3.3.2 Le fonctionnement

Sommaire :

- Les points d'entrée
  - Déclaration AOP
  - Déploiement en servlet
  - Déploiement en portlet
  - Web services

Les points d'entrée

En *esup-commons V1* on devait modifier les différent points d'entrée de l'application (**FacesServlet**, **FacesPortlet**, **XFireServlet**). Cette approche avait 2 désavantages :

1. On était obligé de maintenir ces points d'entrées et il fallait reproduire ce travail si on voulait avoir un nouveau point d'entrée (servlet d'un framework REST par exemple)
2. Ce qui était fait l'était notamment pour JSF (**FacesServlet**, **FacesPortlet**) ce qui rendait *esup-commons V1* très dépendant de la technologie utilisée pour la vue.

C'est la raison pour laquelle *esup-commons V2* utilise une technologie radicalement différente qui est l'utilisation de AOP (*aspect-oriented programming* via Spring AOP) pour gérer les connexions aux bases de données et les transactions.

Déclaration AOP

La gestion des transactions se fait au niveau des appels métier (fichier **resources/properties/domain/domain.xml**). Exemple :

```
<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:jaxws="http://cxf.apache.org/jaxws"
 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/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
 http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
 http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

<aop:config>
 <aop:pointcut id="domainMethods"
  expression="execution(* org.esupportail.*.domain.DomainServiceImpl.*(..))" />
 <aop:advisor advice-ref="txAdvice" pointcut-ref="domainMethods" />
</aop:config>

<tx:advice id="txAdvice" transaction-manager="txManager">
 <tx:attributes>
  <tx:method name="add*" propagation="REQUIRED" />
  <tx:method name="delete*" propagation="REQUIRED" />
  <tx:method name="update*" propagation="REQUIRED" />
  <tx:method name="*" propagation="SUPPORTS" read-only="true"/>
 </tx:attributes>
</tx:advice>

<bean id="domainService" class="org.esupportail.example.domain.DomainServiceImpl">
 <property name="daoService" ref="daoService" />
</bean>
```

Explications :

- La racine **<beans>** contient la définition des espaces de nom **aop:** et **tx:** ainsi que la localisation des schémas XSD ad hoc- qui sont utilisés dans la suite du fichier.
- La balise **<aop:config>**permet de préciser deux choses :
  - Le point de coupe (**pointcut**). C'est-à-dire l'emplacement où va être inséré dynamiquement le code AOP. L'attribut **expression** permet de préciser que l'on va cibler toutes les méthodes, quelle qu'en soit la signature, de la classe **DomainServiceImpl** dans un package donc le path commence par **org.esupportail** et fini par **domain**.
  - La référence au gestionnaire de transactions à brancher sur ce point de coupe.
- La balise **<tx:advice>** permet de préciser la nature de la transaction. Ici on précise, via **propagation="REQUIRED"** que l'on veut créer une transaction, si elle n'est pas encore présente, sur toutes les méthodes dont le nom commence par **add**, **delete** ou **update**.

Sur les autres méthodes la transaction est conservée si elle existe mais elle ne sera pas créé si seulement ces méthodes sont appelés.

Si vous avez besoin de gérer des commit et rollback sur d'autres méthodes que celles commençant part **add**, **delete** ou **update** alors il faut ajouter une balise **<tx:method>** ad hoc.

Déploiement en servlet

En *esup-commons V1* pour *JSF*, il fallait utiliser la *servlet* **FacesServlet** offerte par *esup-commons* (**org.esupportail.commons.web.servlet.FacesServlet**) qui gérait les sessions *Hibernate*. Ce n'est plus utile en *esup-commons V2* grâce à l'utilisation de APO. La servlet utilisée est celle de JSF (dans **web.xml**) :

```
<servlet>
 <description> The main servlet of the application. This class inherits
  from the MyFaces implementation and handles exceptions thrown for
  specific exception handling.
 </description>
 <display-name>Faces Servlet</display-name>
 <servlet-name>Faces Servlet</servlet-name>
 <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
 <servlet-name>Faces Servlet</servlet-name>
 <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

Par contre, avec JSF et JPA on peut faire une requête à la base de données dans la couche métier pour récupérer un objet et avoir besoin de naviguer des les fils de ce dernier au moment de l'affichage de la vue. Dans ce cas on peut avoir besoin de maintenir l'accès à la base de données pendant le temps du rendu de la vue afin d'éviter les erreurs de type **LazyLoadingException**. Pour ce faire on utilise un listerner particulier (dans le **web.xml**) :

```
<filter>
 <display-name>JPA Filter</display-name>
 <filter-name>JPA Filter</filter-name>
 <filter-class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-class>
</filter>

<filter-mapping>
 <filter-name>JPA Filter</filter-name>
 <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
```

Déploiement en portlet

En *esup-commons V1* pour *JSF*, il fallait utiliser la *servlet* **FacesPortlet** offerte par *esup-commons* (**org.esupportail.commons.web.portlet.FacesPortlet**) qui gérait les sessions *Hibernate*. Ce n'est plus utile en *esup-commons V2* grâce à l'utilisation de APO. La portlet utilisée est celle de la spécification *portlet bridge* pour JSF (dans **portlet.xml**)

En mode *portlet*, il faut utiliser la *servlet* **PortletServlet** offerte par *esup-commons*, et qui gère également les sessions *Hibernate* et les exceptions (cf **/web app/WEB-INF/web.xml**) :

```
<portlet>
 <portlet-name>esup-example</portlet-name>
 <display-name xml:lang="fr">esup-example</display-name>
 <portlet-class>javax.portlet.faces.GenericFacesPortlet</portlet-class>
```

Par contre, avec JSF et JPA on peut faire une requête à la base de données dans la couche métier pour récupérer un objet et avoir besoin de naviguer des les fils de ce dernier au moment de l'affichage de la vue. Dans ce cas on peut avoir besoin de maintenir l'accès à la base de données pendant le temps du rendu de la vue afin d'éviter les erreurs de type **LazyLoadingException**. Le listerner utilisé en mode servlet n'est pas utilisable en mode portlet. ESUP-Commons propose un phaseListener JSF qui offre les mêmes services. Il faut le configurer dans le fichier **webapp/WEB-INF/jsf/faces-config.xml** :

```
<phase-listener>org.esupportail.commons.jsf.PortletPhaseListenerWithJPA</phase-listener>
```

En mode portlet on préférera sans doute faire appel à un Web Service exposé par une application métier et ne pas directement faire des appels à la base de données. Dans ce cas le phaseListener à utiliser sera org.esupportail.commons.jsf.PortletPhaseListener.

Web services

En *esup-commons V1*, il fallait utiliser la servlet **XFireServlet** offerte par *esup-commons* (**org.esupportail.commons.web.servlet.XFireServlet**) qui gérait les sessions *Hibernate*. Ce n'est plus utile en *esup-commons V2* grâce à l'utilisation de AOP.