

Guide utilisateur

Cimero2

BULL R&D

Date : 29 août 2007
Version : 2.0
Auteur : Bagnard Natacha
Forot Julien



Version	Description	Ecrit par	Revu par	Date
1.0	Corps du document	Nosedo Anne (BULL)		24/08/2007
2.0	Mise à jour en fonction de la dernière version de Cimero	Nosedo Anne (BULL)		29/08/2007
2.1	Mise à jour	Forot Julien Natacha Bagnard		30/08/2007



Table des matières

<u>1</u>	<u>Introduction.....</u>	<u>4</u>
<u>2</u>	<u>Utilisation.....</u>	<u>5</u>
2.1	<i>Création du projet.....</i>	<i>5</i>
2.2	<i>La barre d'outil Eclipse.....</i>	<i>7</i>
2.3	<i>La palette.....</i>	<i>7</i>
2.4	<i>Les namespaces.....</i>	<i>8</i>
2.5	<i>Les propriétés.....</i>	<i>9</i>
2.5.1	BC HTTP.....	10
2.5.2	BC JMS.....	13
2.5.3	SE EIP Pipeline.....	17
2.5.4	SE BEAN.....	19
2.5.5	SE JSR181.....	23
2.6	<i>Validation.....</i>	<i>25</i>
2.7	<i>Génération JBI.....</i>	<i>26</i>
2.8	<i>Ajout d'une SU.....</i>	<i>28</i>
2.9	<i>Regénération du SA.....</i>	<i>30</i>



1 Introduction

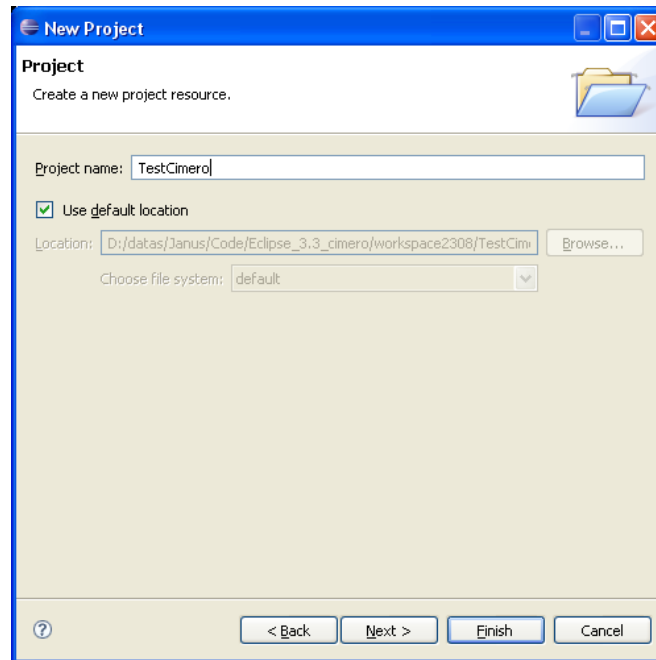
Cimero est un outil de développement graphique intégré à Eclipse. Il permet de dessiner un processus JBI rapidement en faisant des drag-and-drop de composants. Cimero génère ensuite les SU et les SA.

Vous trouverez des informations sur Cimero2 à l'adresse suivante :
http://wiki.eclipse.org/Cimero_2.

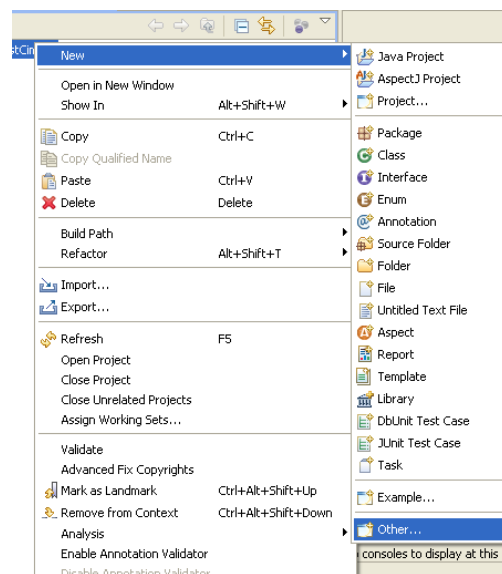
2 Utilisation

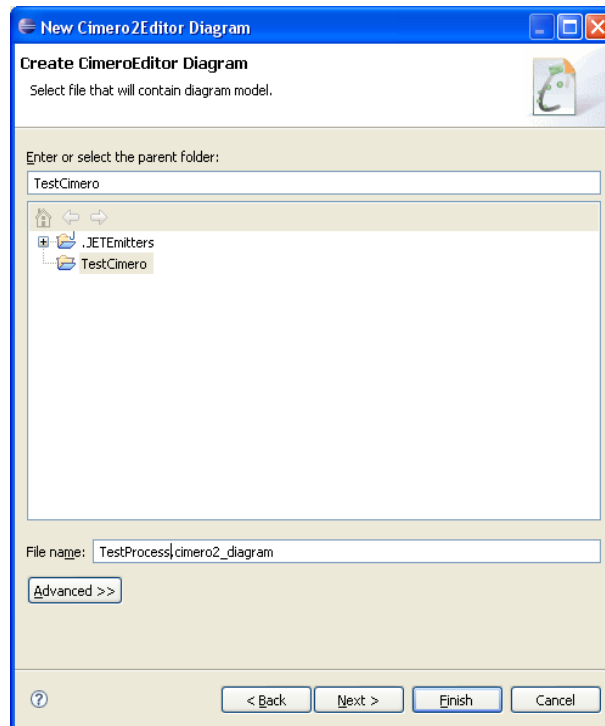
2.1 Création du projet

Démarrez Eclipse Europa (pour l'installation de Cimero, voir Guide d'Installation).
Créez un nouveau projet TestCimero.

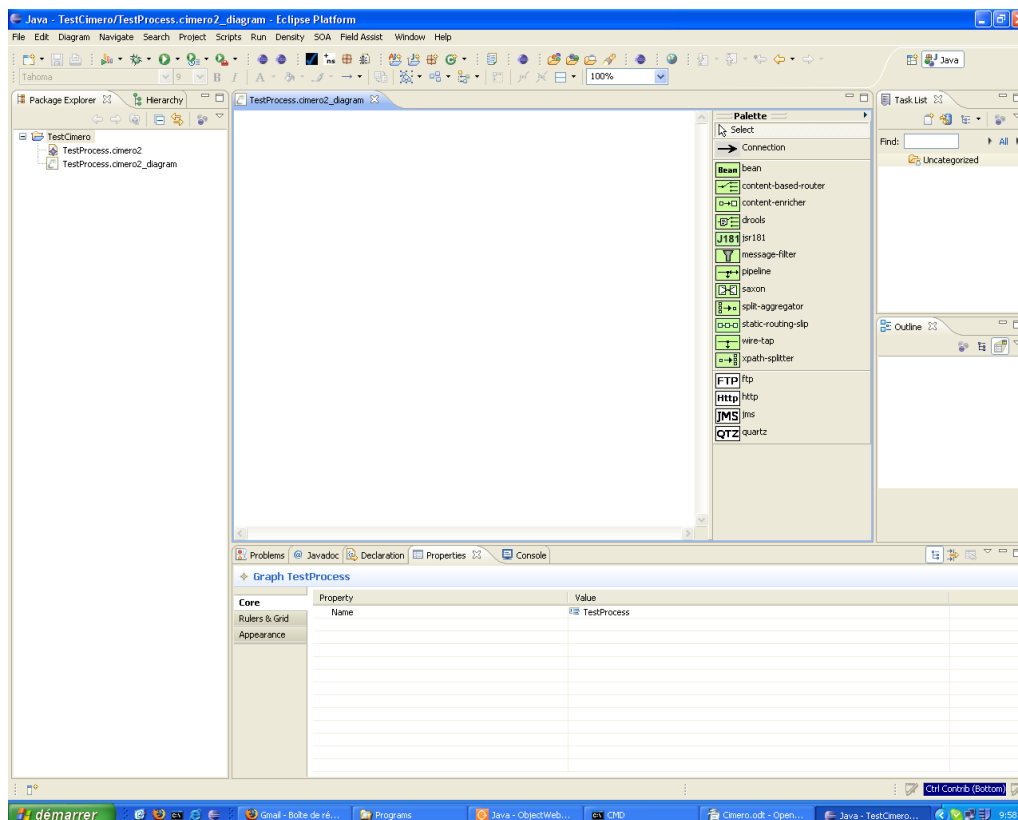


Créez un diagramme Cimero en cliquant New --> Other. Choisissez Cimero --> Cimero diagram, puis cliquez Next et donnez lui un nom (par exemple TestProcces). Choisissez l'ESB Servicemix et cliquez sur Finish.



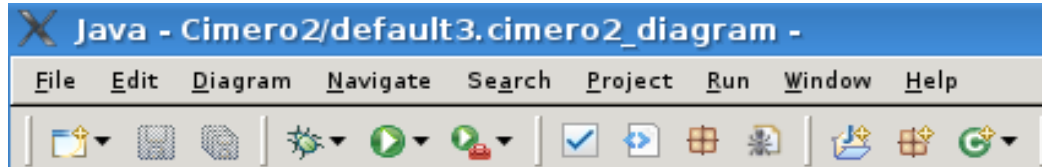


Vous devriez obtenir un fichier blanc vous permettant de dessiner votre processus à l'aide de la palette se trouvant à droite de l'écran.







2.2 La barre d'outil Eclipse

Lorsque le Plugin Cimero2 est installé, les fonctionnalités d'ajout de namespace, de validation, de génération de package ou de tâche Ant décrites plus loin sont accessibles via la barre d'outil d'Eclipse.



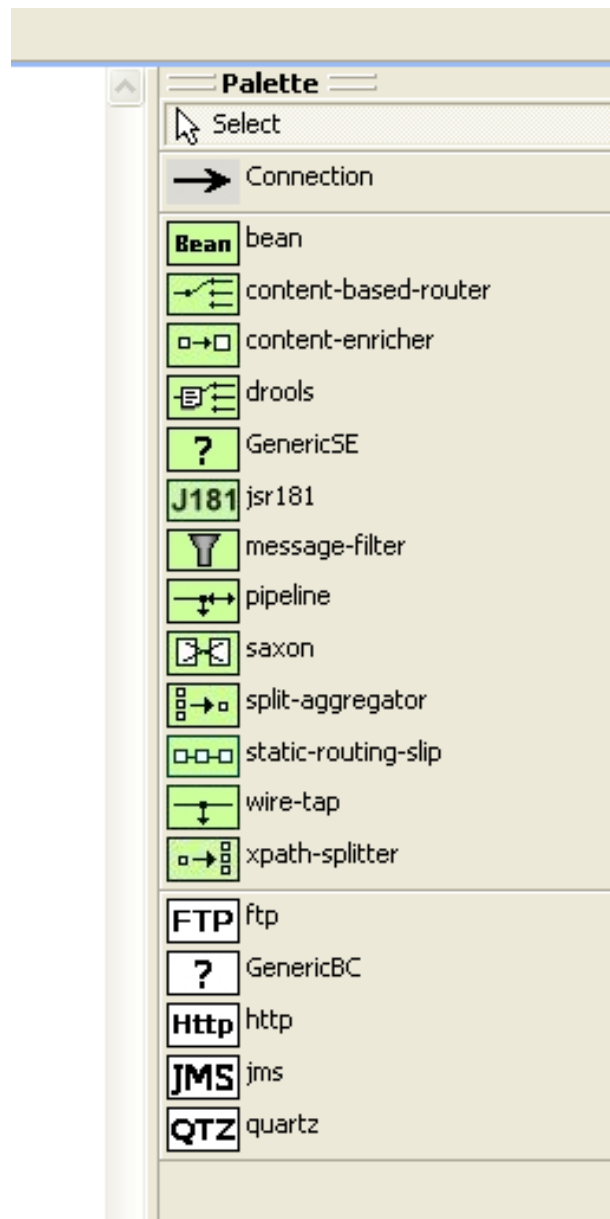
Voici la liste des icônes et des fonctions associées :

- x  Validation du graphe
- x  Ajout de namespace
- x  Génération de package JBI
- x  Génération de tâche Ant

2.3 La palette

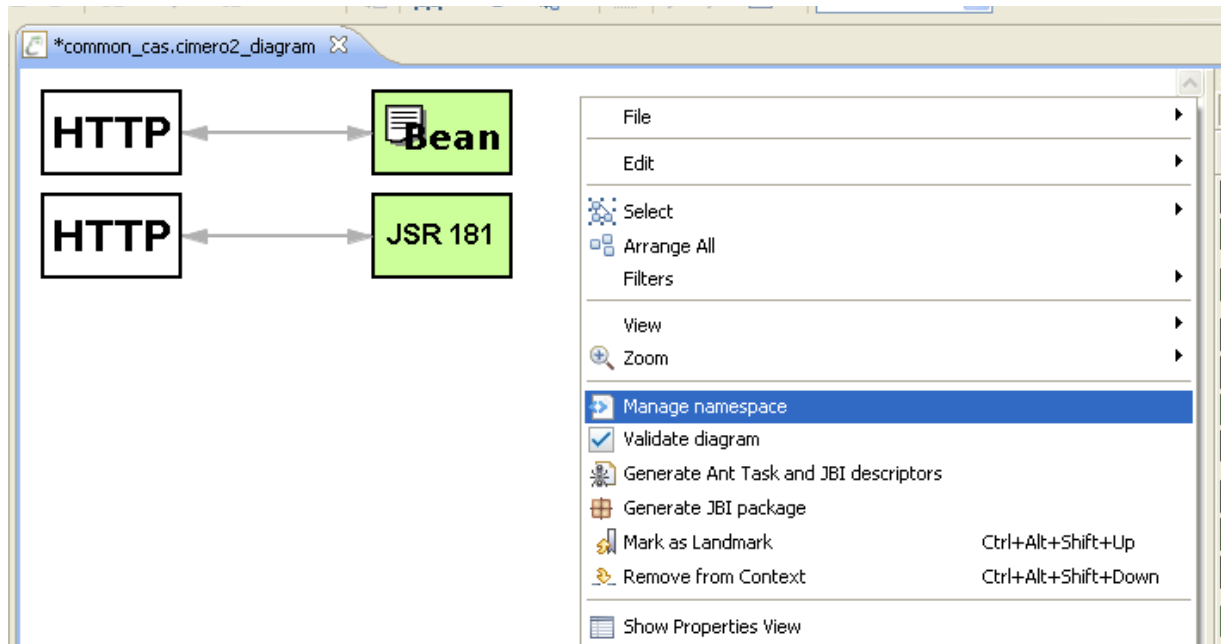
La palette comporte :

- x un outil de sélection
- x un outil de connection
- x des services engines en vert
- x des bindings components en blanc



2.4 Les namespaces

Il est possible de définir des namespaces que vous utiliserez par la suite dans votre diagramme dans les différents composants. Cliquez droit dans votre diagramme et sélectionnez « Manage namespace » ou bien utilisez la barre d'outil Eclipse, une boîte de dialogue apparaît. Cliquez sur le bouton « New Namespace » et introduisez le préfixe et la valeur de votre namespace.



Namespace Repository

Namespace Repository

Prefix	Value
janus	http://www.etnic.be/janus
secu	http://www.etnic.be/janus/securityEnricher

Buttons: Delete Namespace, Edit Namespace, New Namespace, Close

New Namespace

New Namespace

Prefix: janus

Value: http://www.etnic.be/janus













Buttons: OK, Cancel

2.5 Les propriétés




Chaque composant possède une foule de propriétés qui sont configurables dans la vue « Properties » d'Eclipse. Il vous suffit de sélectionner le composant souhaité pour qu'elles apparaissent.

Chaque propriété est symbolisée par un icône.

Cet icône représente à la fois le type de contenu :

- x  Generic
- x  String
- x  Integer
- x  URL
- x  URI
- x  Boolean
- x  QName
- x  Class
- x  File
- x  Role
- x  MEP
- x  Namespace

et comporte un code couleur :

- x  s'il est rouge, la propriété est obligatoire et elle n'est pas remplie actuellement
- x  s'il est bleu, la propriété est facultative et n'est pas remplie actuellement
- x  s'il est vert, la propriété est remplie actuellement

Il est facile d'identifier si un composant est bien configuré à l'aide des points d'exclamations rouges présents dans l'arborescence des propriétés et indiquant qu'une propriété obligatoire n'est pas correctement remplie.

2.5.1 BC HTTP

Problems Javadoc Declaration Properties Console

Component Instance http0

Core	Property	Value
Appearance	Component	
	01- name	
	02- endpoint	!
	01- endpoint	!
	02- targetService	!
	03- targetEndpoint	!
	04- targetInterfaceName	!
	05- role	!
	06- defaultMep	!
	07- soap	!
	08- soapVersion	!
	09- wsdlResource	!
	10- locationURI	!
	11- soapAction	!
	12- interfaceName	
	01- namespace	!
	02- value	!
	13- service	!
	01- namespace	!
	02- value	!
	14- defaultOperation	
	01- namespace	!
	02- value	!
	15- ssl	
	01- keyPassword	!
	02- keyStore	URL
	03- keyStorePassword	!
	04- keyStoreType	!
	05- trustStore	URL
	06- trustStorePassword	!
	07- trustStoreType	!
	08- protocol	!
	09- algorithm	!
	10- wantClientAuth	!
	11- needClientAuth	!
	16- basicAuthentication	
	01- username	!
	02- password	!
	17- policies	
	01- ws-security	
	01- receiveAction	!
	02- sendAction	!
	03- username	!
	04- keystore	!
	05- crypto	
	01- bean	
	01- class	
01- location	!	
02- class value	!	
02- property	Click to add a new property property	

Les propriétés obligatoires sont :

- x le nom de service
- x le nom de endpoint
- x l'URI auquel le service sera accessible
- x le rôle du composants (provider ou consumer)

Voici un exemple de configuration correcte :

Component Instance httpInFaseAsync		
Properties	Property	Value
Appearance	Component	
	01- name	httpInFaseAsync
	02- endpoint	
	01- endpoint	HttpFaseAsyncEndpoint
	02- targetService	fase:async
	03- targetEndpoint	endpoint
	04- targetInterfaceName	
	05- role	Consumer
	06- defaultMep	InOut
	07- soap	true
	08- soapVersion	1.2
	09- wsdlResource	D:\datas\Janus\Code\Eclipse_Europa_Cimero_new\workspace2908\fase_async_
	10- locationURI	http://0.0.0.0:9080/janus/fase/async/
	11- soapAction	
	12- interfaceName	
	13- service	
	01- namespace	fase
	02- value	HttpFaseAsyncService
	14- defaultOperation	
	15- ssl	
	16- basicAuthentication	
	17- policies	
	01- ws-security	
	01- receiveAction	Signature Timestamp
	02- username	smx
	03- keystore	default
	04- crypto	
	01- bean	
	01- class	
	01- location	D:\datas\Janus\Code\Eclipse_Europa_Cimero_new\workspace2908\fase_async_
	02- class value	be.etnic.janus.cas.security.EtnicLDAPCrypto
	02- property	Click to add a new property property
	01- name	ldapServer
	02- Exclusive properties	
	01- value	ldap://setnic40:389/
	02- ref	
	03- property	Click to delete this property
	01- name	rootContext
	02- Exclusive properties	
	01- value	dc=cfwb,dc=be
	02- ref	
	04- property	Click to delete this property
	01- name	usersContext
	02- Exclusive properties	
	01- value	ou=People
	02- ref	

Voici le fichier xbean.xml que Cimero2 génère :

```
<?xml version="1.0"?>
<!--
#####
Generated by CimeroEditor V2
Bull S. A. S., Rue Jean Jaures, B.P.68, 78340, Les Clayes-sous-Bois
#####
-->

<beans xmlns:http="http://servicemix.apache.org/http/1.0"
xmlns:soap="http://servicemix.apache.org/soap/1.0"
xmlns:janus="http://www.etnic.be/janus"
xmlns:fase="http://www.etnic.be/janus/fase">
```

```
<classpath>
  <location>./</location>
</classpath>

<http:endpoint
  service="fase:HttpFaseAsyncService"
  endpoint="HttpFaseAsyncEndpoint"
  targetService="fase:async"
  targetEndpoint="endpoint"
  role="consumer"
  defaultMep="http://www.w3.org/2004/08/wsdl/in-out"
  soap="true"
  soapVersion="1.2"
  wsdlResource="classpath:fase.wsdl"
  locationURI="http://0.0.0.0:9080/janus/fase/async/">

<http:policies>
  <soap:ws-security
    receiveAction="Signature Timestamp"
    username="smx"
    keystore="default">
    <soap:crypto>
      <bean class="be.ethnic.janus.cas.security.EtnicLDAPCrypto">
        <property name="ldapServer" value="ldap://setnic40:389/">
        <property1 name="rootContext" value="dc=cfwb,dc=be"/>
        <property2 name="usersContext" value="ou=People"/>
      </bean>
    </soap:crypto>
  </soap:ws-security>
</http:policies>

</http:endpoint>

</beans>
```

2.5.2 BC JMS

Il est possible de configurer 3 types d'endpoints :

- x jms:endpoint : configuration présente déjà dans la version servicemix 3.1
- x jms:consumer : nouvelle fonctionnalité de la version servicemix 3.2
- x jms:provider : nouvelle fonctionnalité de la version servicemix 3.2

Voici un exemple de configuration correcte pour un « jms:consumer » :

◆ Component Instance jmsInWorkFase		
Properties	Property	Value
Appearance	Component	
	01- name	jmsInWorkFase
	02- Exclusive properties	⚙️
	01- endpoint	
	02- consumer	
	01- endpoint	🔗 JmsOutFaseEndpoint
	02- targetService	🔗 faseAsync:pipeline
	03- targetEndpoint	🔗 pipelineFaseEndpoint
	04- targetInterface	🔗
	05- defaultMEP	🔗
	06- soap	🔗
	07- soapVersion	🔗
	08- wsdlResource	🔗
	09- initialContextFactory	🔗
	10- jndiProviderURL	🔗
	11- destination	🔗
	12- destinationStyle	🔗
	13- useMsgIdInResponse	🔗
	14- synchronous	🔗 true
	15- destinationChooser	🔗
	16- destinationResolver	🔗
	17- pubSubDomaom	🔗
	18- useMessageIdInResponse	🔗
	19- replyDestination	🔗
	20- replyDestinationName	🔗
	21- replyExplicitQosEnabled	🔗
	22- replyDeliveryMode	🔗
	23- replyPriority	🔗
	24- replyTimeToLive	🔗
	25- replyProperties	🔗
	26- stateless	🔗
	27- storeFactory	🔗
	28- store	🔗
	29- listenerType	🔗 default
	30- jms102	🔗
	31- transacted	🔗 xa
	32- clientId	🔗
	33- destinationName	🔗 queue/FASE
	34- durableSubscriptionName	🔗
	35- exceptionListener	🔗
	36- messageSelector	🔗
	37- sessionAcknowledgeMode	🔗
	38- subscriptionDurable	🔗
	39- pubSubNoLocal	🔗
	40- concurrentConsumers	🔗 32
	41- cacheLevel	🔗 3
	42- receiveTimeout	🔗
	43- recoveryInterval	🔗
	44- maxMessagesPerTask	🔗
	45- serverSessionFactory	🔗
	46- interfaceName	
	47- service	
	01- namespace	🔗 faseAsync
	02- value	🔗 JmsOut
	48- defaultOperation	
	49- Exclusive properties	⚙️
	01- connectionFactory	🔗 #connectionFactory
	02- jndiConnectionFactoryName	🔗
	50- marshaler	



Voici le fichier xbean.xml que Cimero génère :

```
<?xml version="1.0"?>
<!--
#####
Generated by CimeroEditor V2
Bull S. A. S., Rue Jean Jaures, B.P.68, 78340, Les Clayes-sous-Bois
#####
-->

<beans xmlns:jms="http://servicemix.apache.org/jms/1.0"
      xmlns:faseAsync="http://www.etnic.be/janus/fase/async"
      xmlns:janus="http://www.etnic.be/janus">

    <jms:consumer service="faseAsync:JmsOut"
        endpoint="JmsOutFaseEndpoint" targetService="faseAsync:pipeline"
        targetEndpoint="pipelineFaseEndpoint"
        connectionFactory="#connectionFactory" synchronous="true"
        listenerType="default" transacted="xa" destinationName="queue/FASE"
        concurrentConsumers="32" cacheLevel="3" />

</beans>
```

Il faut ensuite ajouter manuellement ce qui concerne la connectionFactory JMS (référéncée par #connectionFactory).

Voici un exemple de configuration correcte pour un « `.jms:provider` » :

Component Instance <code>jmsOutCasFaseAsync</code>		
Properties	Property	Value
Appearance	Component	
	01- name	<code>jmsOutCasFaseAsync</code>
	02- Exclusive properties	
	01- endpoint	
	02- provider	
	01- endpoint	<code>JmsInFaseEndpoint</code>
	02- destination	
	03- connectionFactory	<code>#connectionFactory</code>
	04- destinationName	<code>queue/FASE</code>
	05- jms102	
	06- pubSubDomain	
	07- messageIdEnabled	
	08- messageTimestampEnabled	
	09- pubSubNoLocal	
	10- receiveTimeout	
	11- explicitQosEnabled	
	12- deliveryMode	
	13- priority	
	14- timeToLive	
	15- replyDestinationName	
	16- replyDestination	
	17- stateless	
	18- storeFactory	
	19- store	
	20- interfaceName	
	21- service	
	01- namespace	<code>janus</code>
	02- value	<code>JmsInFase</code>
	22- marshaler	

Voici le fichier `xbean.xml` que Cimero génère :

```
<?xml version="1.0"?>
<!--
#####
Generated by CimeroEditor V2
Bull S. A. S., Rue Jean Jaures, B.P.68, 78340, Les Clayes-sous-Bois
#####
-->

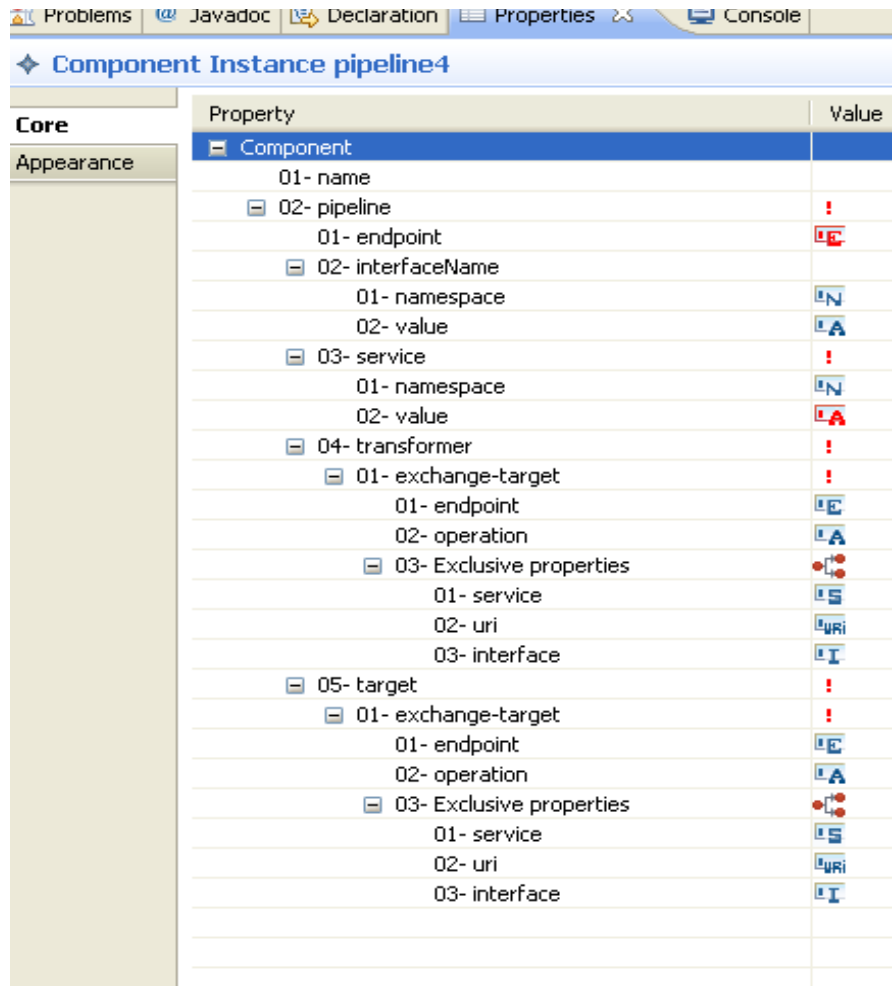
<beans xmlns:jms="http://servicemix.apache.org/jms/1.0"
xmlns:janus="http://www.etnic.be/janus"
xmlns:fase="http://www.etnic.be/janus/fase">

<jms:provider
service="janus:JmsInFase"
endpoint="JmsInFaseEndpoint"
connectionFactory="#connectionFactory"
destinationName="queue/FASE"/>

</beans>
```


Il faut ensuite ajouter manuellement ce qui concerne la connectionFactory JMS (référéncée par #connectionFactory).

2.5.3 SE EIP Pipeline



Property	Value
Component	
01- name	
02- pipeline	!
01- endpoint	!
02- interfaceName	
01- namespace	EN
02- value	EA
03- service	!
01- namespace	EN
02- value	EA
04- transformer	!
01- exchange-target	!
01- endpoint	EC
02- operation	EA
03- Exclusive properties	!
01- service	ES
02- uri	URI
03- interface	IT
05- target	!
01- exchange-target	!
01- endpoint	EC
02- operation	EA
03- Exclusive properties	!
01- service	ES
02- uri	URI
03- interface	IT

Les propriétés obligatoires sont :

- x le nom de l'endpoint
- x le nom de service
- x le nom de service du « transformer component »
- x le nom de service du « target component »

Voici un exemple de configuration correcte :

Component Instance pipeline		
Core	Property	Value
Appearance	Component	
	01- name	pipeline
	02- pipeline	
	01- endpoint	pipelineFaseEndpoint
	02- interfaceName	
	01- namespace	
	02- value	
	03- service	
	01- namespace	faseAsync
	02- value	pipeline
	04- transformer	
	01- exchange-target	
	01- endpoint	Interface_To_FaseAsync_Http
	02- operation	janus:janusRequest
	03- Exclusive properties	
	01- service	janus:Interface_To_FaseAsync_Service
	02- uri	
	03- interface	janus:Interface_To_FaseAsync_PortType
	05- target	
	01- exchange-target	
	01- endpoint	
	02- operation	
	03- Exclusive properties	
	01- service	janus:lazy
	02- uri	
	03- interface	

Voici le fichier xbean.xml que Cimero génère :

```
<?xml version="1.0"?>
<!--
#####
Generated by CimeroEditor V2
Bull S. A. S., Rue Jean Jaures, B.P.68, 78340, Les Clayes-sous-Bois
#####
-->

<beans xmlns:pipeline="http://servicemix.apache.org/eip/1.0"
xmlns:foo="http://foonamespace/"
xmlns:cimero2="http://cimero2.bull.net"
xmlns:faseAsync="http://www.etnic.be/janus/fase/async"
xmlns:janus="http://www.etnic.be/janus">

<pipeline:pipeline
service="faseAsync:pipeline"
endpoint="pipelineFaseEndpoint">
<pipeline:transformer>
<pipeline:exchange-target
interface="janus:Interface_To_FaseAsync_PortType"
service="janus:Interface_To_FaseAsync_Service"
endpoint="Interface_To_FaseAsync_Http"
operation="janus:janusRequest"/>
</pipeline:transformer>
```

```
<pipeline:target>
  <pipeline:exchange-target
    service="janus:lazy"/>
  </pipeline:target>
</pipeline:pipeline>

</beans>
```

2.5.4 SE BEAN

Component Instance bean3		
Core	Property	Value
Appearance	[-] Component	
	01- name	
	[-] 02- endpoint	!
	01- endpoint	!
	02- bean	!
	[-] 03- interfaceName	
	01- namespace	!
	02- value	!
	[-] 04- service	!
	01- namespace	!
	02- value	!
	[-] 05- bean	!
	01- id	!
	[-] 02- class	!
	01- location	!
	02- class value	!
	[-] 03- property	Click to add a new property property
	01- name	!
	[-] 02- Exclusive properties	!
	01- value	!
	02- ref	!

Les propriétés obligatoires sont :

- x le nom de l'endpoint
- x le nom de service
- x le référencement à un bean
- x l'id du bean
- x le nom de la classe d'implémentation du bean
- x l'emplacement du répertoire contenant le package et le fichier .class ou l'emplacement du fichier .jar contenant le package et la classe.

Exemple de configuration correcte d'un composant BEAN :

Problems @ Javadoc Declaration Properties Console		
Component Instance beanLazy		
Core	Property	Value
Appearance	Component	beanLazy
	01- name	beanLazy
	02- endpoint	
	01- endpoint	lazyEndpoint
	02- Add a new bean	
	03- bean	#theDuke
	04- interfaceName	
	01- namespace	
	02- value	
	05- service	
	01- namespace	janus
	02- value	lazy
	03- bean	Click to add a new bean property
	01- id	theDuke
	02- class	
	01- location	D:\datas\Janus\Code\Eclipse_3.3_cimero\workspace2808\common_work_new\lib\janus-lazy.jar
	02- class value	be.etnic.janus.lazy.Lazy
	03- property	Click to add a new property property
	01- Add a new name	
	02- name	
	03- Exclusive properties	
	01- value	
	02- ref	

Voici le fichier généré par Cimero :

```
<?xml version="1.0"?>
<!--
#####
Generated by CimeroEditor V2
Bull S. A. S., Rue Jean Jaures, B.P.68, 78340, Les Clayes-sous-Bois
#####
-->
<beans xmlns:bean="http://servicemix.apache.org/bean/1.0"
xmlns:foo="http://foonamespace/"
xmlns:cimero2="http://cimero2.bull.net"
xmlns:janus="http://www.etnic.be/janus"
xmlns:fh="http://www.etnic.be/janus/faultHandler">

<bean:endpoint
service="janus:lazy"
endpoint="lazyEndpoint"
bean="#theDuke"/>

<bean
id="theDuke"
class="be.etnic.janus.lazy.Lazy"/>

</beans>
```

Remarque importante sur le classLoader

Dans la propriété « location » de la propriété « class » du composant BEAN, il vous est possible :

- x soit de donner la localisation d'un répertoire contenant votre package et votre fichier .class : sans doute la solution la plus simple si votre BEAN n'utilise qu'une classe Java ;
- x soit de donner la localisation d'un fichier .jar contenant tous les packages et toutes les classes que vous désirez. Seule solution si votre BEAN utilise plus d'une classe Java.

Si vous avez choisi la première option, le package contenant la classe sera copié à la racine de votre SU. Par contre, si vous avez choisi la deuxième option, votre fichier .jar sera copié dans un répertoire « lib » à la racine de votre SU.

Dans les 2 cas, la ou les classes seront chargées au démarrage de Servicemix.

Enfin, si jamais vous avez besoin de référencer au sein de votre fichier xbean.xml une ressource se trouvant à la racine de votre SU, il vous faut ajouter dans le fichier xbean.xml généré, les lignes suivantes :

```
<classpath>
  <location>./</location>
</classpath>
```

sous le tag <beans>.

Attention, si vous avez choisi d'importer un fichier .jar, il faut alors absolument que vous le référenciez également dans le classpath à l'aide d'un tag <location>.

Voici un exemple :

Component Instance beanValidation		
Property	Value	
Component		
01- name	beanValidation	
02- endpoint		
01- endpoint	validation	
02- bean	#valBean	
03- interfaceName		
01- namespace	N	
02- value	bean2	
04- service		
01- namespace	valid	
02- value	validationService	
05- bean		
01- id	valBean	
02- class		
01- location	D:\datas\Janus\Code\Eclipse_3.3_cimero\workspace2308\fase_common_work\janus-validation.jar	
02- class value	be.etnic.janus.validation.ValidationBean	
03- property	Click to add a new property property	
01- Add a new name		
02- name	xsd	
03- Exclusive properties		
01- value	classpath:fase.xsd	
02- ref		



```
<?xml version="1.0"?>
<!--
#####
Generated by CimeroEditor V2
Bull S. A. S., Rue Jean Jaures, B.P.68, 78340, Les Clayes-sous-Bois
#####
-->

<beans xmlns:bean="http://servicemix.apache.org/bean/1.0"
  xmlns:foo="http://foonamespace/"
  xmlns:cimero2="http://cimero2.bull.net"
  xmlns:valid="http://www.ethnic.be/janus/validation"
  xmlns:janus="http://www.ethnic.be/janus"
  xmlns:ethnic="http://services.fase.ethnic.be">

  <classpath>
    <location>./</location>
    <location>./lib/janus-validation.jar</location>
  </classpath>

  <bean:endpoint service="valid:validationService"
    endpoint="validation" bean="#valBean" />

  <bean id="valBean"
    class="be.ethnic.janus.validation.ValidationBean">
    <property name="xsd" value="classpath:fase.xsd" />
  </bean>

</beans>
```

TIPS : N'oubliez pas également de copier votre fichier (ici fase.xsd) à la racine du SU généré et de recompiler le SA via la tâche Ant.

2.5.5 SE JSR181

Problems @ Javadoc Declaration Properties Console		
Component Instance jsr1812		
Core	Property	Value
Appearance	Component	
	01- name	
	02- endpoint	!
	01- endpoint	!
	02- annotations	!
	03- mtomEnabled	!
	04- serviceInterface	!
	05- typeMapping	!
	06- wsdlResource	!
	07- style	!
	08- interfaceName	
	01- namespace	!
	02- value	!
	09- service	!
	01- namespace	!
	02- value	!
	10- Exclusive properties	!
	01- pojo	!
	01- bean	
	01- class	
	01- location	!
	02- class value	!
	02- property	Click to add a new property property
	01- name	!
	02- Exclusive properties	!
	01- value	!
	02- ref	!
	02- pojoClass	
	01- location	!
	02- pojoClass value	!

Les propriétés obligatoires sont :

- x le nom de l'endpoint
- x le nom de service
- x le nom de la classe d'implémentation du jsr181 (via pojo ou pojoClass)
- x l'emplacement du répertoire contenant le package et le fichier .class ou l'emplacement du fichier .jar contenant le package et la classe.

Voici un exemple de configuration correcte :

Component Instance jsrResponseWriter		
Property	Value	
Component		
01- name	jsrResponseWriter	
02- endpoint		
01- endpoint	responseWriter	
02- annotations		
03- mtomEnabled		
04- serviceInterface		
05- typeMapping		
06- wsdlResource		
07- style	document	
08- interfaceName		
09- service		
01- namespace	rw	
02- value	responseWriterService	
10- Exclusive properties		
01- pojo		
02- pojoClass		
01- location	D:\datas\Janus\Code\Eclipse_Europa_Cimero_new\workspace2908\common_wor	
02- pojoClass value	be.etnic.janus.responsewriter.responseWriterServiceImpl	

Voici le fichier généré par Cimero :

```
<?xml version="1.0"?>
<!--
#####
    Generated by CimeroEditor V2
    Bull S. A. S., Rue Jean Jaures, B.P.68, 78340, Les Clayes-sous-Bois
#####
-->

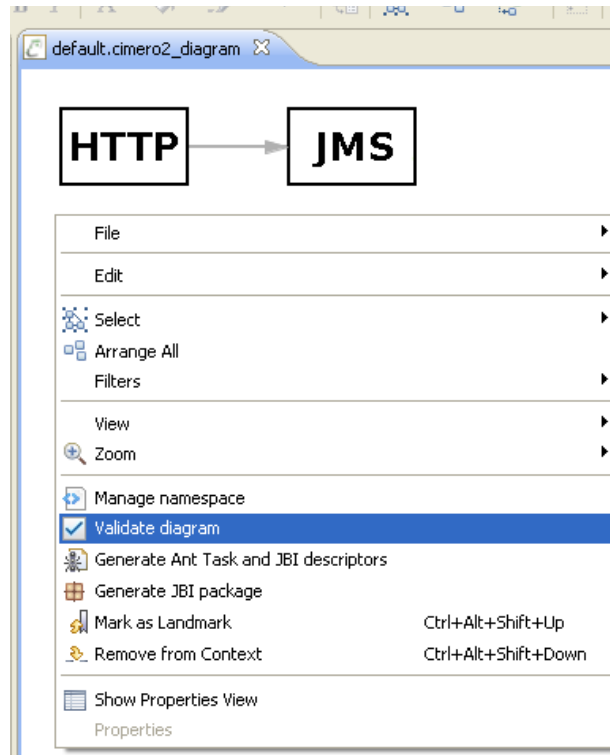
<beans xmlns:jsr181="http://servicemix.apache.org/jsr181/1.0"
xmlns:janus="http://www.etnic.be/janus"
xmlns:rw="http://www.etnic.be/janus/responseWriter"
xmlns:fh="http://www.etnic.be/janus/faultHandler">

<jsr181:endpoint
    service="rw:responseWriterService"
    endpoint="responseWriter"
    style="document"
    pojoClass="be.etnic.janus.responsewriter.responseWriterServiceImpl"/>

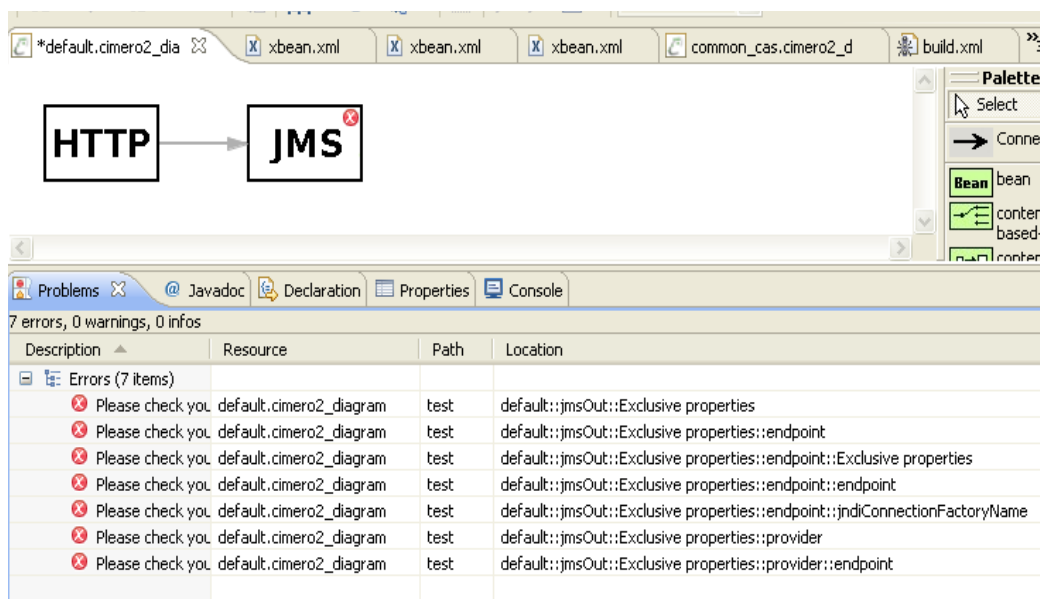
</beans>
```


2.6 Validation

Vous pouvez valider votre diagramme en cliquant droit dans le diagramme pour faire apparaître le menu déroulant et en cliquant sur « Validate diagram » ou en utilisant la barre d'outil d'Eclipse.

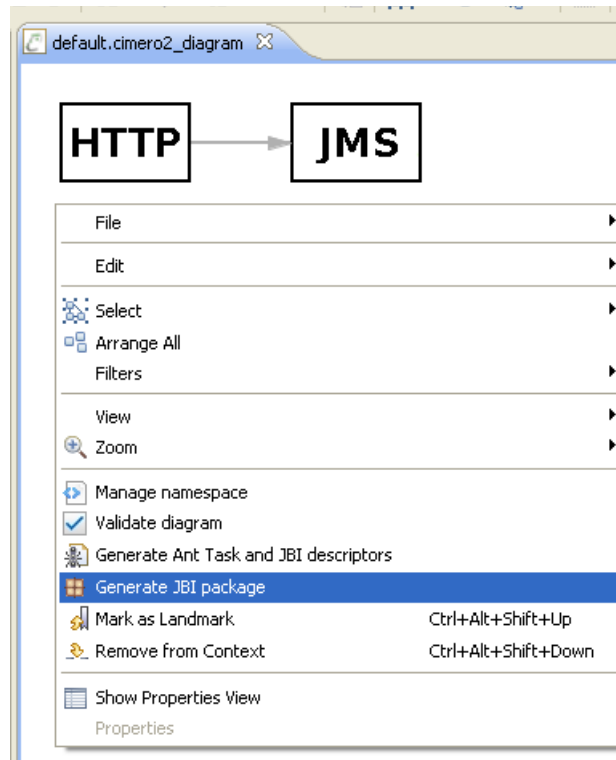


Si votre diagramme est valide, il ne se passe rien sinon, une croix rouge apparaît sur le ou les composants incorrects. Vous pouvez visualiser les erreurs plus en détails dans la vue « Problems » d'Eclipse.

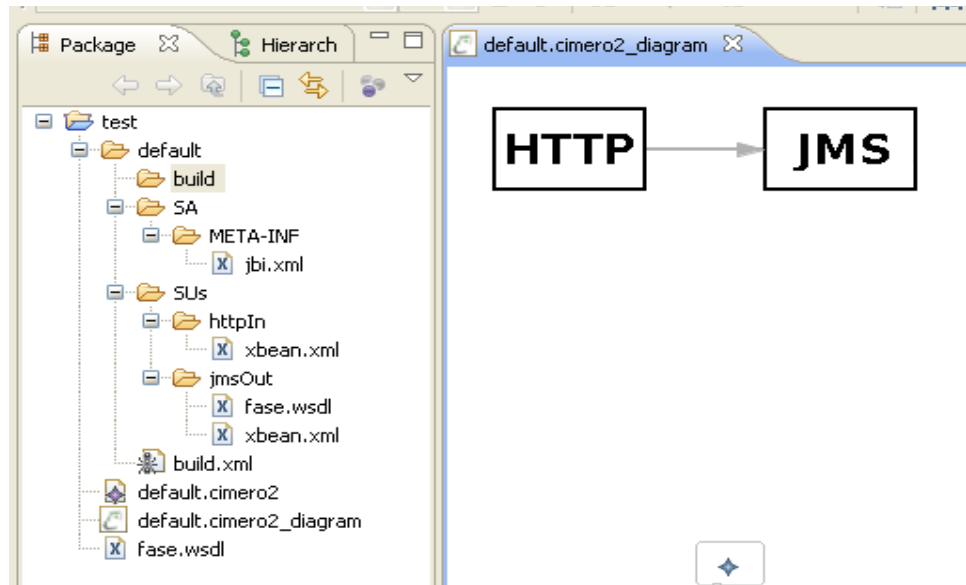


2.7 Génération JBI

Si le diagramme est valide, vous pouvez générer le package JBI, sinon la validation appelée automatiquement montre les erreurs de la façon décrite précédemment. Cliquez droit dans le diagramme pour faire apparaître le menu déroulant et cliquez sur « Generate JBI Package » ou utilisez la barre d'outil d'Eclipse.



Dans l'arborescence « Package Explorer », vous devez voir apparaître un dossier du même nom que votre diagramme. Si vous le déployez, vous trouverez un dossier build contenant le SA et les SU packagés ainsi qu'un répertoire SA contenant le fichier jbi.xml et un dossier Sus contenant toutes les SU et leurs fichiers de configuration (xbean.xml).

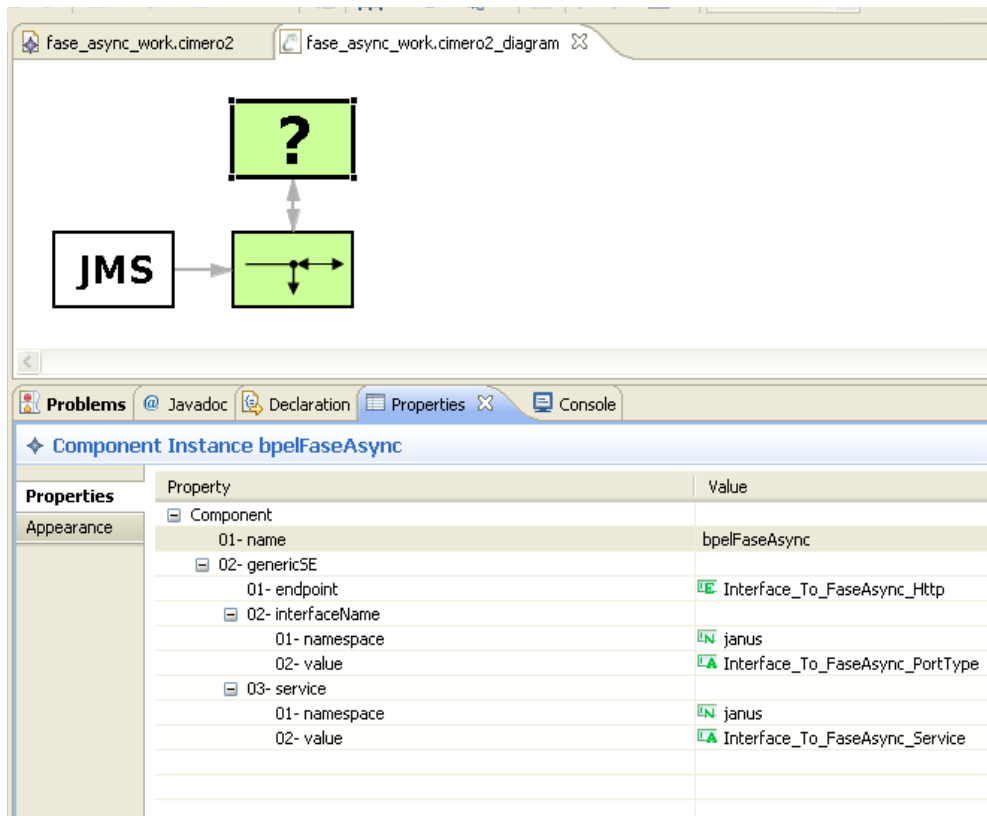


Il vous est ensuite possible de modifier votre diagramme et de régénérer cette arborescence via « Generate JBI Package » (toutes les modifications manuelles seront alors perdues) ou bien de régénérer uniquement le contenu du répertoire build via la tâche ant (voir régénération SA).

Remarque : Si vous souhaitez ajouter / compléter des Sus manuellement, vous pouvez utiliser la fonction Generate Ant Task and JBI descriptors. Elle génère la même chose que la fonction précédente mais sans les ZIP dans build. Pour les créer il vous suffit de lancer la tâche Ant.

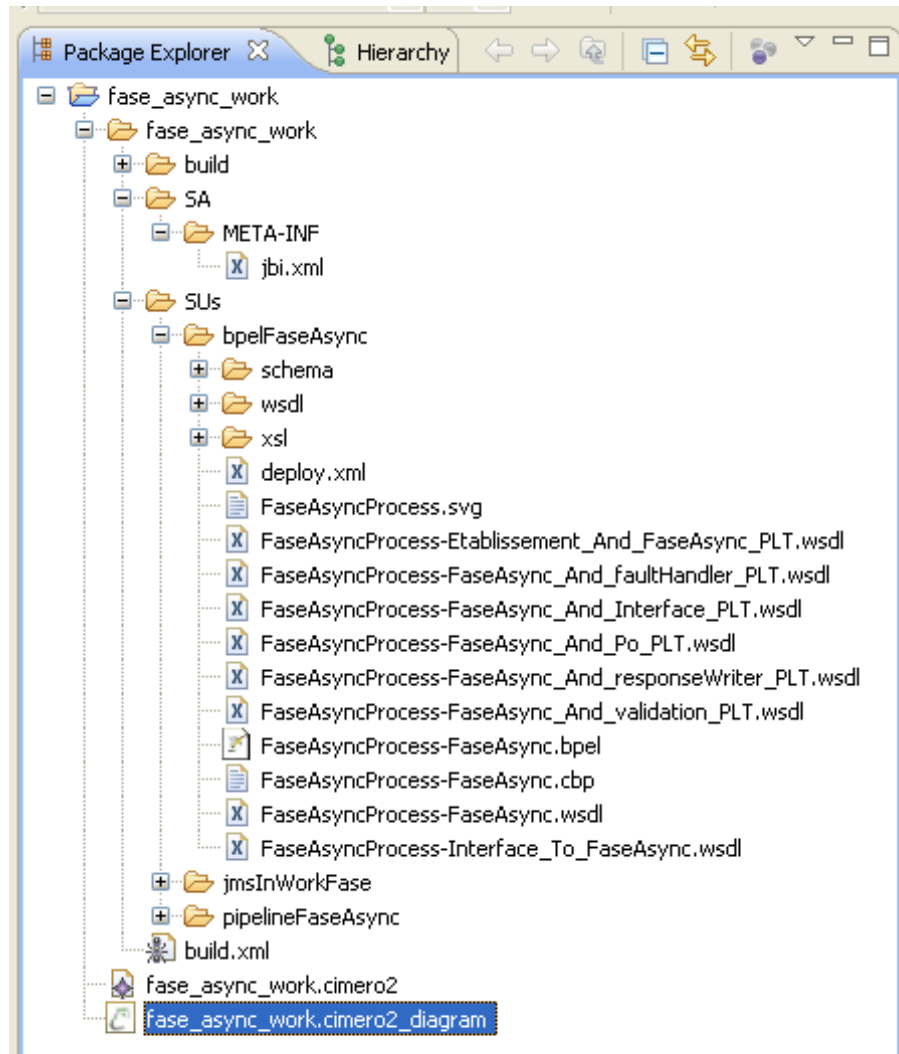
2.8 Ajout d'une SU

Il peut être utile de rajouter une SU dans le cas où Cimero ne comporte pas un des composants de votre architecture. C'est le cas du composant BPEL par exemple.



Créez un composant générique dans votre diagramme (symbolisé par un point d'interrogation) et nommez le bpelFaseAsync par exemple. Les autres propriétés (nom de service, ...) ne sont pas utiles dans ce cas-ci.

Générer votre packaging JBI comme d'habitude, un répertoire bpel apparaît dans le répertoire Sus. Effacer le fichier xbean.xml généré et copiez-y à la place tous les fichiers nécessaires au composant BPEL (*.wsdl, *.xsd, *.bpel, deploy.xml, ...).



Editez ensuite le fichier `jbi.xml` (dans `SA/META-INF`) et modifiez les lignes suivantes sous le tag `<service-assembly>` en donnant le nom correct du composant « `OdeBpelEngine` »:

```
<!-- SU : bpelFaseAsync -->
<service-unit>
  <identification>
    <name>bpelFaseAsync</name>
    <description>bpelFaseAsync SU</description>
  </identification>
  <target>
    <artifacts-zip>fase_async_work_bpelFaseAsync.zip</artifacts-zip>
    <component-name>OdeBpelEngine</component-name>
  </target>
</service-unit>
```

Enfin, n'oubliez pas de régénérer votre SA.

2.9 Régénération du SA

Si vous modifiez quoi que ce soit au niveau de vos Sus manuellement :

- x ajout d'un fichier
- x modification du xbean.xml généré
- x ...

, il vous est possible de régénérer le SA sans perdre vos modifications.

Il suffit de sélectionner le fichier build.xml généré (tâche ant – symbolisée par une araignée), de faire apparaître le menu déroulant (clic droit) et de sélectionner « Run As » et ensuite « Ant build ».

