Compte rendus sur l'enregistrement d'un client comme listener dans ActiveBpel

1. Premiére étude.

Nous voulions offrir aux clients la possibilitée de s'enregistrer comme listener afin d'intercepter chaque modifications au niveau du moteur d'ActiveBpel ainsi que pour chaque processus. Nous avions auparavant réalisé une étude sur ActiveBpel afin d'obtenir un premier service ou le client a la possibilitée de demander lui même l'état de tout le systéme, nous avons donc continué a utiliser le WebService BpelEngineAdmin. Les méthodes aux signatures suivantes aurait apparemment pour but d'enregistrer le client comme listener :

- public void **addEngineListener**(long aContextId,
  java.lang.String aEndpointURL)
  throws java.rmi.RemoteException,
  AeBusinessProcessException

- public void **addProcessListener**(long aContextId,
  long aPid,
  java.lang.String aEndpointURL)
  throws java.rmi.RemoteException,
  AeBusinessProcessException

Nous avons ensuite fait une recherche dans les sources d'ActiveBpel à notre disposition, afin de trouver un classe implémentant le concept de listener comme on pourrait le voir en Java. Nous avons trouver une classe se rapprochant de notre idée, cette classe est **org.activebpel.rt.bpel.server.admin.rdebug.server.AeRemoteDebugImpl** qui implémente l'interface **org.activebpel.rt.bpel.server.admin.rdebug.server.IAeBpelAdmin** qui a la particularitée d'être remote. Ce qui nous a intéresser dans la classe **AeRemoteDebugImpl** c'est qu'elle contient des classes internes *protected* qui implémente entre autre les interfaces **org.activebpel.rt.bpel.IAeProcessListener** et **org.activebpel.rt.bpel.IaeEngineListener** . Nous sommes donc partis de ces classes la et nous avons ajouter à notre composant de test une nouvelle interface de type **org.activebpel.rt.bpel.IaeEngineListener** à laquelle nous avons attaché un proxy construit avec Fractal2WS repésentant le Web Service **IaeEngineListener** que nous pensions être l'écouteur. Nous avons donc fait un addEngineListener avec aContextId = 1 et aEndpointURL= **http://localhost:8080/axis/services/IaeEngineListener**. Nous avons fait implémenter l'interface **IaeEngineListener** a notre composant test et implémenter les méthodes de signature :

- public boolean **handleEngineEvent**(IAeEngineEvent aEvent)
- public void **handleAlert**(IAeEngineAlert aEvent)

avec de simple traces.

Malgrés celà nous n'avons pas reussi a recevoir les informations d'ActiveBpel, une TargetException nous est renvoyer au moment du addEngineListener et nous n'avons aucun moyen d'en savoir plus. Nous avons donc mis de coté notre idée de listener.

2. Lundi 28 avril : Reprise de l'étude du fonctionnement des listener.

Nous avons decidés de nous remettre dans l'étude du fonctionnement des listener dans ActiveBpel car celle-ci permet de nous enlever certaines contraintes liées au fait que le client doit lui même aller demander les informations au moteur d'orchestration. Nous avons donc continuer notre recherche sur les forums dédiés à ce sujet sans aucun résultat très significatif, les classes qui nous interesse fesant partis des tutoriaux de la version payante du moteur d'orchestration comme le dis un des membres du forum officiel :

« There are no code samples that use AeEngineListener, because so far it is used only in our commercial products ».

Ne sachant pas trop comment résoudre le probléme, nous avons donc décider de recompiler les sources d'ActiveBpel nous même et d'ajouter nos propres traces afin de voir l'endroit ou cela bloque. Nous avons donc retenter l'expérience de l'ajout d'un listener comme précedemment et nous avons cibler l'endroit qui n'allais pas. Il s'agit de la classe **AeRemoteDebugImpl** dans la méthode ayant pour implémentation (les System.out.println ainsi que le e.printStackTrace() sont nos ajouts) :

```
protected static IAeEventHandlerService getEventHandlerServiceLocator(String
aEndpointURL)
      throws AeBusinessProcessException
   {
      if ( AeUtil.isNullOrEmpty(sEventHandlerLocator) )
         throw new AeBusinessProcessException(
               AeMessages.getString("AeRemoteDebugImpl.ERROR_12")); //$NON-
NLS-1$

      try
      {
      System.out.println("Dans getEventHandlerServiceLocator de
AeRemoteDebugImpl, avant chargement sEventHandlerLocator " +
sEventHandlerLocator);
        //sEventHandlerLocator="org.activebpel.rt.axis.bpel.rdebug.client.AeEven
tHandlerLocator");
        Class c = Class.forName(sEventHandlerLocator);
      System.out.println("Dans getEventHandlerServiceLocator de
AeRemoteDebugImpl, avant construction constructeur");
         Constructor constructor = c.getConstructor( new Class[]
{String.class} );
         return (IAeEventHandlerService) constructor.newInstance( new Object[]
{aEndpointURL} );
      }
      catch (Exception e)
      {
      System.out.println("Dans getEventHandlerServiceLocator de
AeRemoteDebugImpl, erreur");
      e.printStackTrace();
         throw new
AeBusinessProcessException( MessageFormat.format(AeMessages.getString("AeRemoteD
ebugImpl.ERROR_13"), //$NON-NLS-1$
                                                                  new Object[]
{sEventHandlerLocator}), e);
      }
   }
```

Nous avons ainsi remarqué qu'aprés l'instruction :

```
Class c = Class.forName(sEventHandlerLocator);
```

plus aucunes informations ne nous parviens, alors qu'une classe repondant au nom qui est mis en paramétre existe bien. Nous avons donc essayer de rajouter du code static dans la classe qui était sencé être chargée, ainsi qu'un constructeur par defaut au cas ou, et la toujours rien. Nous avons donc cherché a tester la méthode `getEventHandlerServiceLocator` en dehors d'ActiveBpel, en créant un simple programme. Nous avons donc chargé le projet, qui avait déjà été compilé une fois en ligne de commande avec ant, dans Eclipse et nous avons fait notre expérience, qui a reussi puisque la classe qui devait être chargée a bien été trouvée. Nous avons donc recompiler le projet en ligne de commande avec ant afin de créer de nouveau jar pour ActiveBpel (et cela sans aucune modification de code, le seul ajout étant simplement des traces d'execution) et la nous nous sommes aperçus que le jar ae_wsio.jar d'ActiveBpel était passer de la taille de 17ko à 3.5 Mo. Quattre packages se sont donc ajouté au moment de la compilation répondant au nom de ddl,rt,timer et work et cela simplement à cause du fait que le projet ai été mis dans Eclipse (nous avons vérifié à plusieurs reprise que cela ne venez pas du code que nous avions ajouté, en fesant la même expérience avec des sources d'ActiveBpel sortis directement de l'archive qu'il nous fournisse).

Nous avons donc retenter l'experience de l'ajout d'un listener et la plus aucune erreur, le message passe bien, par contre nous ne sommes toujours pas notifier. Nous avions des hésitations sur la classe d'implémentation du listener, celui que nous avions, implémenté l'interface `IaeProcessListener`, mais celle-ci ne semblait pas convenir nous avons donc regarder plus précisément le code d'ActiveBpel pour le traitement de cette partie. Dans la classe interne a **AeRemoteDebugImpl** , **`protected static class AeProcessListener implements IaeProcessListener`**, nous avons pus voir dans le constructeur l'instruction suivante :

```
mHandler =
AeRemoteDebugImpl.getEventHandlerServiceLocator(aEndpointURL).getRemoteDebugServ
ice();
```

mHandler représente simplement l'adresse URL de notre listener et toujours dans cette même classe interne dans la méthode **`public boolean`** `handleProcessEvent(IAeProcessEvent aEvent)`, nous avons l'instruction :

```
mHandler.processEventHandler(mContextId, aEvent.getPID(), aEvent.getNodePath(),
                     aEvent.getEventID(), aEvent.getFaultName(),
aEvent.getAncillaryInfo(), aEvent.getQName(),
                     aEvent.getTimestamp());
```

Ce qui nous montre que notre listener doit avoir une méthode processEventHandler, ce qui n'était pas notre cas. Nous avions déjà tester avec des listener avec cette implémentation nous en avons donc refait un, implémentant l'interface `IAeEventHandler`, nous nous sommes aussi aperçus que le nom de nos Web Service servant de listener devait correspondre à un certain format, comme on peut le voir dans le constructeur de la classe **AeEventHandlerLocator** ainsi que dans l'interface **IaeEventHandlerConstants** d'ActiveBpel**,** si le nom se termine par `ActiveBpelEventHandler` les informations ne sont pas transmise par RPC apparemment (nous n'avons pas étudié d'avantage celà, nous avons simplement observer qu'aucune notification n'était faites lorsque le listener se terminé avec un nom de la sorte), le nom aura certainement d'avantage d'importance dans les prochaines version d'ActiveBpel comme le laisse entendre le commentaire laissé par les concepteur du logiciel précédé d'un TODO :

```
public interface IAeEventHandlerConstants
{
   /** Target namespace for engine administration web service */
   public final static String EVENT_HANDLER_NS = "http://docs.active-
endpoints/wsdl/eventhandler/2007/01/eventhandler.wsdl"; //$NON-NLS-1$
   /** Target namespace for engine administration web service */
```

```
   public final static String EVENT_HANDLER_SERVICE = "ActiveBpelEventHandler";
//$NON-NLS-1$

   // TODO (RN) These constants will go away in next release
   /** Target namespace for RPC style engine administration web service. */
   public final static String RPC_EVENT_HANDLER_NS =
"urn:AeRemoteDebugServices"; //$NON-NLS-1$
   /** Service name for RPC style engine administration service */
   public final static String RPC_EVENT_HANDLER_SERVICE = "BpelEventHandler"; //
$NON-NLS-1$
}
```

Nous avons donc fait un Web Service avec un nom quelconque (le service est par défaut *RPC_EVENT_HANDLER_SERVICE* si l'on ne met pas un nom particulier) et nous avons recommencer l'expérience et nous avons enfin été notifié. Le test était d'observer l'orchestration Loan Approvement qui est présente dans les tutoriaux d'ActiveBpel et nous avons reçus 27 notifications pour les processus grace aux méthodes :

```
processEventHandler(long aContextId, long aProcessId, java.lang.String aPath,
int aEventType, java.lang.String aFaultName, java.lang.String aText,
javax.xml.namespace.QName aName, java.util.Date aTimestamp)

processInfoEventHandler(long aContextId, long aProcessId, java.lang.String
aPath, int aEventType, java.lang.String aFaultName, java.lang.String aText,
java.util.Date aTimestamp)
```

Nous avons seulement tester pour des processus mais il est aussi possible de faire celà pour voir l'état du moteur d'orchestration ainsi que pour des breakpoint.

Il nous reste encore a tester cela avec des composants fractal, car pour l'instant nous n'avons qu'un simple objet java pour l'implémentation du listener et nous imprimons les message reçus dans les logs de tomcat, mais normalement il ne devrait pas y avoir de probléme particulier. Nous n'avons pas encore étudier les classes qui ont été ajouté dans le jar ae_wsio.jar mais cela représente 2524 fichiers, le plus interessant est probablement la package work qui contient 20 classes ayant pour nom entre autre IaeProcessWorkManager.class, AeWorkEvent.class ,AeWorkerThread.class ... mais nous n'avons pas d'avantages regarder.