Structure du processus axb.bpel (avec extension asynchrone):

client (soap-ui ou ...)
envoie requête synchrone process_axb(a=2,x=3,b=4,corrId=1)

-----> ***receive*** by axb/**process_axb**

----------> invoke ***multiplication***(2,3) sur
WS calculateur

----------> invoke ***addition***(6,4) sur
WS calculateur

assign $output (result) var

puis en //

<---------- ----------> invoque async/OneWay ***testCorrelation*** sur WS "*MyRemoteLogger*" (début correlation sur numero)

reply (10) vers appelant synchrone (ex: soap-ui)

.... (attente opération longue).....

<---------- attente de la réponse asynchrone via receive d'une "***callback***" sur WS "*MyRemoteCallback*" (fin correlation sur numero)

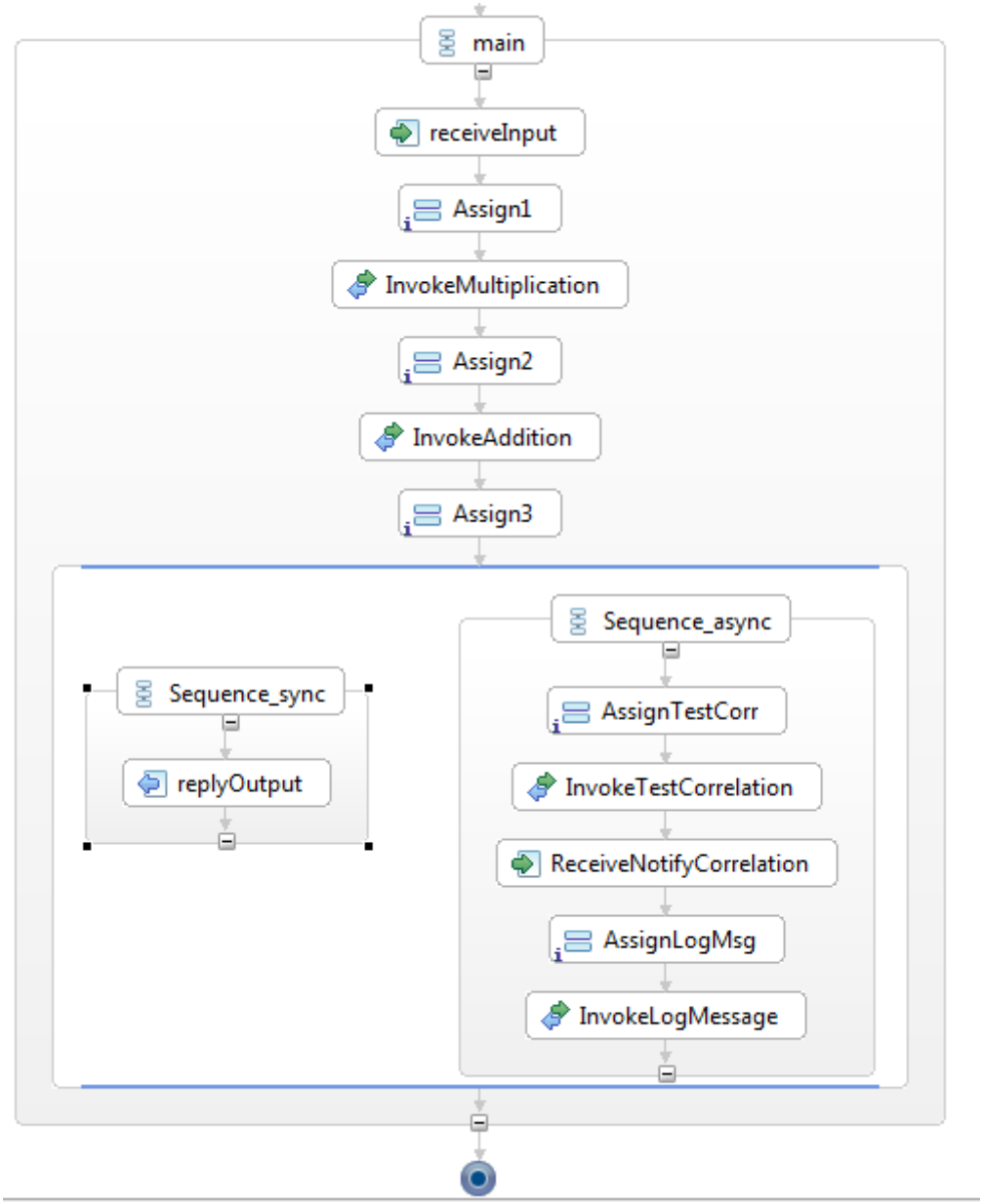
-------> envoie en OneWay vers ***logMessage*** du service "*MyRemoteLogger*" ou ... (suivant le même principe on peut déclencher l'envoie d'un mail)

---

et du coté (java/cxf) / "MyRemoteLogger"

----------> reception async/OneWay "***testCorrelation***"
affichage message console
+ démarrer Thread ----------> pause 5 seconde (simulation traitement long)
+ rendre la main (fin)
....
<---------- appel de "**callback**" sur service "*MyRemoteCallback*" en repassant le numero (info de correlation,)

Processus "BPEL" (vue eclipse/Bpel Designer):



Remarque:

Comme le prévoie la norme BPEL , les *portTypes wsdl* "*MyRemoteLogger*" et "*MyRemoteCallback*" sont placés dans un ***même "partnerLink"*** avec *partnerRole*="role du service externe invoqué" et *myRole*="role du process gérant la callback" .

```
<bpel:partnerLink name="myRemoteAsyncParnter"
        partnerLinkType="tns:myRemotePartnerLinkType"
        partnerRole="myRemoteLogger"
        myRole="myRemoteCallback"     />
```

Pour des raisons syntaxiques (namespaces xml), le partnerLinkType associé à été placé dans le fichier asyncPartner.wsdl .

asyncPartner.wsdl

```
<definitions name="asyncPartner"   targetNamespace="http://tp"       xmlns:tns="http://tp"
      xmlns:nsc="http://service.tp/"       xmlns:nscb="http://callback.tp/"
      xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  >

  <import location="myRemoteLogger.wsdl" namespace="http://service.tp/" />
  <import location="myRemoteCallback.wsdl" namespace="http://callback.tp/" />

  <plnk:partnerLinkType name="myRemotePartnerLinkType">
      <plnk:role name="myRemoteCallback" portType="nscb:MyRemoteCallback"/>
      <plnk:role name="myRemoteLogger" portType="nsc:MyRemoteLogger"/>
   </plnk:partnerLinkType>
</definitions>
```

le fichier ***abstractProperties.wsdl*** comporte la définition d'un attribut de corrélation qui sera utilisé par la partie "correlationSet" du processus bpel:

```
<definitions name="properties"
      targetNamespace="http://tp"    xmlns:tns="http://tp"
      xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"       >

  <vprop:property name="number" type="xsd:int"/>

</definitions>
```

Cette proprieté de correlation *number* est (par configuration) associée à

- l' élément "***numero***" du message de requete "*testCorrelation*" de *myRemoteLogger.wsdl*
- l' élément *"**numero**"* du message de requete *"notifyCorrelation"* de *myRemoteCallback.wsdl*
- une partie du ***correlationSet*** du processus **bpel**

dans *myRemoteCallback.wsdl*

```
<vprop:propertyAlias propertyName="corr:number" xmlns:corr="http://tp"
                     messageType="tns:notifyCorrelation" part="parameters">
                     <vprop:query>numero</vprop:query>
</vprop:propertyAlias>
```

dans myRemoteLogger.wsdl:

```
<vprop:propertyAlias propertyName="corr:number"   xmlns:corr="http://tp"
                     messageType="tns:testCorrelation" part="parameters">
                     <vprop:query>numero</vprop:query>
</vprop:propertyAlias>
```

dans axb.bpel

```
<bpel:correlationSets xmlns:cor="http://tp">
        <bpel:correlationSet   name="numberCorrSet"    properties="cor:number" />
 </bpel:correlationSets>
```

Schéma particulier d'inclusion (ici) :

axb.bpel ---> axbArtifacts.wsdl
----->calculateur.wsdl
---> asyncPartner.wsdl -----> myRemoteCallback.wsdl ---->
-----> myRemoteLogger.wsdl   ----> abstractProperties.wsdl
---------->

(d'autres combinaisons/configurations d'inclusions) sont envisageables .

Détails sur la partie "correlation asynchrone (bpel)":

```
 <bpel:invoke name="InvokeTestCorrelation" partnerLink="myRemoteAsyncParnter"
operation="testCorrelation" portType="nsc:MyRemoteLogger"
inputVariable="testCorrelationInput">
              <bpel:correlations>
                 <bpel:correlation initiate="yes" set="numberCorrSet"></bpel:correlation>
              </bpel:correlations>
 </bpel:invoke>
```

```
<bpel:receive name="ReceiveNotifyCorrelation" partnerLink="myRemoteAsyncParnter"
operation="notifyCorrelation" portType="nscb:MyRemoteCallback"
variable="notifyCorrelationInput">
              <bpel:correlations>
                 <bpel:correlation initiate="no" set="numberCorrSet"></bpel:correlation>
              </bpel:correlations>
</bpel:receive>
```

Structure globale du process (traitements en parallèle)

```
<bpel:sequence name="main">
   receive , assign , invoque(s) synchrone(s) , assign
   <bpel:flow  name="Flow_en_parallele>
       <bpel:sequence  name="Sequence_sync">
              reply immediat
       </bpel:sequence>
       <bpel:sequence  name="Sequence_Async">
             séquence asynchrone potentiellement longue
             (attente)
       </bpel:sequence>
   </bpel:flow>
</bpel:sequence>
```

--> fonctionne bien sur courte durée --> à tester (et si besoin adapter) sur très longue duréee .

Le lien entre le processus BPEL et les services externes asynchrones peut se faire de 2 façons:

- via une prise en charge directe de ODE au sein d'un ESB compatible (ex: serviceMix)
- via des communications http/"oneWay" vers des services web soap-over-http (qui peuvent à leurs tours retransmettre les messages via d'autres technologies (JMS , mail SMTP , ...)).

De façon à ne pas imposer l'ESB "servicemix" (assez complexe) , la suite de ce document montre comment le processus bpel peut directement communiquer (en mode "onWay") avec des services web externes SOAP-over-http .

URL des différents services tournant autour du processus BPEL:

http://localhost:8080/ode/processes/axb/process_axb	URL (interne) du processus bpel . Définie dans axbArtifacts.wsdl
http://localhost:8080/ode/processes/axb/myRemoteCallback	URL (interne) du service de callback du bpel . Définie dans myRemoteCallback.wsdl
http://localhost:8080/wsCalculateur/services/myRemoteLogger	URL (externe) du service "myRemoteLogger" . Définie dans myRemoteLogger.wsdl
http://localhost:8080/wsCalculateur/services/calculateur	URL (externe) du service synchrone "Calculateur" . Définie dans calculateur.wsdl

Code du service asynchrone "MyRemoteLogger" (en java/cxf):

```
package tp.service;

import javax.jws.Oneway;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService
public interface MyRemoteLogger {
	@Oneway
	public void logMessage(@WebParam(name="msg")String msg);

	@Oneway
	public void testCorrelation(@WebParam(name="numero")int num,
				@WebParam(name="requestMsg")String requestMsg);
}
```

vision java de l'interface de callback (qui sera appelée sur le bpel):

```
@WebService
public interface MyRemoteCallback {
	@Oneway
	public void notifyCorrelation(@WebParam(name="numero")int num,
				@WebParam(name="notifMsg")String notifMsg);
}
```

Code java du service "MyRemoteLogger"

```
@WebService(endpointInterface="tp.service.MyRemoteLogger")
public class MyRemoteLoggerImpl implements MyRemoteLogger{
	public void logMessage(String msg) {
		System.out.println("logged message:" + msg);
	}
	public void testCorrelation(int num, String requestMsg) {
		System.out.println("testCorrelation [num=" + num + "," + requestMsg + "]");
		//plus appel d'une callback via nouveau thread en differe (+5s)
		Thread t = new Thread(new CodeThreadAppelantCallBack(num, requestMsg));
		t.start();
	}
}
```

NB: Le thread employé ici ne sert qu'à simuler un traitement long tout en rendant la main aussitôt. Dans le cas d'une véritable appli , un message est stocké quelquepart (mémoire , base de données) et c'est lorsque un utilisateur traite le message via l'IHM , qu'il déclenche alors ensuite la "callback".

Code java du Thread appelant la callback:

```
package tp.callback;

import javax.xml.namespace.QName;	import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class CodeThreadAppelantCallBack implements Runnable{
	private int numero;	private String requestMsg;

	public CodeThreadAppelantCallBack(int numero,String requestMsg){
		this.numero= numero;		this.requestMsg = requestMsg;
	}

public void run() {
	System.out.println("5s de pause");
		try {
			Thread.sleep(5 * 1000); //5000 ms = 5s
		    } catch (Exception e) { e.printStackTrace();	}
	System.out.println("appel de la callback");

	QName SERVICE_NAME = new QName("http://callback.tp/", "MyRemoteCallbackImplService");
	QName PORT_NAME = new QName("http://callback.tp/", "MyRemoteCallbackImplPort");
	Service service = Service.create(SERVICE_NAME);
	String endpointAddress = "http://localhost:8080/ode/processes/axb/myRemoteCallback";
	service.addPort(PORT_NAME,
			SOAPBinding.SOAP11HTTP_BINDING,endpointAddress);
	MyRemoteCallback callbackProxy = (MyRemoteCallback)
				service.getPort(PORT_NAME, MyRemoteCallback.class);

	callbackProxy.notifyCorrelation(this.numero,
					"notification: message recu=" + this.requestMsg);
	}
}
```