et le framework va automatiquement les lui *injecter*.

On indique une dépendance devant être injectée en ajoutant une annotation `@In` à une variable d'instance d'un composant :

```
@In
private Session sessionHibernate;

public void initDB() {
    lignes = sessionHibernate
        .createQuery(" from LigneCatalogue")
        .list();
    ...
}
```

Par défaut, le framework va chercher un composant du nom de la variable, et l'affecter à la variable avant tout appel de méthode.

Si la variable et le composant à injecter n'ont pas le même nom, on précise le nom du composant dans la propriété *value* de l'annotation *@In* de la variable :

```
@In(value="sessionHibernate")
private Session session;
```

Le framework Seam s'attend à ce que le composant à injecter existe, dans le cas contraire il signale une erreur. S'il est possible que le composant n'existe pas, on peut soit mettre :

```
@In(required=false)
```

pour indiquer que le composant n'est pas requis, soit mettre :

```
@In(create=true)
```

pour demander au framework d'instancier le composant s'il ne le trouve pas.

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.

```
@Out
private User user;

public void login(String login, String password) {
    ...
    this.user = new User(...);
}
```

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.

Configuration externe des composants

Les composants Seam livrés prêts à l'emploi, notamment ceux fournis avec le framework, peuvent être configurés de façon externe dans le fichier WEB-INF/components.xml.

```
<persistence:hibernate-session-factory
    name="hibernateSessionFactory"/>

<persistence:managed-hibernate-session
    name="sessionHibernate"
    auto-create="true"
```

```
        session-factory="#{hibernateSessionFactory}" />
<transaction:hibernate-transaction
    session="#{sessionHibernate}" />
```

Cette possibilité de configuration externe des composants permet de livrer des composants standards packagés dans des fichiers .jar, et de les configurer dans l'application qui les utilise sans avoir à modifier les classes Java.

Interconnexion événementielle

Le framework Seam permet également de connecter les composants par un couplage faible basé sur des événements. Ce type d'interconnexion est intéressant lorsqu'on veut éviter un couplage plus fort basé sur de l'injection de dépendance, puisqu'il ne nécessite aucune dépendance entre les classes Java des composants.

Un composant déclenche un événement, et les composants à l'écoute de cet événement sont alors activés automatiquement.

Méthode d'un composant déclenchant un événement (annotation **@RaiseEvent**) :

```
@Name("helloWorld")
public class HelloWorld {

    @RaiseEvent("hello")
    public void sayHello() {
        FacesMessages.instance().add("Hello World!");
    }

}
```

Méthode d'un autre composant exécutée automatiquement lorsque l'événement est déclenché (annotation **@Observer**) :

```
@Name("helloListener")
public class HelloListener {

    @Observer("hello")
    public void sayHelloBack() {
        FacesMessages.instance().add("Hello to you too!");
    }

}
```

Les méthodes de composants devant être exécutées lorsqu'un événement est déclenché peuvent aussi être spécifiées de façon externe dans le fichier **components.xml** :

```
<event type="hello">
```

```
<action execute="#{helloListener.sayHelloBack}"/>
  <action execute="#{logger.logHello}"/>
</event>
```

Utilisation des composants Seam dans les vues JSF

Le framework Seam permet de référencer directement les composants applicatifs dans les vues JSF (pages web ou parties de pages web).

Langage d'expression JSF EL

Le langage d'expression **JSF EL** (*JSF Expression Language*) permet, dans un attribut d'un composant JSF, d'utiliser une formule pour le lier à une propriété d'un objet Java, ou à une méthode à exécuter.

Les expressions JSF EL sont délimitées par `#{...}`

Les propriétés référencées doivent correspondre à la norme JavaBean, avec un getter normalisé (`getProp()`, ou `isProp()` pour une propriété booléenne), et un setter normalisé (`setProp(...)`) si la propriété doit être mise à jour.

L'accès à une **propriété** se fera à partir du nom d'un composant Seam, comme première partie de l'expression. On peut enchaîner plusieurs propriétés comme suit :

```
<h:outputText value="#{login.user.name}" />
<h:inputText value="#{login.user.name}" />
```

Une **méthode** (sans paramètre) est indiquée comme une propriété, pour des attributs dans lesquels les composants JSF attendent une méthode :

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

Le framework Seam ajoute une extension au langage JSF EL, qui permet dans une expression d'appeler des méthodes Java en leur passant un ou plusieurs paramètres, placés dans ce cas entre parenthèses :

```
<h:commandButton value="Save" action="#{service.save(user)}" />
```