
SOA

(Web Services, ESB , BPEL / Bpm, ...)

Table des matières

I - Présentation SOA / Web Services / Integration.....	5
1. SOA (présentation).....	5
2. SOA au niveau du Middle office.....	7
3. SOA et Intégration.....	8
4. Paradigme "tout est service"	9
5. Gains de l'approche SOA et enjeux.....	10
6. SOA et mode asynchrone.....	12
7. Repères SOA (architecture , technologie, ...).....	13
8. Deux grands types de WS (REST et SOAP).....	15
II - Api JAX-WS et implémentations.....	17
1. Présentation de JAX-WS.....	17

2. Mise en oeuvre de JAX-WS (coté serveur).....	17
3. Tests via SOAPUI.....	21
Utilisation de JAX-WS coté client.....	22
4. Redéfinition de l'URL d'un service à invoquer	23
5. Client JAX-WS sans wsimport et directement basé sur l'interface java.....	24
6. Transfert des exceptions via SOAP (Fault).....	25
III - CXF (mise en œuvre avec Spring)	28
1. CXF.....	28
IV - E.A.I. & E.S.B. (présentation).....	32
1. EAI et ESB.....	32
2. EAI (Enterprise Application Integration).....	32
3. ESB (Enterprise Service Bus).....	35
4. Notion de "moteur de services".....	40
5. Catégories d'ESB.....	41
V - Mule ESB.....	44
1. Mule Esb (présentation).....	44
2. IDE Mule Studio (utilisation).....	44
3. Configuration Mule Esb (essentiel).....	48
4. Environnement Mule Esb (standalone ou intégré).....	56
VI - ESB "Apache ServiceMix" (OSGi , camel).....	59
1. Présentation de l'ESB "ServiceMix".....	59
2. OSGi (présentation).....	62
3. Spring Dynamic Module et OSGi Blueprint.....	64
4. Camel.....	71
5. Integration "camel+cxfr" dans un bundle OSGi de serviceMix.....	74
VII - Modélisation SOA (problématiques , bpmn , ...).....	76
1. Modélisation et problématiques SOA.....	76
2. Transactions longues et compensations.....	77
3. Intégrité référentielle & MDM.....	78
4. Quelques repères d'urbanisation.....	80
5. Quintessence d'une bonne modélisation SOA.....	81
6. BPMN.....	86

VIII - BPEL (présentation , mode synchrone).....	90
1. BPEL (présentation).....	90
2. BPEL (détails).....	96
IX - ODE et BPEL_Designer.....	110
1. ODE (Moteur BPEL d'Apache).....	110
2. Plugin eclipse "BPEL Designer"	111
X - BPEL asynchrone.....	119
1. BPEL en mode asynchrone.....	119
XI - Bpmn2 , activiti , jbpm (présentation).....	122
1. jBPM et activiti (présentation).....	122
XII - Activiti (bpmn2/java , intégration Spring)	124
1. Activiti (présentation détaillée).....	124
2. intégration de Activiti au sein de Spring.....	129
3. (interactive) UserTask – activiti.....	132
4. Framework "generic_bpm" de mycontrib.org.....	136
XIII - jBPM et drools (moteur de règles).....	137
1. Jbpm (présentation détaillée).....	137
2. Drools (présentation).....	141
3. Événements et tâches de jbpm 5.....	145
4. Bpmn2 webDesigner (of drools).....	150
XIV - Annexe – SOAP et WSDL.....	152
1. Protocole SOAP	152
2. Eléments du protocole SOAP.....	152
3. WSDL (Web Service Description Language).....	154
4. "Basic profile" pour services "web".....	156
5. "Style/use" SOAP (rpc/literal , document/literal).....	156
6. Détails sur WSDL.....	160
XV - Annexe – Services REST et JAX-RS.....	164
1. Web Services "R.E.S.T."	164
2. API java pour REST (JAX-RS).....	165

XVI - Annexe – Sécurité sur les services WEB.....	171
1. Gestion de la sécurité / Services Web	171
2. Authentification basique http avec JAX-WS.....	175
3. Authentification soap via WS-Security et jax-ws/cxf.....	180
XVII - Normes JBI et SCA (présentation).....	188
1. SCA (Service Component Architecture).....	188
2. JBI (Java Business Integration).....	189
3. Détails de l'architecture JBI.....	191
XVIII - Annexe – Annuaire de services et UDDI.....	193
1. Utilité d'un annuaire de "service WEB".....	193
2. Annares UDDI (présentation).....	194
3. Annares publics et annuares privés.....	194
4. Structure d'un annuaire UDDI.....	196
5. taxonomies.....	200
XIX - Annexe – Intalio Bpms Designer (->bpel).....	203
1. Intalio Bpms (BPMN → BPEL).....	203

I - Présentation SOA / Web Services / Integration

1. SOA (présentation)

1.1. Agilité visée dans l'architecture SOA

Architecture **SOA** (*orientée services*)

SOA:

- faible couplage entre les différents services
- découplage entre invocation et implémentation (avec éventuels intermédiaires [adaptateur,...])
- fait ressortir l'aspect logique (services rendus)

==> manifestations:

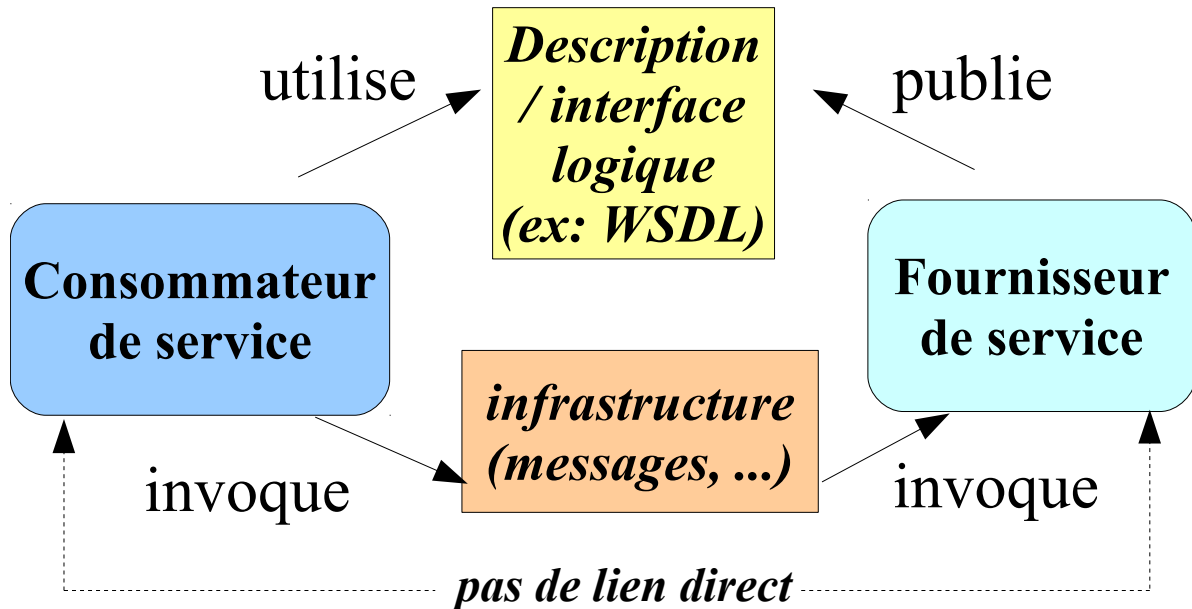
- * *infrastructure & organisationnel* ==> **ESB**
- * *fonctionnel / orga.* ==> **BPM / BPEL**, pivot , urba.
- * *technique* ==> **service web** (interface fonctionnelle
+ port logique d'invocation dans *.WSDL*
et url vers point d'accès à une certaine implémentation)

Une **architecture "SOA"** vise essentiellement à mettre en œuvre un **S.I. Agile** :

- souple
 - modulaire
- facilement reconfigurable

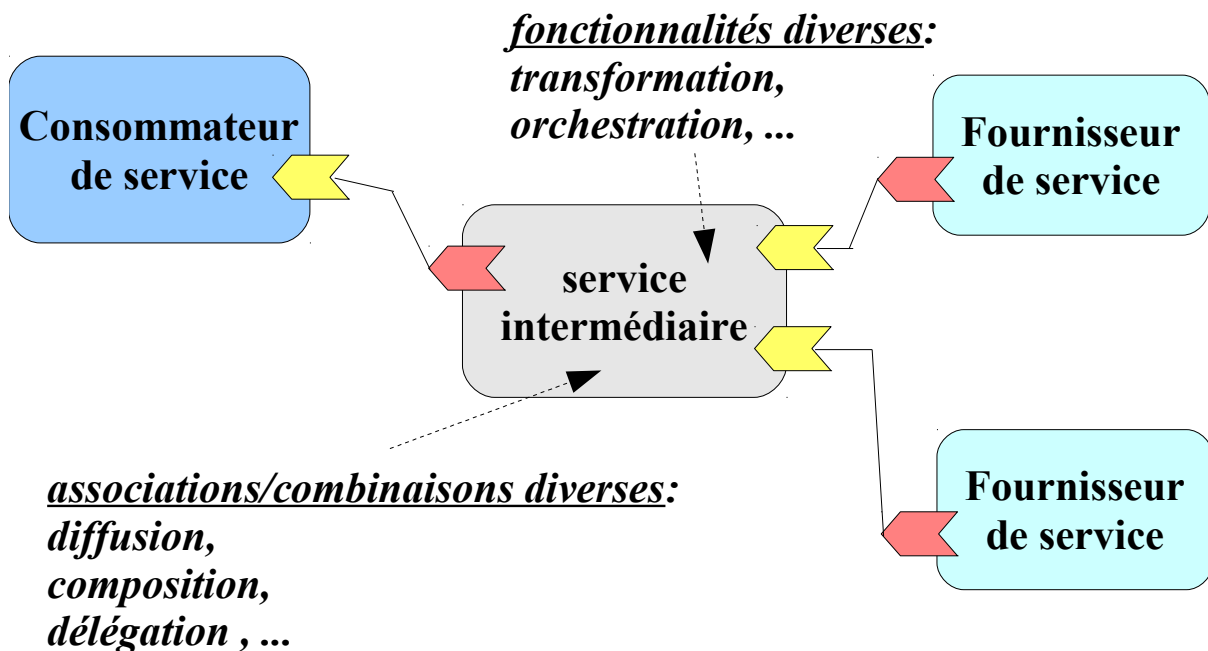
1.2. Découplages entres invocations et implémentations

SOA : Combinaison de services sur le mode "fournisseurs/consommateurs faiblement couplés"

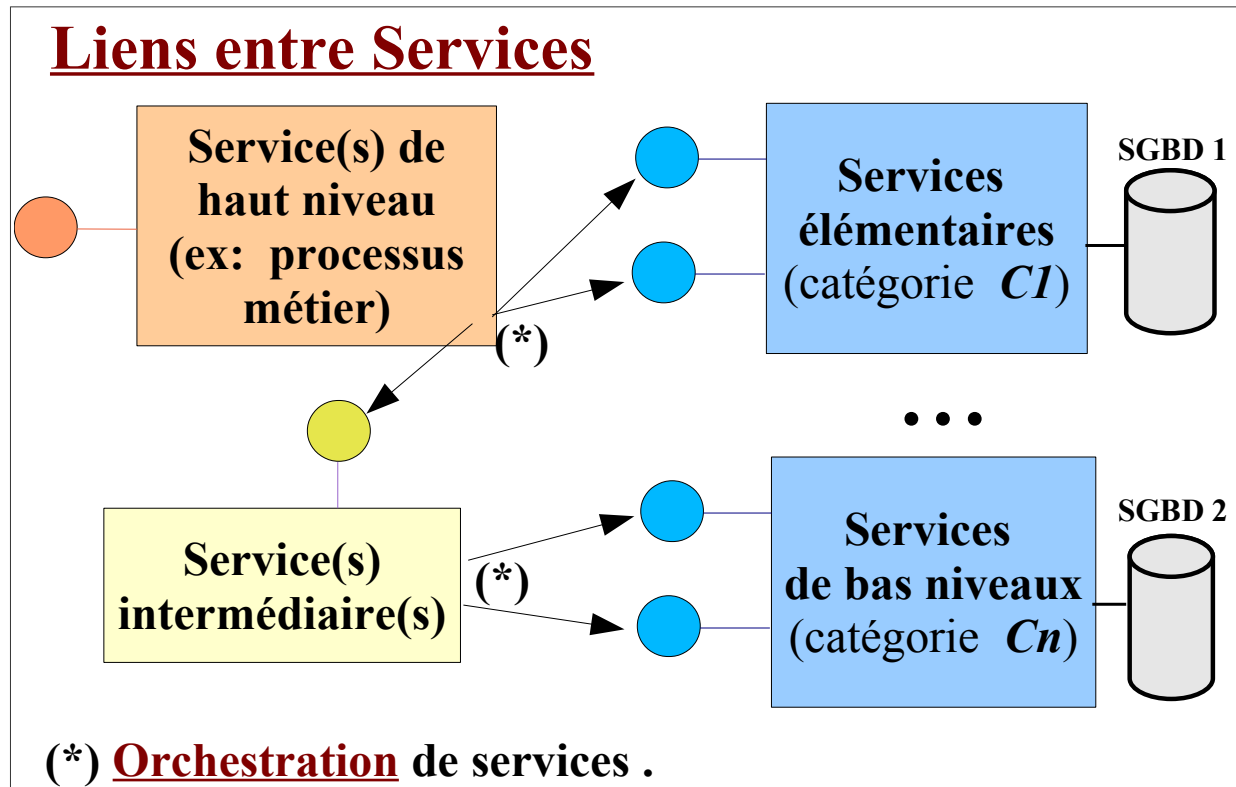


1.3. Consommation et offre de services

SOA : Services intermédiaires = fournisseurs et consommateurs.



1.4. Différents niveaux de services



2. SOA au niveau du Middle office

Beaucoup d'éléments de l'architecture SOA (*ESB, adaptateurs, processus BPEL, ...*) sont utilisés sur la partie "Middle Office" d'un SI d'une assez grande entreprise .

Autrement dit ,

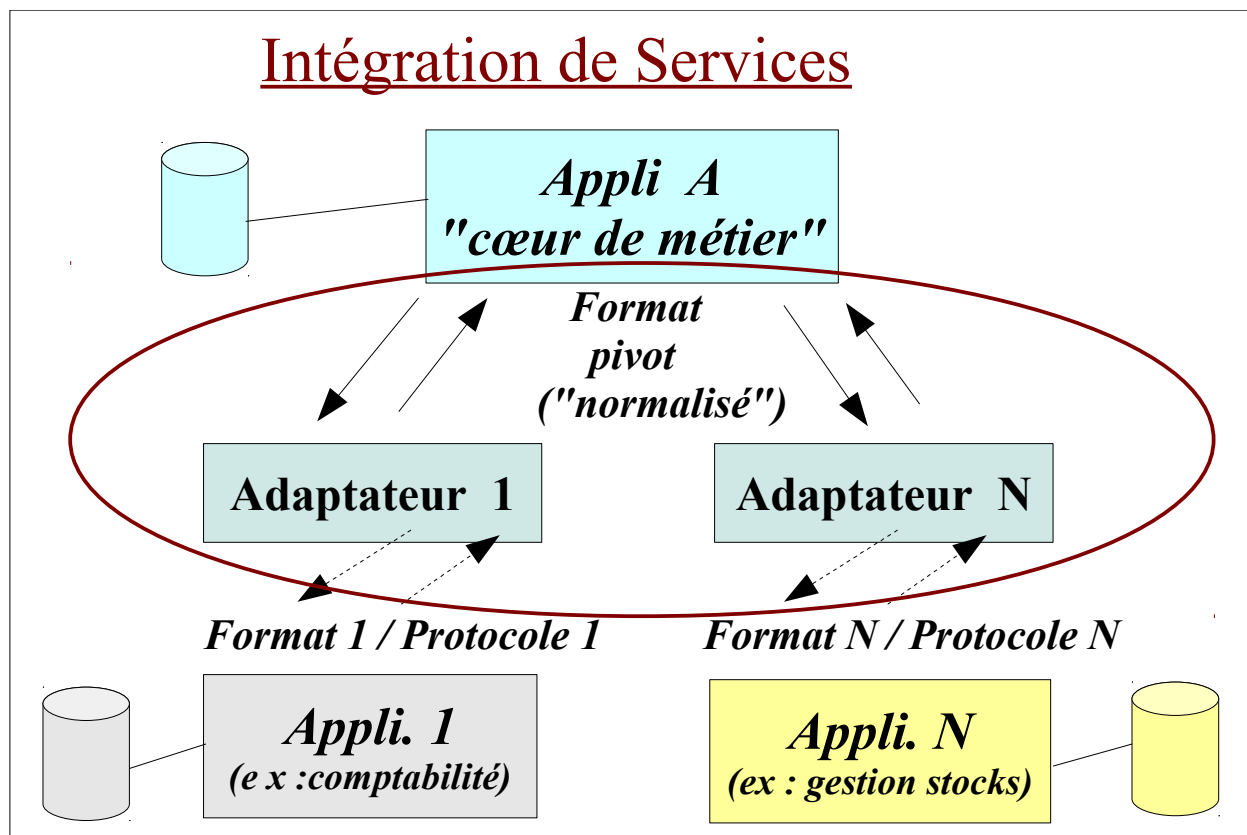
- Mise à part les web services REST, SOA n'est pas souvent directement lié à la partie IHM/web (c'est le rôle de la partie front-office)
- SOA n'est que rarement lié aux SGBDR (c'est le rôle de la partie back-office)
- **SOA est avant tout à voir comme un système fonctionnel .**

3. SOA et Intégration

L'architecture SOA est souvent utilisée pour **intégrer** dans un **SI spécifique au cœur de métier** des **applications existantes (non développées en interne mais achetées)** qui **rendent des services transverses ou génériques** (ex: Compta , gestion de Stock, ...).

Pour faire communiquer entre elles des applications provenant de différents éditeurs , on a besoin de tout un tas d'éléments **intermédiaires** fournis par l'**infrastructure SOA** (*ESB* , *Adaptateurs*, ...).

Dans le cas d'une application "clef en main" achetée auprès d'un éditeur de logiciels, on ne maîtrise absolument pas le format exact des services offerts (liste des méthodes , structures des données en entrées et en sorties des appels). Via des paramétrages de transformations "ad-hoc" effectuées au niveau d'un ESB, il sera (en général) tout de même possible d'invoquer cette application depuis d'autres applications internes au SI de l'entreprise.



Une ancienne norme SOA du monde Java s'appelait JBI (Java Business **I**ntegration)

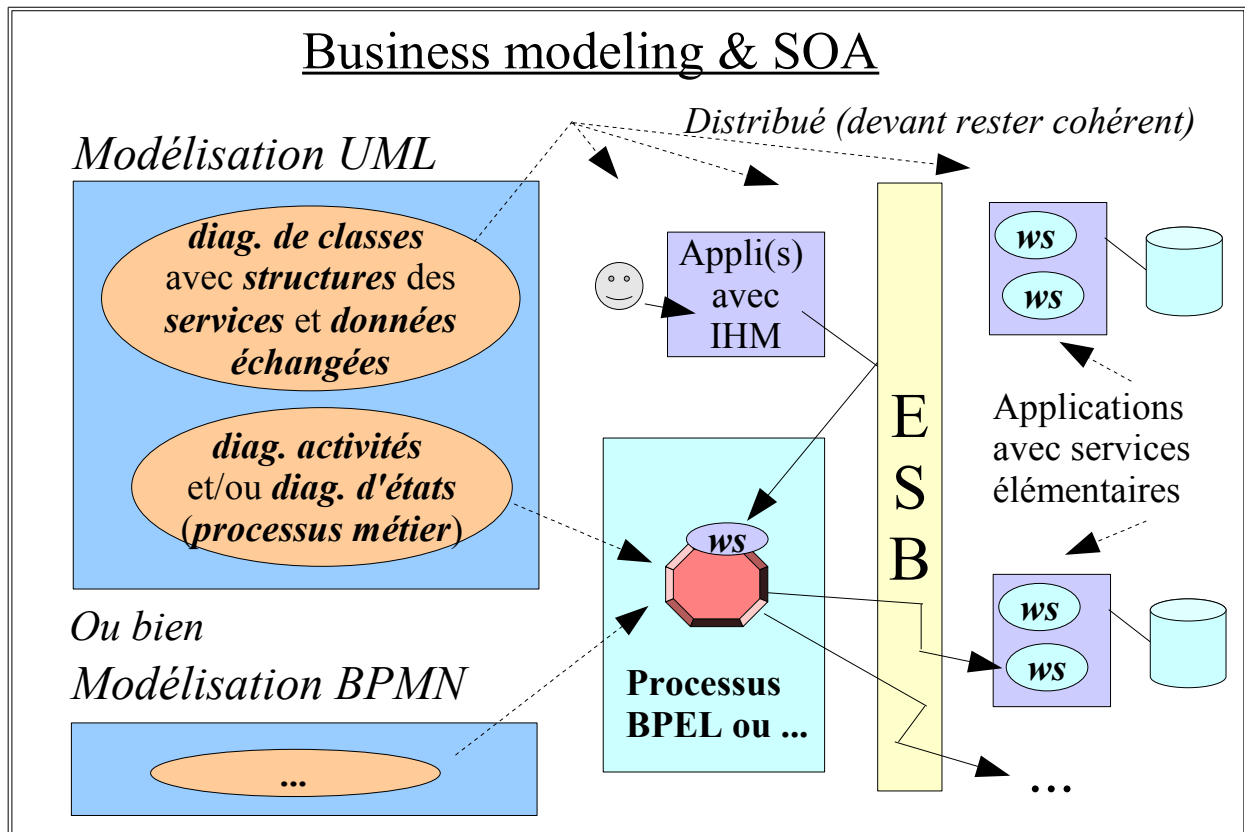
Attention, attention :

Les "Web-Services", ESB et "Processus BPM" ne constituent que l'*infrastructure technique*.

Un vrai projet SOA d'entreprise doit absolument comporter en outre un volet "**MODELISATION**" (avec UML et/ou BPMN) qui est vraiment **fondamental** !!!

SOA c'est un peu "**programmer/configurer le SI à grande échelle**".

Vue la grande portée de SOA, l'aspect modélisation est très important .



4. Paradigme "tout est service"

Au sein d'une architecture SOA , on peut (dans une première approche conceptuelle) considérer que presque tous les éléments d'un système informatique sont des services :

- services de persistance (lectures & mises à jour dans une base de données ou ...)
- services d'habilitation (vérifier les privilèges d'accès , ...)
- services de vérifications , contrôles ,
- services métiers
-

Mise à part quelques contraintes techniques dont il faut tenir compte (transactions courtes proches des données , performances correctes , sécurité ,) , la plupart des services (ou sous services) peuvent se mettre en œuvre de manières très diverses :

- prise en charge directe par l'entreprise ou bien sous traitance (SAAS ,) .
- service local , proche ou lointain (géographiquement) .
- service programmé sur mesure ou bien "acheté + adapté"
-

==> Il convient donc d'effectuer une première phase de modélisation SOA en prenant du recul et en ne se focalisant que sur des considérations purement sémantiques/fonctionnels .

Beaucoup d'autres aspects ("organisationnel" , "logiciel" , "géographique" ,) sont idéalement à modéliser comme des aspects séparés (que l'on règle par paramétrages ou via des implémentations dédiées) .

5. Gains de l'approche SOA et enjeux

5.1. Principaux apports de l'approche SOA :

- **interopérabilité** (java , .net , c++ , php , ...)
- **flexibilité/agilité** (relocalisations et/ou reconfigurations possibles des services via intermédiaire de type ESB)
- **partage de traitements métiers** (règles métiers , services d'entreprise , ...)
- **partage de données** (accessibles via les services) et donc moins de besoin(s) de "re-saisie" ou de "copie/réplication" .
- **urbanisation plus aisée** (chaque partie peut évoluer à son rythme tant que certains contrats fonctionnels sont respectés en utilisant s'il le faut des adaptateurs).
- ...

Ces apports sont tout de même **conditionnés par** :

- le besoin quasi-impératif d'une **bonne modélisation** SOA (à caractère *transverse*)
- l'**utilisation maîtrisée de technologies SOA** (WS,ESB,...) fiables et performantes.

5.2. Enjeux pour l'entreprise :

- **Potentielles économies d'échelle** (si partage effectif de services mutualisés).
- **Meilleur réactivité** (du fait d'un SI plus souple/flexible/agile) .
- **Meilleurs communications externes** (B2C , B2B) via des services publics visibles à l'extérieur de l'entreprise.
- ...

Et selon la qualité de la modélisation et de la mise en œuvre :

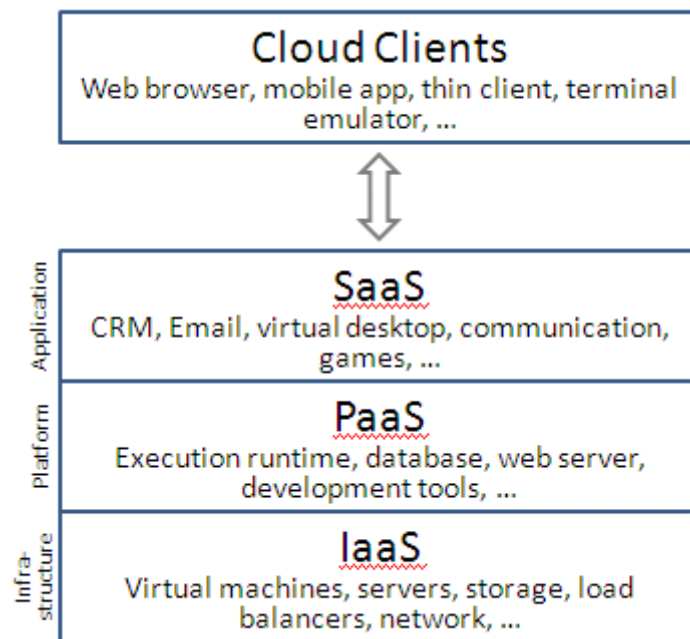
- **Peut être une maintenance plus aisée**
(il est plus facile de maintenir ou faire évoluer plusieurs petites applications assez indépendantes et qui communiquent via SOA que de maintenir une énorme application monolithique).
- Des performances à peu près aussi bonnes (si ESB bien choisi et bien paramétré).

5.3. Enjeux stratégiques (liés au cloud-computing)

Le concept de **cloud-computing** (à dimensionnement flexible) est actuellement à la mode et considéré quelquefois comme un enjeu stratégique .

La mise en œuvre du "**cloud-computing**" s'appuie essentiellement sur :

- **Saas** (software as a service)
- **Paas** (platform as a service)
- **Iaas** (infrastructure as a service)



Le "as a service" montre bien à quel point SOA peut être utile à la mise en œuvre du cloud .
Même certains aspects très techniques (configuration d'un parc de machines virtuelles (linux ou ...) en cluster ou ...) peuvent être pilotés via des services WEB.

Exemple (quelques AWS : Amazon Web Services) :

- *Amazon Elastic Compute Cloud (EC2)* → fournissant des serveurs virtuels évolutifs utilisant Xen.
- *Amazon Elastic Block Store (EBS)* → fournissant un niveaux de blocs persistants pour les volumes de stockage EC2.
- *Amazon Simple Storage Service (S3)* → *fournissant un stockage basé sur les services web.*
- *Amazon Simple Queue Service (SQS)* → fournissant une file de messages hébergé pour les applications web.
- ...

6. SOA et mode asynchrone

Liens entre "appel de fonction" et document

Une *structure XML* de type

```
<commande>
  <numero>1</numero>
  <adresse>...</adresse>
  ...
</commande>
```

peut aussi bien représenter :

- * une requête vers une opération d'un service web
- * un document transmis à traiter/analyser

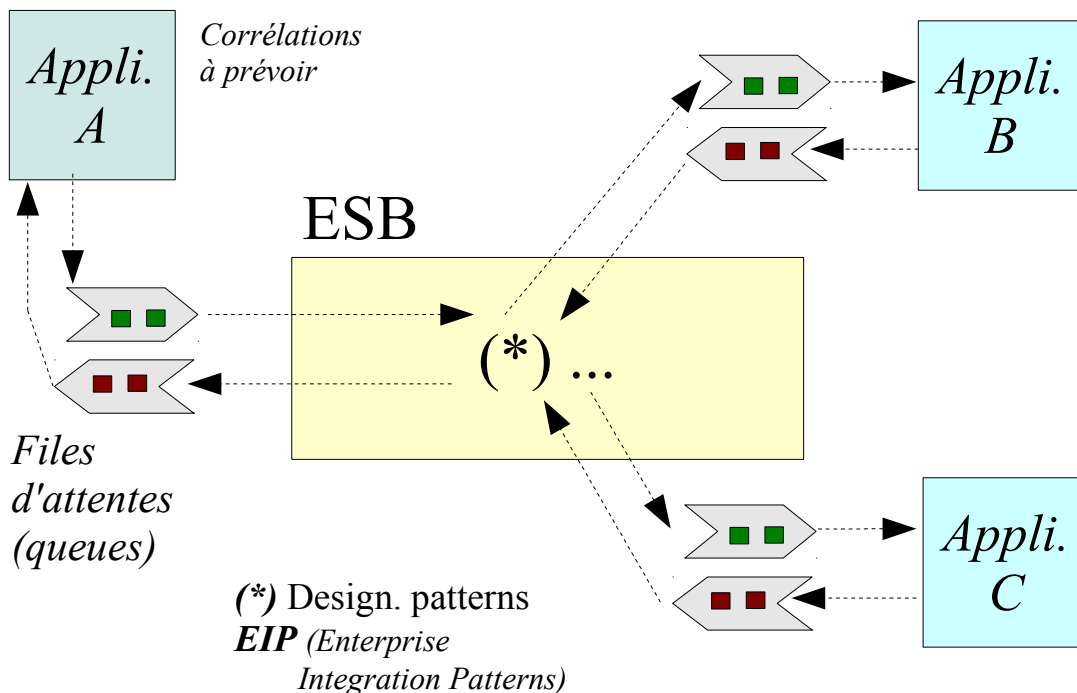
Cette structure peut être véhiculée par :

- * une enveloppe SOAP (elle même véhiculée par HTTP ou ...)
- * un message JMS (déposé dans une file d'attente)
- * ...

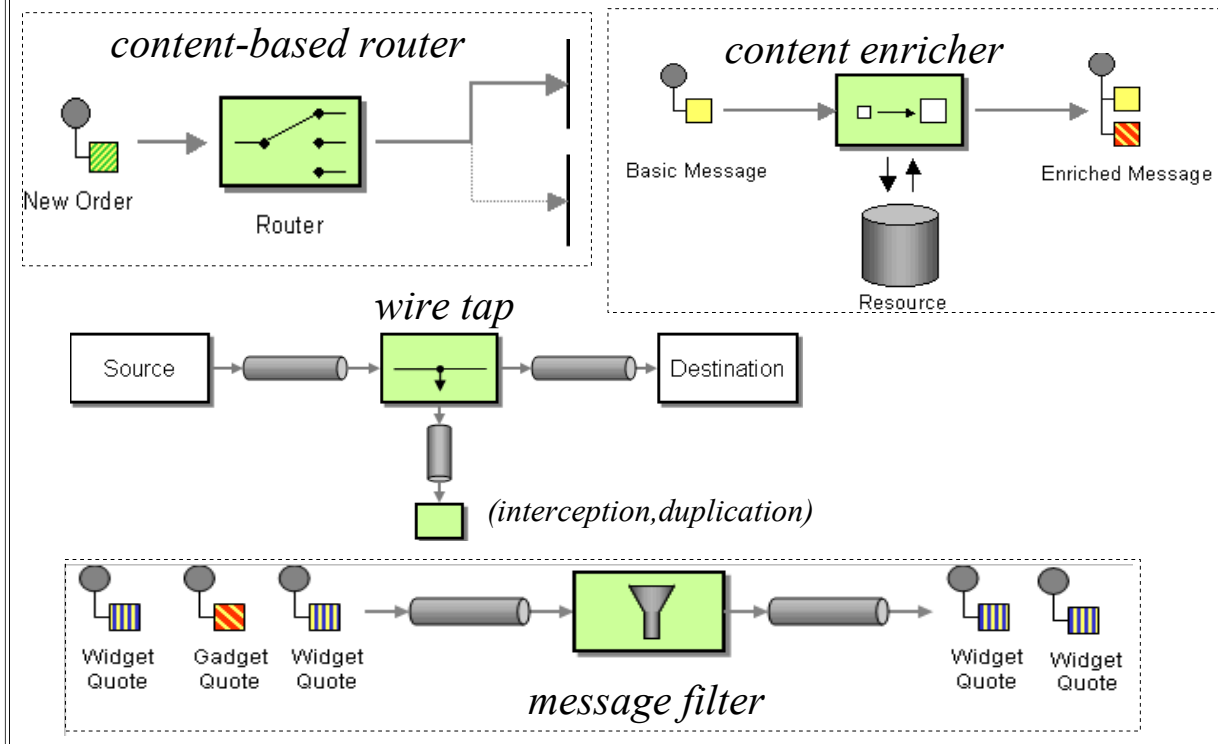
==> passerelles techniques envisageables entre :

- * synchrones et asynchrones
- * document et RPC (Remote Procedure Call) .

SOA en mode asynchrone



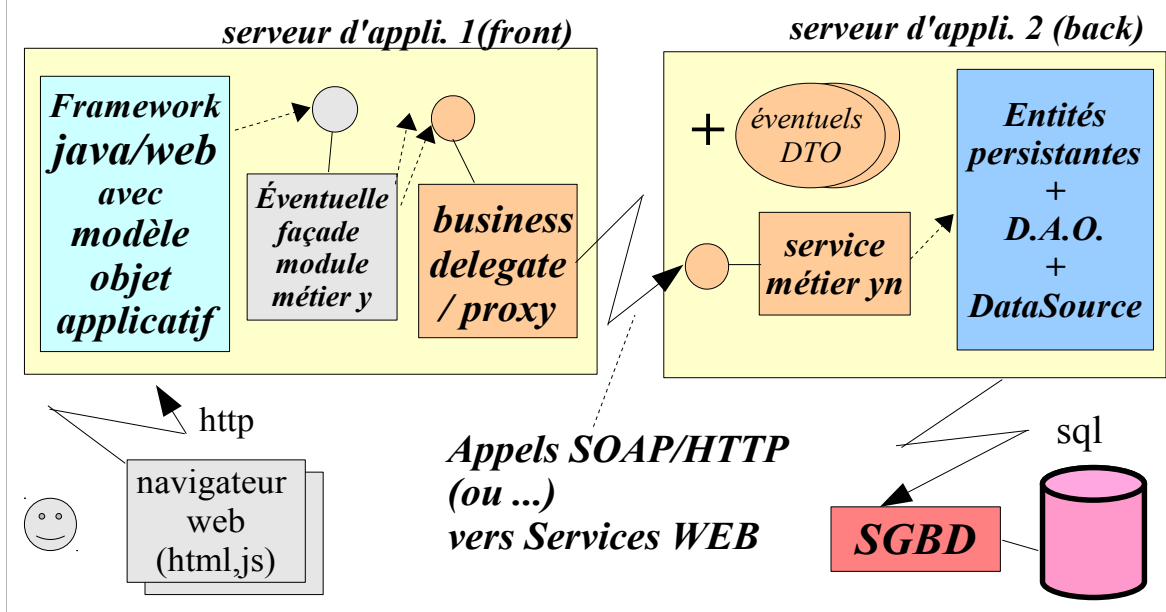
Quelques "design patterns" EIP (Enterprise Integration Patterns)



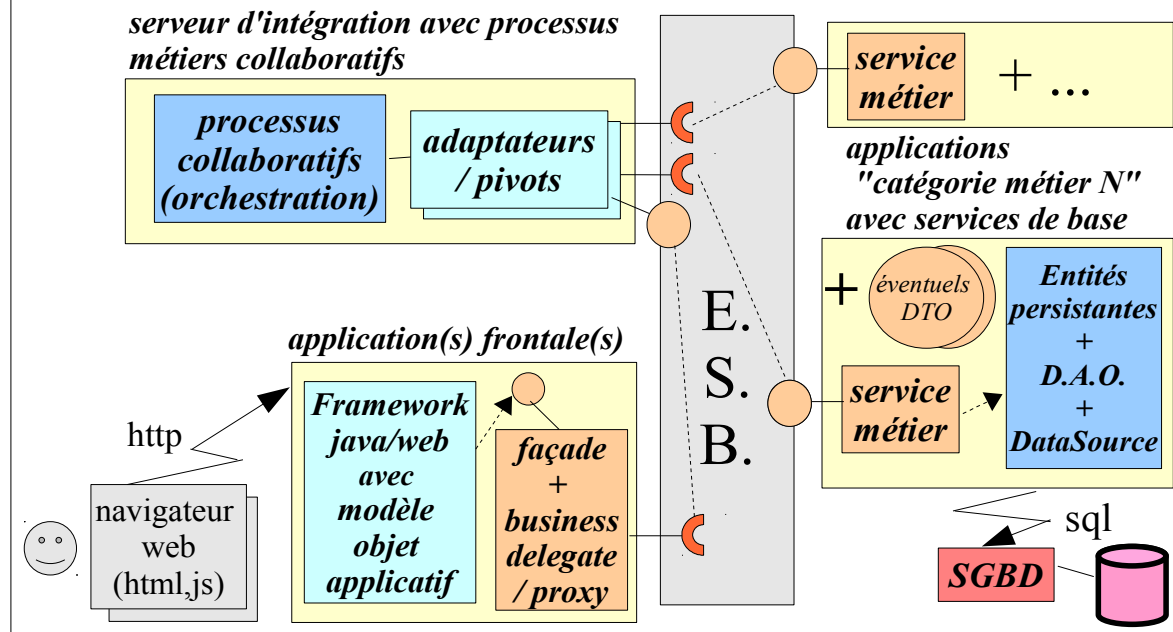
7. Repères SOA (architecture, technologie, ...)

7.1. Place de SOA dans une architecture n-tiers

Architecture 3-tiers **distribuée**: Application Web, services métiers sur 2 serveurs distincts.

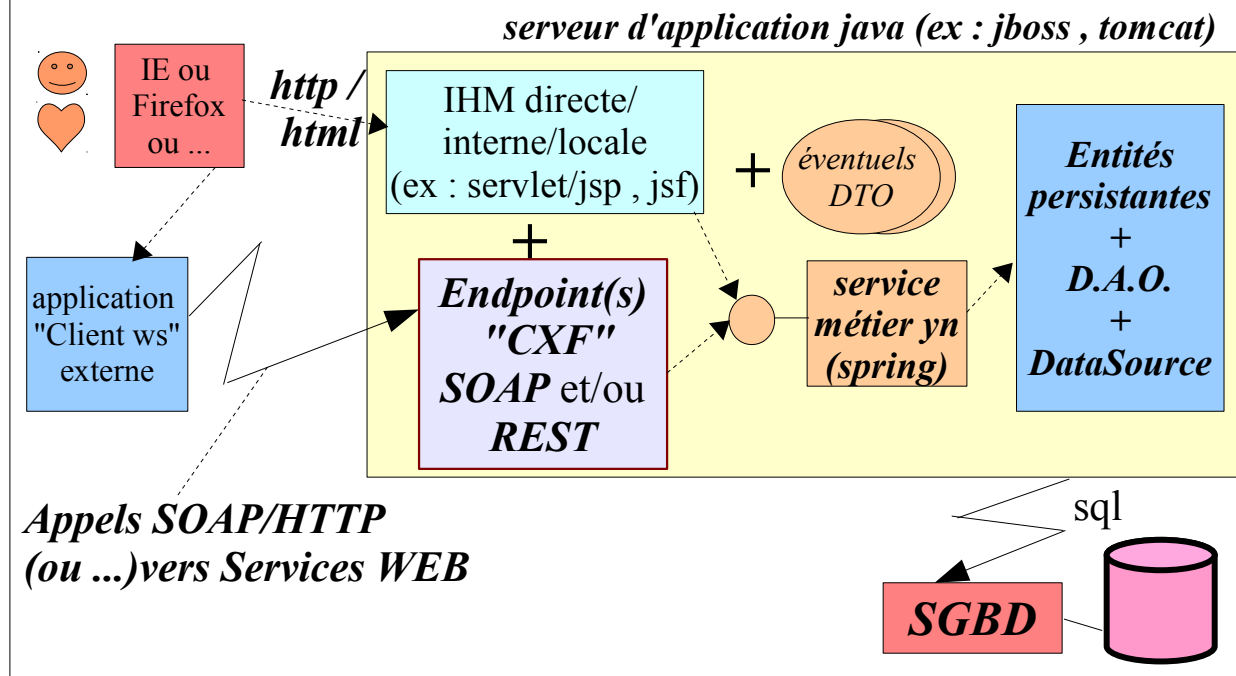


Architecture n-tiers orientée services (S.O.A.): avec Bus "E.S.B." et processus collaboratifs.



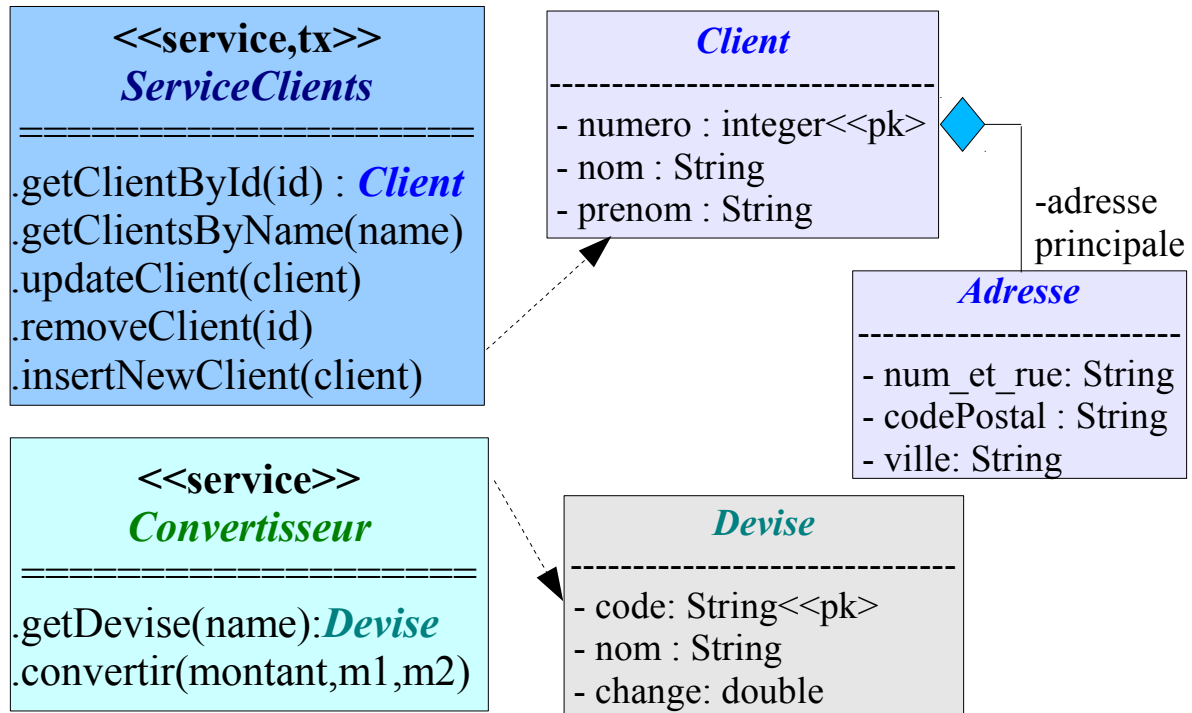
7.2. Exemple de mise en œuvre en java

WebService : Exemple de mise en œuvre avec Spring + CXF (en java) :



7.3. Modélisation essentielle (UML) d'un service web

Modélisation UML essentielle d'un Webservice (exemples)



8. Deux grands types de WS (REST et SOAP)

2 grands types de services WEB: **SOAP/XML** et **REST**

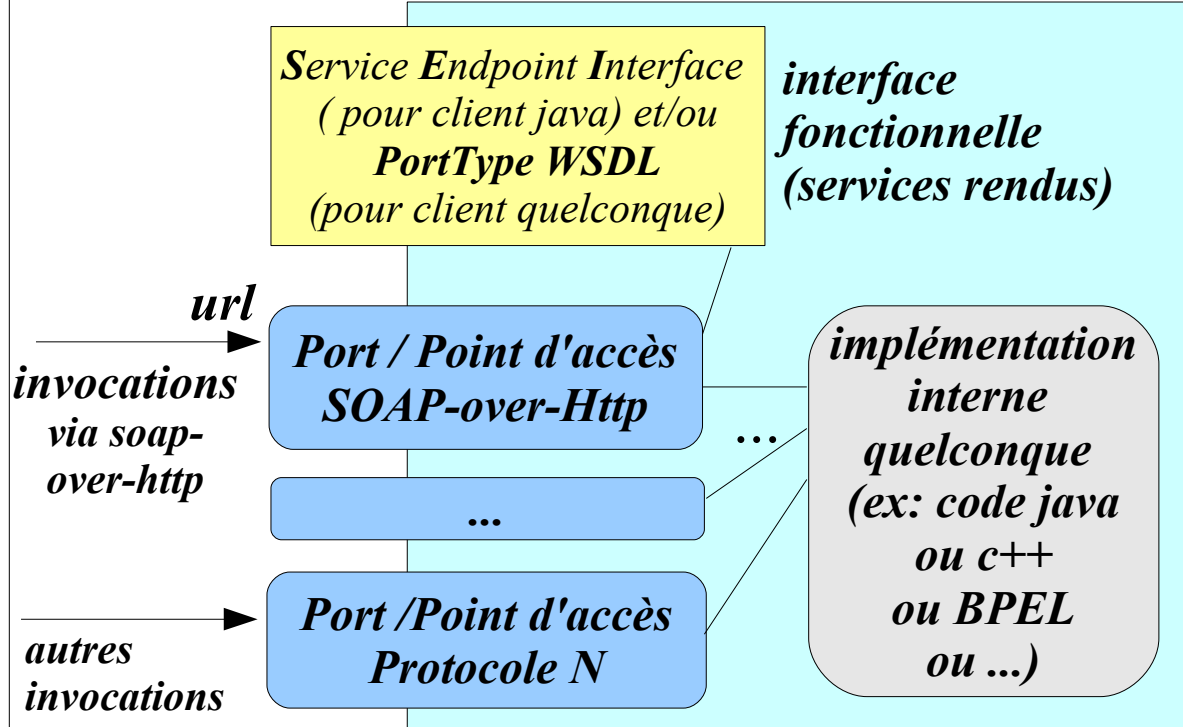
WS-* (SOAP / XML)

- "Payload" systématiquement en XML (sauf pièces attachées / HTTP)
- Enveloppe SOAP en XML (header facultatif pour extensions)
- Protocole de transport au choix (HTTP, JMS, ...)
- Sémantique quelconque, **description WSDL**
- Plutôt orienté SOA / proche "back office"

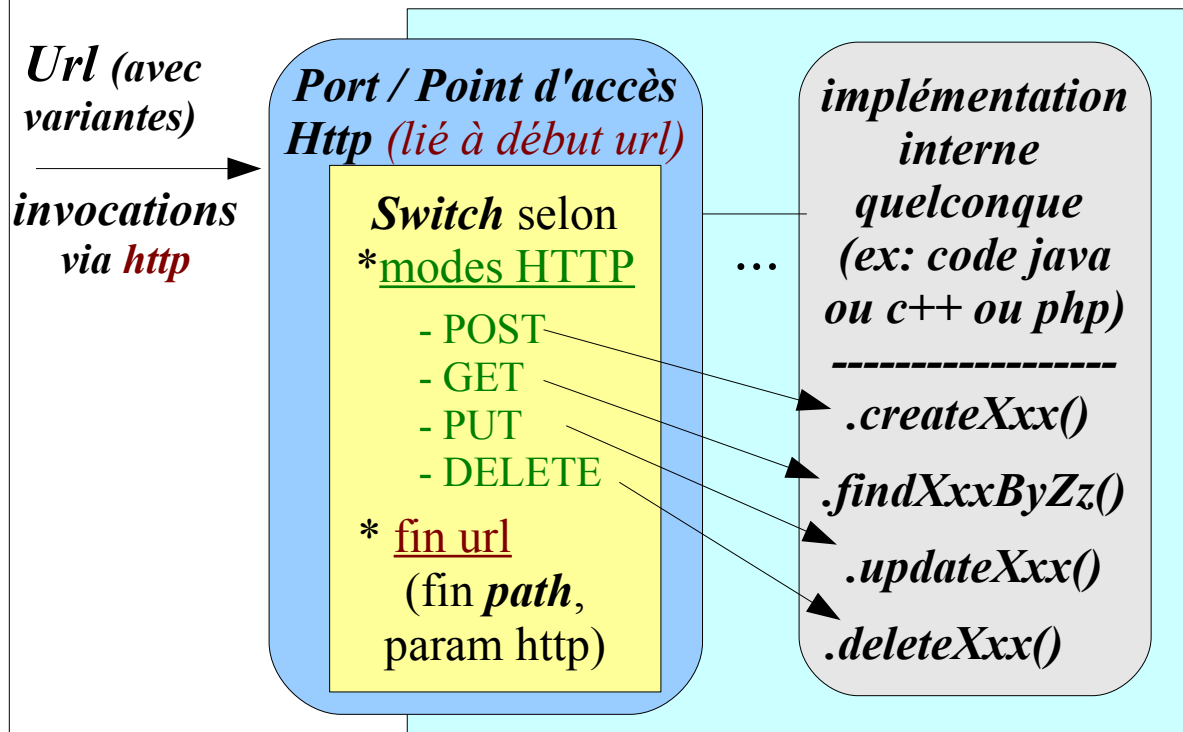
REST (HTTP)

- "Payload" au choix (XML, HTML, JSON, ...)
- Pas d'enveloppe imposée
- **Protocole de transport = toujours HTTP.**
- Sémantique "CRUD" (**modes http PUT, GET, POST, DELETE**)
- Plutôt orienté Web/Web2 et proche "front office"

Anatomie d'un service Web "SOAP"



Anatomie d'un service Web "REST"



II - Api JAX-WS et implémentations

1. Présentation de JAX-WS

JAX-WS signifie *Java Api for Xml Web Services*

L'api **JAX-WS** est une api utilisable à partir du jdk 1.5 et qui remplace maintenant l'ancienne api JAX-RPC . L'ancienne Api JAX-RPC n'est aujourd'hui intéressante que pour maintenir d'anciennes applications basées sur le jdk 1.4 .

L'api **JAX-WS ne fonctionne qu'avec le jdk 1.5 , 1.6 ou 1.7** et n'est donc utilisable qu'au sein des serveurs **JEE** récents (ex: *JBoss 4.2 , Tomcat 5.5 + Apache_CXF ,*).

Important : **JAX-WS est déjà intégré dans le jdk ≥ 1.6** (pas de ".jar" à ajouter mais "update récent du jdk1.6 conseillé pour une version récente et optimisée/performante de JAX-WS) .

Dans le cas du **jdk1.5** , **JAX-WS s'ajoute en tant que ".jar"** d'une des implémentations disponibles (metro , **CXF** ,).

Les principales spécificités de **JAX-WS** (vis à vis de JAX-RPC) sont les suivantes:

- paramétrage par **annotations Java5** (ex: `@WebService` , `@WebMethod`)
- utilisation interne de **JAXB2**

NB:

- L'utilisation "**coté client**" de JAX-WS ne nécessite que le **jdk ≥ 1.6** (ou des ".jar" additionnels pour le jdk 1.5)
- L'utilisation "**coté serveur**" de JAX-WS nécessite généralement une **prise en charge évoluée et intégrée** (ex : **EJB3** ou bien "**CxfServlet**" dans une **appli web Spring**).

2. Mise en oeuvre de JAX-WS (coté serveur)

La mise en oeuvre d'un service web coté serveur avec **JAX-WS** consiste essentiellement à utiliser les annotations `@WebService` , `@WebParam` , `@WebMethod` , `@WebResult` , au niveau du code source.

L'interface d'un service WEB nécessite simplement l'annotation `@WebService` et n'a pas absolument besoin d'hériter de `java.rmi.Remote` .

```
package calcul;
import javax.jws.WebService;

@WebService
public interface Calculator_SEI {
    //Calculator ou Calculator_SEI ou .... avec SEI = Service Endpoint Interface
    public int addition(int a,int b);
}
```

NB: les noms associés au service (**namespace**, **PortType**, **ServiceName**) sont par défaut basés sur les noms des éléments java (package , nom_interface et/ou classe d'implémentation) et peuvent

éventuellement être précisés/redéfinis via certains attributs facultatifs de l'annotation **@WebService**

NB: de façon à ce que les noms des paramètres des méthodes d'une interface java soient bien retranscrits dans un fichier WSDL, on peut les préciser via **@WebParam(name="paramName")**.

```
public int addition(int a,int b); ==> addition(xsd:int arg0,xsd:int arg1) dans WSDL
public int addition(@WebParam(name="a") int a, @WebParam(name="b") int b)
==> addition(xsd:int a ,xsd:int b) dans WSDL
```

L'annotation facultative **@WebResult** permet entre autre de spécifier le nom de la valeur de retour (dans SOAP et WSDL) . Par défaut le nom du résultat correspond à **"return"** .

NB: le sigle *SEI* signifie **Service EndPoint Interface** . il désigne simplement l'interface d'un service WEB et ne fait l'objet d'aucune convention de nom.

Au dessus de la classe d'implémentation , l'annotation **@WebService** peut éventuellement être sans argument si la classe n'implémente qu'une seule interface. Dans le cas où la classe implémente(ra) plusieurs interfaces, il faut préciser celle qui correspond à l'interface du Service Web:

```
@WebService (endpointInterface="package.Interface_SEI")
```

La classe d'implémentation d'un service Web peut éventuellement préciser via l'annotation **@WebMethod(exclude=true or false)** les méthodes qui seront vues du côté client et qui constituent le service web.

Si aucune annotation **@WebMethod** n'est mentionnée, alors toutes les méthodes publiques seront exposées.

```
package calcul;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@WebService (endpointInterface="calcul.Calculator_SEI")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface .... {

    @WebMethod(operationName="add")
    public int addition(int a, int b) .....
}
```

- **NB:** L'annotation *SOAPBinding* est facultative. Elle permet de choisir (quand c'est possible) un style SOAP (ex: Style.RPC ou Style.DOCUMENT) .
- Le mode par défaut **"document/literal"** convient **très bien** dans presque tous les cas .

NB: Ces annotations ne sont utiles que si elles sont interprétées par un serveur JEE5 ou 6 (ex: GlassFish de Sun ou Jboss 4.2 ou 5) ou bien par une technologie JAX-WS (telle que CXF) ajoutée à

un serveur J2EE (ex: Tomcat 5.5).

NB: Selon l'implémentation retenue de JAX-WS (metro , jboss-ws , CXF,) , les annotations sont acceptées ou pas lorsqu'elles sont directement placées au dessus de la classe d'implémentation.

Autres paramétrages importants (packages/namespaces):

- Lorsque l'interface et la classe d'implémentation d'un service WEB sont placées dans des packages java différents , ceci se traduit par différents "namespaces" xml dans le fichier WSDL. Préciser l'attribut **"targetNamespace"** dans l'annotation **@WebService** de l'interface et aussi dans l'annotation **@WebService** de la classe d'implémentation permet de bien contrôler la (ou les) valeurs attendues dans le WSDL .
- Certaines méthodes ont des paramètres (en entrée ou en sortie) qui correspondent à des classes de données (JavaBean/POJO) .
Placer **@XmlType(namespace="http://yyy/data")** permet de bien contrôler le namespace XML qui sera associé à ces classes de données.

Exemple :

```
@XmlType(namespace="http://entity.tp/")
@XmlRootElement(name="stat")
public class Stat {
    private int num_mois; //de 1 à 12
    private double ventes; //+get/set
    ...
}
```

```
@WebService(targetNamespace="http://minibank.myapp.tp/")
public interface GestionComptes {

    public Compte getCompteByNum(@WebParam(name="numCpt")long numCpt)
        throws MyServiceException;

    public List<Stat> getStats(@WebParam(name="annee")int annee);

    public void transferer(@WebParam(name="montant")double montant,
        @WebParam(name="numCptDeb")long numCptDeb,
        @WebParam(name="numCptCred")long numCptCred)
        throws MyServiceException;

}
```

```
@WebService(targetNamespace="http://impl.minibank.myapp.tp/",
    endpointInterface="tp.myapp.minibank.itf.domain.service.GestionComptes")
public class GestionComptesImpl implements GestionComptes {
    ...
}
```

namespace par défaut : "http://" + nom_package_java_à_l_envers + "/"

2.1. Implémentation sous forme d'EJB3 (pour serveur JEE5 tel que JBoss4.2 ou 5 , Glassfish de Sun , Geronimo 2 d'apache , Jonas d'OW2 ou ...)

- Combiner les annotations précédentes (@WebService ,) avec l'annotation **@Stateless** au niveau de la classe d'implémentation d'un EJB3 Session sans état.
- Déployer le tout selon les spécifications JEE (.jar , .ear) au sein d'un serveur d'application comportant un conteneur d'EJB3
- Repérer l'url du service WEB et de la description WSDL (**au cas par cas selon serveur**)
[ex: <http://localhost:8080/xyz/XXXBean?wsdl>]

2.2. Test sans serveur et wsgen (jdk 1.6)

NB: à des simples fins de "tests unitaires" , il est éventuellement possible de tester un service WEB en lançant une simple application java (non web) compilée avec le jdk 1.6 (*sans CXF*) qui comporte déjà en lui l'api "JAX-WS" et un mini mini conteneur Web appelé "Jetty" .

==> Ceci n'est valable que pour des petits tests (sans serveur) et nécessite tout de même l'invocation d'un script (.bat / .sh ou ant) qui lance la commande bin/**wsgen** du jdk 1.6 .

```
set JAVA_HOME=C:\.....\java\jdk1.6.....
set WKSP=D:\....\workspace
set PRJ=myProject
set SEI_Impl=basic.SimpleCalculatorImpl
"%JAVA_HOME%\bin\wsgen" -cp %WKSP%\%PRJ%\bin
                        -d %WKSP%\%PRJ%\bin          %SEI_Impl%
```

NB: ceci est complètement **inutile** (et même quelquefois source d'incompatibilités) avec une utilisation standard de *CXF* dans *Tomcat* ou bien des *EJB3* dans *Jboss* . ==> **CXF et les EJB3 déclenchent automatiquement un équivalent de wsgen** .

2.3. Lancement mini serveur sans Container Web pour Tests

[==> OK avec jdk >=1.6 ou bien CXF .]

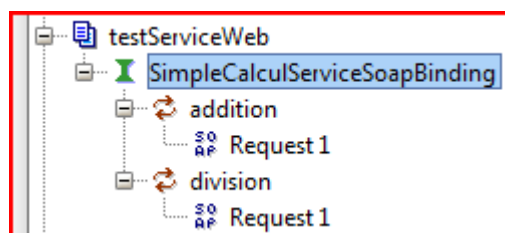
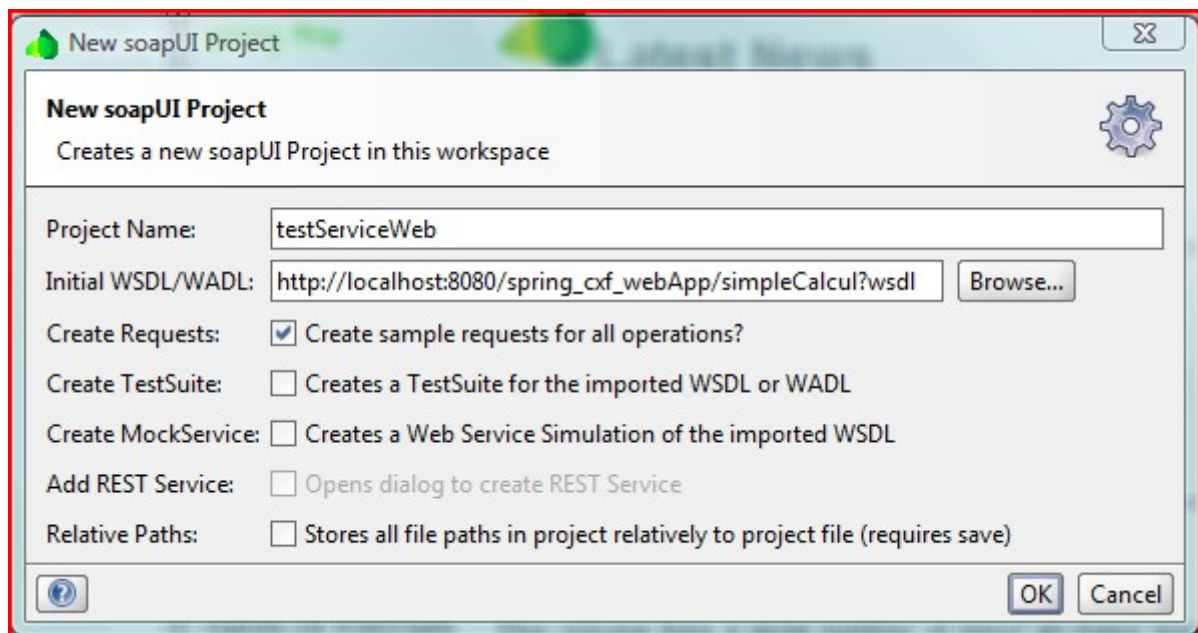
```
...
public class StandaloneServTestApp {
    protected StandaloneServTestApp() throws Exception {
        System.out.println("Starting Server");
        CalculateurImpl implementor = new CalculateurImpl();
        String address ="http://localhost:8080/myApp/services/calculateur";
        Endpoint.publish(address, implementor);
    }
    public static void main(String args[]) throws Exception {
        new StandaloneServTestApp();      System.out.println("Server ready...");
        Thread.sleep(15 * 60 * 1000);// 15 minutes
        System.out.println("Server exiting"); System.exit(0);
    }
}
```

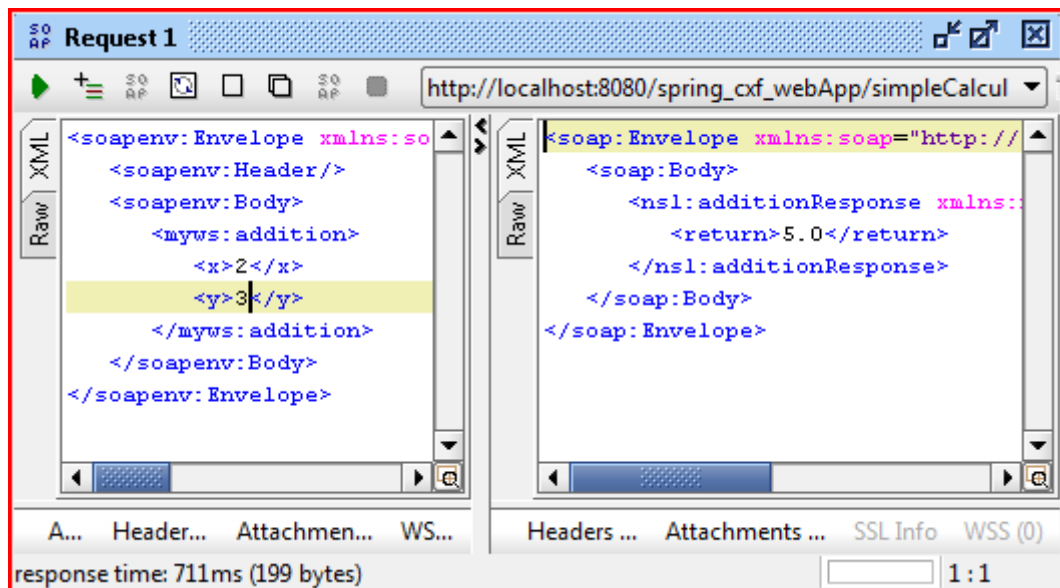
3. Tests via SOAPUI

==> pour tester rapidement un service Web , le plus simple est d'utiliser le programme utilitaire "soapui" (dont une version gratuite est disponible) .

Mode opératoire:

- Télécharger et installer soapui .
- Lancer le produit via le script ".bat" du sous répertoire bin.
- Nouveau projet (de test) avec nom à choisir.
- Préciser l'URL (se terminant généralement par *"?wsdl"*) de la description du service à invoquer.
- Sélectionner une des méthodes pour la tester en mode requête / réponse.
- Renseigner quelques valeurs au sein d'une requête.
- Déclencher l'invocation de la méthode via SOAP
- Observer la réponse (ou le message d'erreur)





Utilisation de JAX-WS coté client

3.1. Mode opératoire général (depuis WSDL)

1. générer (à partir du fichier WSDL) toutes les classes nécessaires au "*proxy JAX-WS*"
2. écrire le code client utilisant les classes générées:

```
package client;
import calcul.CalculatorSEI;
import calcul.CalculatorService;

public class MyWSClientApp {

    public static void main (String[] args) {
        try {
            CalculatorSEI calc = new CalculatorService().getCalculatorPort();

            int number1 = 10; int number2 = 20;
            System.out.printf ("Invoking addition(%d, %d)\n", number1,
                               number2);

            int result = calc.addition (number1, number2);
            System.out.printf ("The result of adding %d and %d is %d.\n\n",
                               number1, number2, result);
        } catch (Exception ex) {
            System.out.printf ("Caught Exception: %s\n", ex.getMessage ());
        }
    }
}
```

Le tout s'interprète très bien avec le **jdk >=1.6** (comportant déjà une implémentation de **JAX-WS**) ou bien avec un **jdk 1.5** augmenté de quelques librairies (ex: .jar de CXF).

3.2. Génération du proxy jax-ws avec wsimport du jdk >=1.6

```
set JAVA_HOME=C:\Prog\java\jdk\jdk1.6
cd /d "%~dp0"
"%JAVA_HOME%\bin/wsimport" [- ....] http://localhost:8080/...../serviceXY?wsdl
pause
```

sans l'option `-keep` ==> proxy au format compilé seulement

avec l'option `-keep` ==> code source du proxy également

l'option `-d` permet d'indiquer le répertoire *destination* (ou le proxy sera généré).

D'autres options existent (à lister via `-help`)

4. Redéfinition de l'URL d'un service à invoquer

4.1. Redéfinition de l'URL "SOAP"

Si l'on souhaite *invoquer un service Web localisé à une autre URL "SOAP" que celle qui est précisée dans le fichier WSDL* (ayant servi à générer le proxy d'appel) , on peut utiliser l'interface cachée *"BindingProvider"* de la façon suivante:

```

Calculateur calculateur = serviceCalculateur.getCalculateurServicePort();
javax.xml.ws.BindingProvider bp = (javax.xml.ws.BindingProvider) calculateur;
Map<String,Object> context = bp.getRequestContext();
context.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
            "http://localhost:1234/myWebApp/services/calculateur");
System.out.println(calculateur.getTva(200, 19.6));

```

4.2. Redéfinition de l'URL WSDL (et indirectement SOAP)

Il est souvent plus simple (et efficace) de redéfinir l'URL du fichier WSDL qui indirectement précise l'URL SOAP à utiliser :

exemple :

```

URL urlWsdL = new URL("http://localhost:8080/convertisseur-web/services/convertisseur?wsdl");
QName sQname = new QName("http://conversion/","ConvertisseurImplService");
ConvertisseurImplService s = new ConvertisseurImplService(urlWsdL,sQname);
Convertisseur proxyConv = s.getConvertisseurImplPort(); ....

```

NB : En mode non dynamique (basé sur WSDL et wimport) , le fichier WSDL doit absolument être accessible lors de l'exécution (sinon exception et pas de connexion)

5. Client JAX-WS sans wsimport et directement basé sur l'interface java

Dans le cas (plus ou moins rares) où les cotés "client" et "serveur" sont tous les deux en Java, il est possible de se passer de la description WSDL et l'on peut générer un client d'appel (proxy/business delegate) en se basant directement sur l'interface java du service web (*SEI : Service EndPoint Interface*).

NB: L'interface java doit au minimum comporter @WebService (et souvent @WebParam)

5.1. avec le jdk 1.6 et un accès au wsdl (sans CXF ni Spring)

```
private static void testCalculeteurServiceWithoutWsImport() {

    QName SERVICE_NAME = new QName("http://myws/", "CalculeteurService");
    QName PORT_NAME = new QName("http://myws/", "CalculeteurServicePort");

    // en précisant une URL WSDL connue et accessible
    String wdlUrl =
        "http://localhost:8080/spring_cxf_webApp/services/calculateur?wsdl";

    URL wsdlDocumentLocation=null;
    try {wsdlDocumentLocation = new URL(wdlUrl);
        } catch (MalformedURLException e) { e.printStackTrace();}

    //avec import javax.xml.ws.Service;
    Service service = Service.create(wsdlDocumentLocation, SERVICE_NAME);

    Calculeteur caculeteurWSPProxy = (Calculeteur)
        service.getPort(PORT_NAME, Calculeteur.class);

    double tauxTvaReduitPct =
        caculeteurWSPProxy.getTauxTva("TauxReduit");
    System.out.println("TauxReduit=" + tauxTvaReduitPct);
}
```

5.2. avec le jdk 1.6 et quelques ".jar" de CXF

```
private static void testCalculeteurServiceWithCxfAndWithoutWsImport() {

    QName SERVICE_NAME = new QName("http://myws/", "CalculeteurService");
    QName PORT_NAME = new QName("http://myws/", "CalculeteurServicePort");

    Service service = Service.create(SERVICE_NAME); //javax.xml.ws.Service
    // Endpoint Address
    String endpointAddress =
        "http://localhost:8080/spring_cxf_webApp/services/calculateur";

    // Add a port to the Service , javax.xml.ws.soap.SOAPBinding
    service.addPort(PORT_NAME, SOAPBinding.SOAP11HTTP_BINDING,
        endpointAddress);

    Calculeteur caculeteurWSPProxy = (Calculeteur)
        service.getPort(PORT_NAME, Calculeteur.class);

    double tauxTvaReduitPct =
        caculeteurWSPProxy.getTauxTva("TauxReduit");
    System.out.println("TauxReduit=" + tauxTvaReduitPct);
}
```

....

6. Transfert des exceptions via SOAP (Fault)

Exemple:

```
public class CalculException extends Exception {

    private static final long serialVersionUID = 1L;

    public static enum ErrorCodeEnum { UNKNOWN , DIV_BY_ZERO ,
                                      ERROR2 , ERROR3 };
    private ErrorCodeEnum errorCode = ErrorCodeEnum.UNKNOWN;
    private String details="";

    public ErrorCodeEnum getErrorCode() { return errorCode; }

    public void setErrorCode(ErrorCodeEnum errorCode) {
        this.errorCode = errorCode;
    }
    public String getDetails() { return details; }
    public void setDetails(String details) { this.details = details; }
    public CalculException() {super(); }
    public CalculException(String msg, Throwable cause) {
        super(msg, cause);
    }
    public CalculException(String msg) {super(msg);}
    public CalculException(Throwable cause) {super(cause);}
}
```

```
@WebService
public interface SimpleCalcul {
    ...
    public int division(@WebParam(name="a")int a,
                       @WebParam(name="b")int b)
        throws CalculException;
}
```

```
@WebService(endpointInterface="myws.SimpleCalcul",
             serviceName="SimpleCalculService" ,
             portName="SimpleCalculServicePort")
public class SimpleCalculImpl implements SimpleCalcul {

    public int division(int a, int b) throws CalculException {
        int res=0;
        if(b==0) {
            CalculException e = new CalculException("division par 0 interdite");
            e.setErrorCode(CalculException.ErrorCodeEnum.DIV_BY_ZERO);
            e.setDetails("a="+a+" div by b="+b);
            throw e;
        }
        else res=a/b;
        return res;
    }
}
```

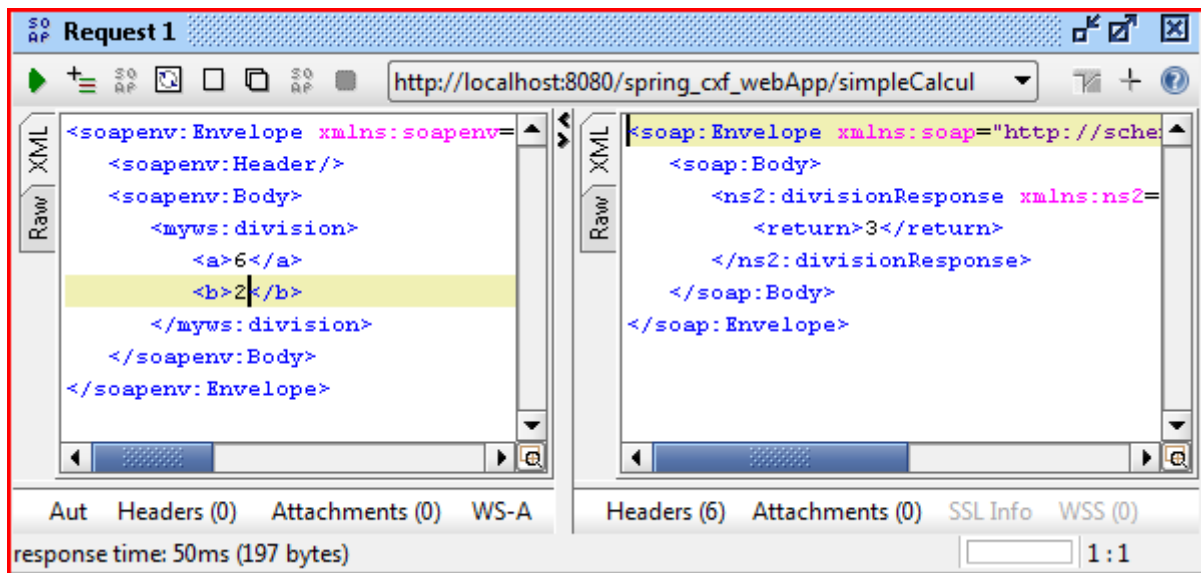
description WSDL avec Exception / Fault SOAP

```

<wsdl:definitions name="SimpleCalculService" targetNamespace="http://myws/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://myws/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <xs:schema attributeFormDefault="unqualified"
elementFormDefault="unqualified" targetNamespace="http://myws/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
      ...
      <xs:simpleType name="errorCodeEnum">
        <xs:restriction base="xs:string">
          <xs:enumeration value="UNKNOWN"/>
          <xs:enumeration value="DIV_BY_ZERO"/>
          <xs:enumeration value="ERROR2"/>
          <xs:enumeration value="ERROR3"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:element name="CalculException" type="tns:CalculException"/>
      <xs:complexType name="CalculException">
        <xs:sequence>
          <xs:element name="errorCode" nillable="true"
type="tns:errorCodeEnum"/>
          <xs:element name="details" nillable="true" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  ...
  <wsdl:message name="CalculException">
    <wsdl:part element="tns:CalculException" name="CalculException"/>
  </wsdl:message>
  <wsdl:portType name="SimpleCalcul">
    <wsdl:operation name="division">
      <wsdl:input message="tns:division" name="division"/>
      <wsdl:output message="tns:divisionResponse" name="divisionResponse"/>
      <wsdl:fault message="tns:CalculException" name="CalculException"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SimpleCalculServiceSoapBinding" type="tns:SimpleCalcul">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="division">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="division">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="divisionResponse">
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="CalculException">
        <soap:fault name="CalculException" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="SimpleCalculService">
    <wsdl:port binding="tns:SimpleCalculServiceSoapBinding"
name="SimpleCalculServicePort">
      <soap:address
location="http://localhost:8080/spring_cxf_webApp/simpleCalcul"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```

test avec b différent de zéro ==> aucune exception :



retour SOAP en cas d'erreur (b=0):

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>division par 0 interdite</faultstring>
      <detail>
        <ns1:CalculException xmlns:ns1="http://myws/">
          <errorCode xsi:type="ns2:errorCodeEnum"
            xmlns:ns2="http://myws/"
            xmlns:xsi="
              http://www.w3.org/2001/XMLSchema-instance">
            DIV_BY_ZERO</errorCode>
          <details xmlns:ns2="http://myws/">
            a=6 div by b=0</details>
        </ns1:CalculException>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

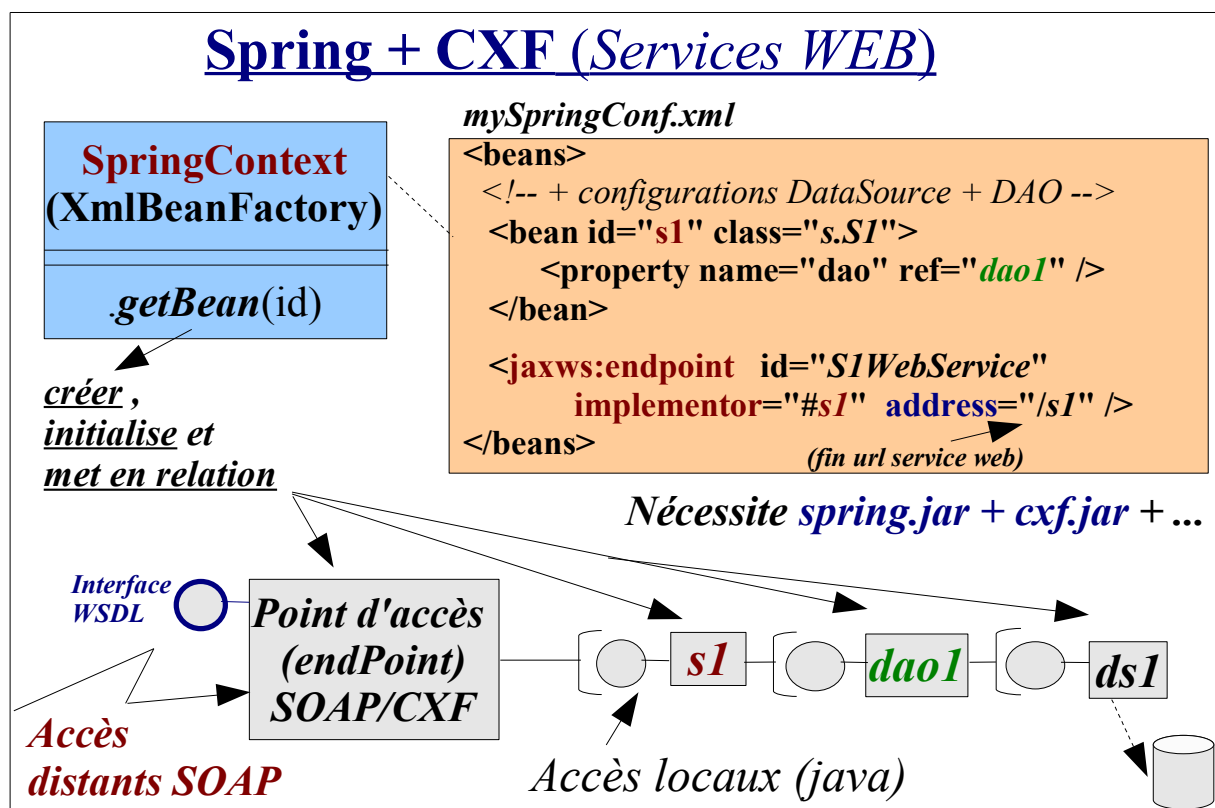
III - CXF (mise en œuvre avec Spring)

1. CXF

1.1. Présentation de CXF (apache)

- CXF est une technologie facilitant le développement et la construction de **Web Services** en se basant sur l'API **JAX-WS**.
- Le framework CXF (de la fondation Apache) permet de mettre en place des **points d'accès (SOAP)** vers les **services métiers** d'une application.
- Les points d'accès (endpoints) sont pris en charge par le servlet prédéfini "**CxfServlet**" qu'il suffit de paramétrer et d'intégrer dans une application java web (*sans nécessiter d'EJB*).

L'intégration de CXF avec Spring est simple :



- Une description WSDL du service web sera alors automatiquement générée et le service métier Spring sera non seulement accessible localement en java mais sera accessible à distance en étant vu comme un service Web que l'on peut invoquer par une URL http .
- Il faudra penser à déclarer le servlet de CXF dans WEB-INF/web.xml et placer tous les ".jar" de CXF dans WEB-INF/lib .
- Le paramétrage fin du service Web s'effectue en ajoutant les annotations standards de JAX-WS (@WebService , @WebParam , ...) dans l'interface et la classe d'implémentation du service métier .

Remarque importante: CXF (et CxfServlet) peut à la fois prendre en charge **SOAP** (via JAX-WS) et aussi **REST** (via JAX-RS) . CXF peut également s'utiliser sans Spring.

1.2. implémentation avec CXF (et Spring) au sein d'une application Web (pour Tomcat 5.5, 6 ou 7)

- Rapatrier les ".jar" de CXF au sein du répertoire **WEB-INF/lib** d'un projet Web. (ou bien configurer convenablement les dépendances "maven").
- Ajouter la configuration suivante au sein de **WEB-INF/web.xml**:

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/classes/beans.xml</param-value> <!-- ou .... -->
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
<servlet>
    <servlet-name>CXFServlet</servlet-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>CXFServlet</servlet-name>
    <url-pattern>/services/*</url-pattern> <!-- ou autre que services/* -->
</servlet-mapping>
```

- Ajouter la configuration Spring suivante dans WEB-INF ou bien dans [src ou src/main/resources ==> WEB-INF/classes] :

beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
    <import resource="classpath:META-INF/cxf/cxf.xml" />
    <import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
    <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />
    <jaxws:endpoint id="calculateurService"
        implementor="service.CalculateurService" address="/Calculateur" />
</beans>
```

variante plus pratique si plusieurs niveaux d'injections (ex: DAO , DataSource, ...):

```
<bean id="calculateurService_impl" class="service.CalculateurService" >
  <property name="...." ref="...." />
</bean>

<jaxws:endpoint
  id="calculateurService"
  implementor="#calculateurService_impl"
  address="/Calculateur" />
```

NB:

- La syntaxe est "#idComposantSpring"
- Dans l'exemple ci dessus "service.CalculateurService" correspond à la classe d'implémentation du service Web (avec @WebService) et "/Calculateur" est la partie finale de l'url menant au service web pris en charge par Spring+CXF .
- Après avoir déployé l'application et démarré le serveur d'application (ex: run as/ run on serveur),il suffit de préciser une URL du type <http://localhost:8080/myWebApp/services> pour obtenir la liste des services Web pris en charge par CXF. En suivant ensuite les liens hypertextes on arrive alors à la description WSDL .

==> c'est tout simple , modulaire via Spring et ça fonctionne avec un simple Tomcat !!!

Remarque :

En ajoutant le paramétrage bindingUri="<http://www.w3.org/2003/05/soap/bindings/HTTP/>" on peut générer un deuxième "endpoint" en version "soap 1.2" plutôt que soap 1.1 :

```
<jaxws:endpoint id="serviceCalculateur_soap12_endPoint"
  bindingUri="http://www.w3.org/2003/05/soap/bindings/HTTP/"
  implementor="#calculateurService_impl" address="/calculateur_soap12">
  <!-- version SOAP 1.2 -->
</jaxws:endpoint>
```

1.3. Client JAX-WS sans wsimport avec CXF et Spring

Rappel: Dans le cas (plus ou moins rares) où les cotés "client" et "serveur" sont tous les deux en Java , il est possible de se passer de la description WSDL et l'on peut générer un client d'appel (proxy/business delegate) en se basant directement sur l'interface java du service web (SEI : Service EndPoint Interface)

La configuration Spring suivante permet de générer automatiquement un "business_delegate" (dont l'id est ici "client" et qui est basé sur l'interface "service.Calculateur" du service Web).

Ce "business delegate" délèguera automatiquement les appels au service Web distant dont l'url est précisé par la propriété "address".

src/bean.xml

```

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://cxf.apache.org/jaxws http://cxf.apache.org/schema/jaxws.xsd">
  <bean id="clientFactory" class="org.apache.cxf.jaxws.JaxWsProxyFactoryBean">
    <property name="serviceClass" value="service.Calculateur"/> <!-- interface_SEI -->
    <property name="address" value="http://localhost:8080/serveur_cxf/Calculateur"/>
  </bean>
  <bean id="client" class="service.Calculateur" factory-bean="clientFactory"
    factory-method="create"/>
</beans>

```

autre possibilité (syntaxe plus directe) :

```

<jaxws:client id="client"
  serviceClass="service.Calculateur"
  xmlns:tp="http://service.tp/"
  serviceName="tp:CalculateurImplService"
  endpointName="tp:CalculateurServiceEndpoint"
  address="http://localhost:8080/serveur_cxf/Calculateur"/>
  <!-- avec serviceClass="package.Interface_SEI" et la possibilité d'ajouter des
intercepteurs -->

```

il ne reste alors plus qu'à utiliser ce "business_delegate" au sein du code client basé sur Spring:

```

import org.springframework.context.support.ClassPathXmlApplicationContext;
...
public class WsTestApp {

    public static void main(String[] args) {
        ClassPathXmlApplicationContext context = new
            ClassPathXmlApplicationContext(new String[] { "beans.xml" });

        Calculateur calculateur = (Calculateur) context.getBean("client");
        System.out.println(calculateur.getTva(200, 19.6));
    }
}

```

NB: il faut penser à rapatrier les ".jar" de cxf et le code source de l'interface SEI (ici service.Calculateur) .

1.4. avec CXF sans Spring

→ voir fin du chapitre "essentiel jax-ws" .

IV - E.A.I. & E.S.B. (présentation)

1. EAI et ESB

EAI, ESB

E.A.I. = Enterprise Application Integration :

Un E.A.I. est essentiellement un concept qui vise à fédérer (à un certain niveau du S.I.) la plupart des communications entre les différentes applications de l'entreprise.

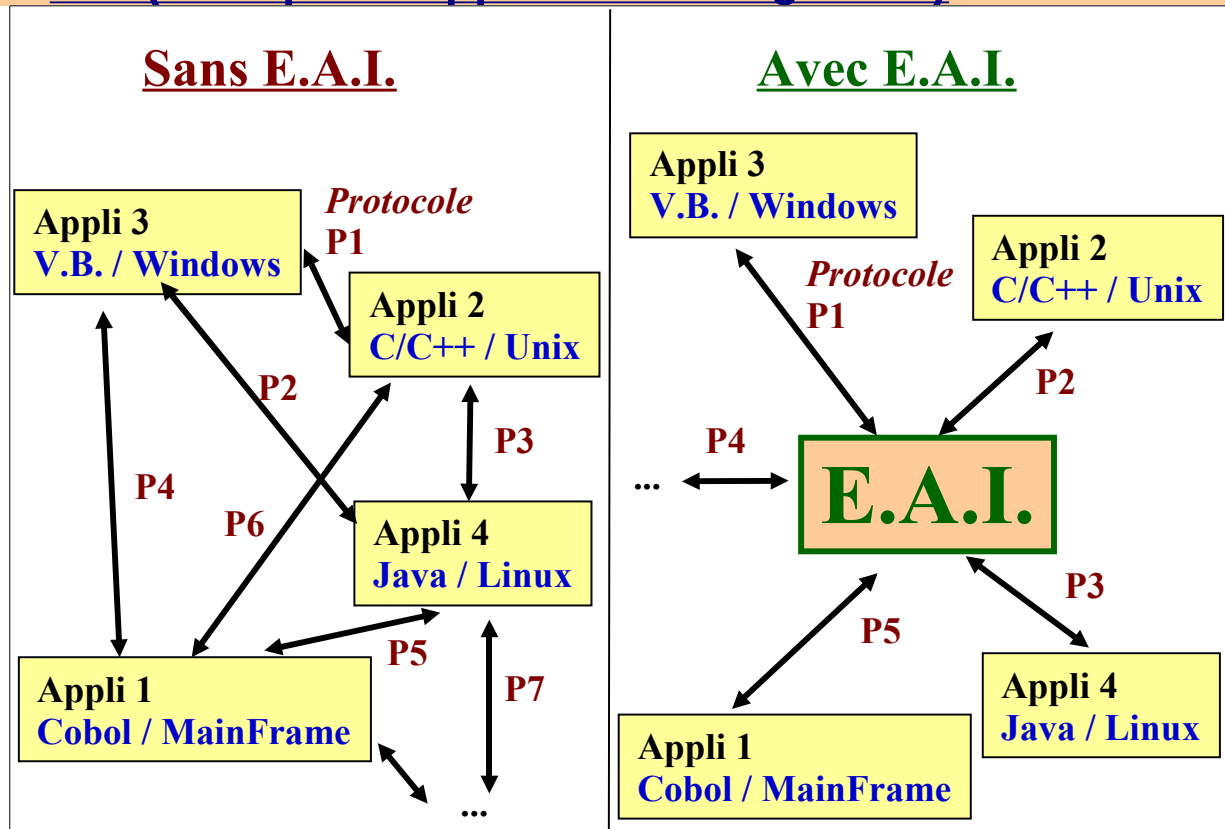
Plus de connexions directes mais un système intermédiaire logiquement centralisé qui re-route (et transforme si besoin) les messages.

E.S.B. = Enterprise Service Bus :

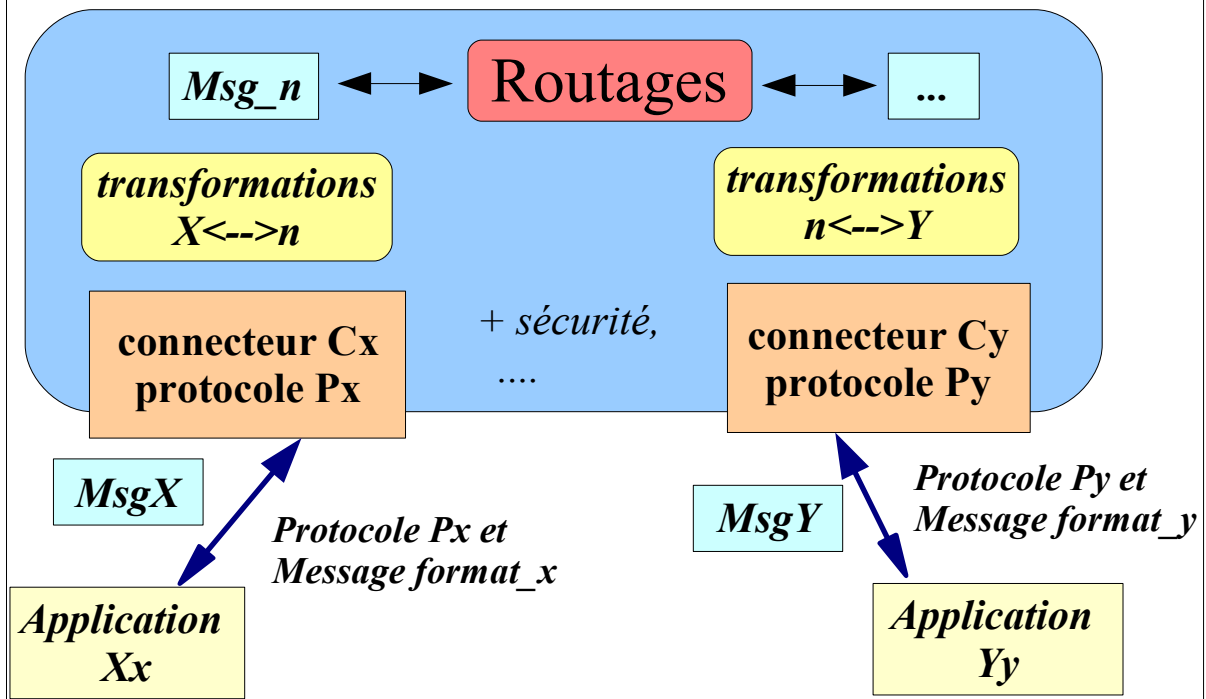
Un E.S.B. est une implémentation d'E.A.I. qui est:

- * compatible avec les technos Services Web (WSDL, SOAP)
- * structurée sous forme de BUS (interopérable avec d'autres Bus de type "ESB")

2. EAI (Enterprise Application Integration)



Fonctionnalités d'un EAI:



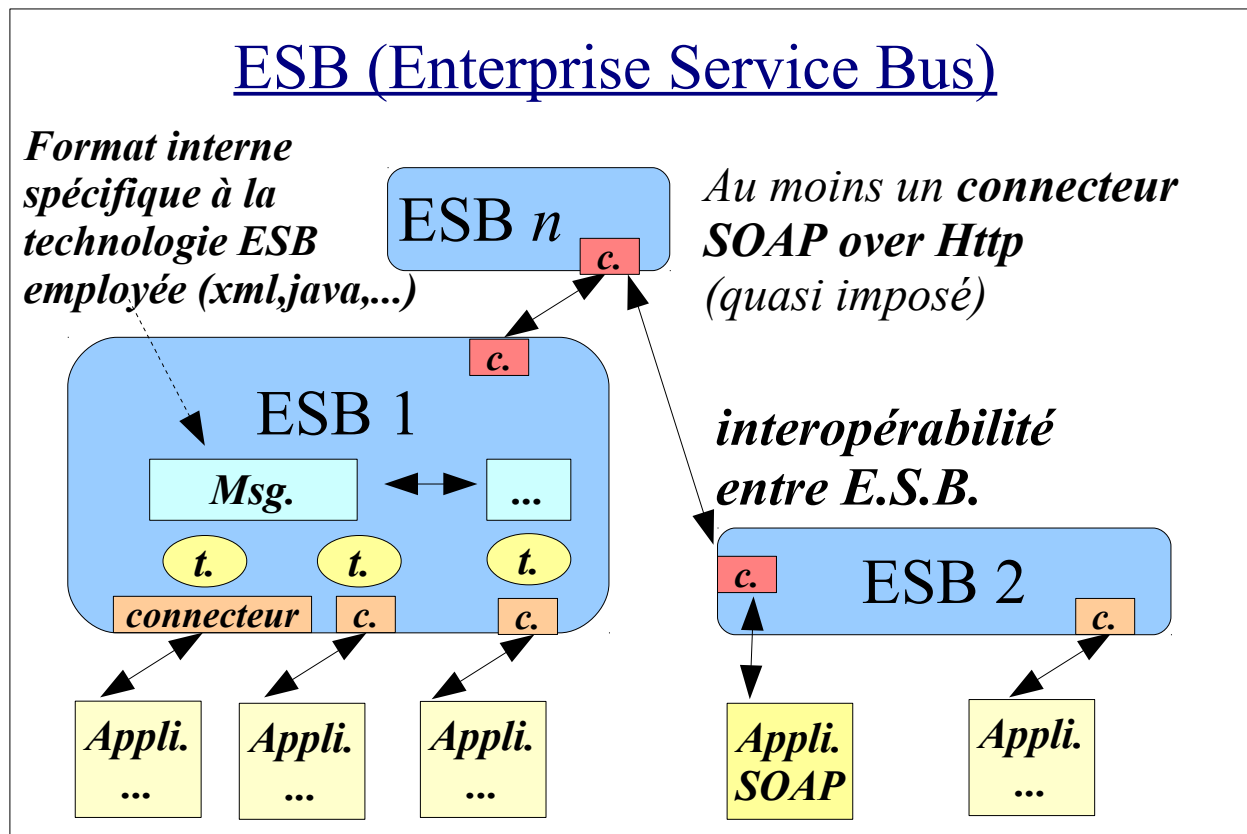
Transformations techniques internes à l'EAI/ESB

- * Si au cas par cas : potentiellement **$O(n^2)$**
transformations directes
[complexe , performant]
- * Si format interne "*neutre*" ou "*normalisé*"
 $(x <--> n <--> y)$
 $\rightarrow 2*n$ doubles transformations
[simple , moins performant]

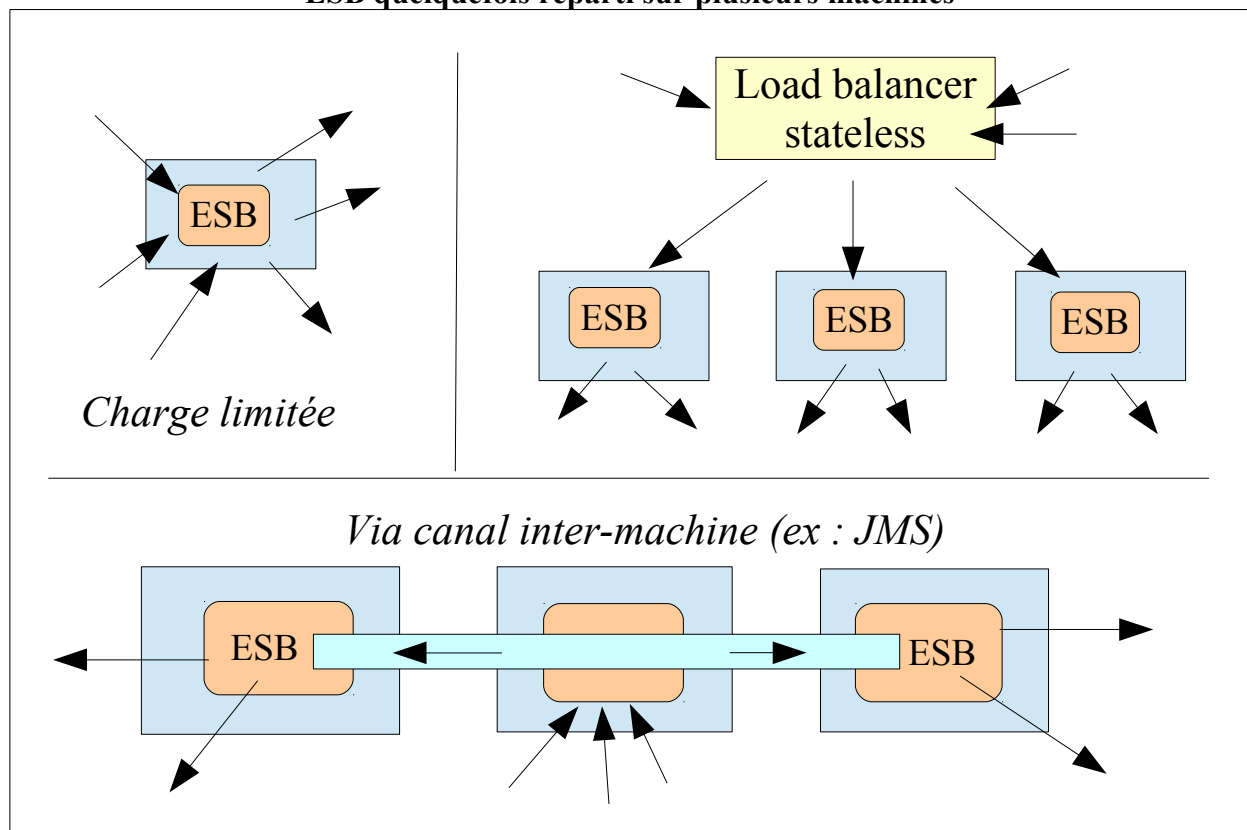
Grands traits des premiers EAI:

- Xml quelquefois utilisé pour le format des messages intermédiaires (re-transformations pratiques).
- EAI = concept
==> différentes implémentations
assez propriétaires (ex: **Tibco** , **WebMethod**, ...)
et pas directement interopérables.
- Attention aux performances (beaucoup de trafic de messages + traitements CPU liés aux transformations , ...).

3. ESB (Enterprise Service Bus)



ESB quelquefois réparti sur plusieurs machines



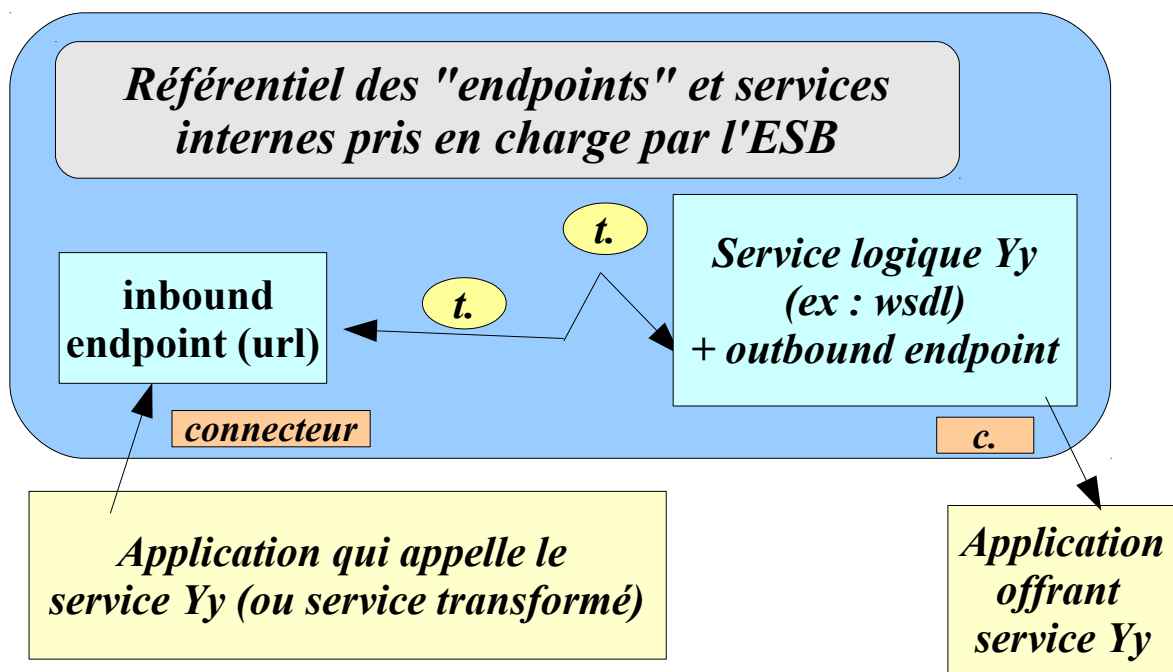
Configuration requise au sein d'un ESB

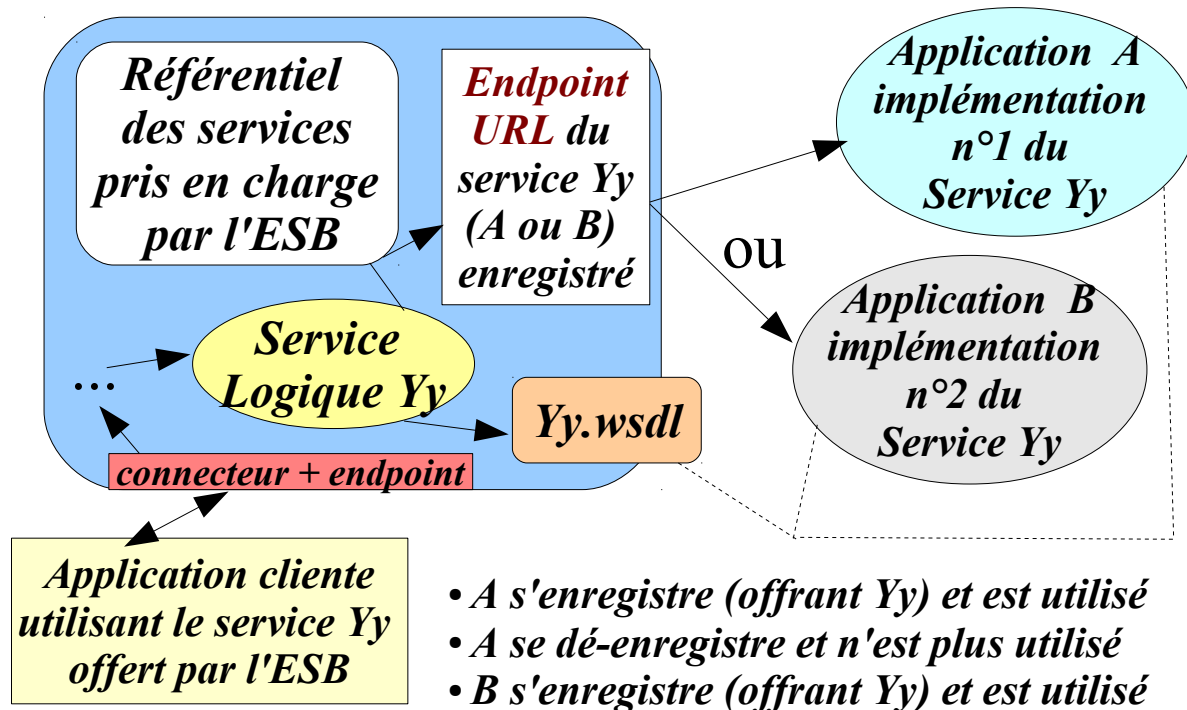
Un ESB offre généralement une infrastructure à base de connecteurs et transformateurs paramétrables .

On a généralement besoin de configurer :

- * des *points d'entrées et de sorties (endpoints)* avec des *URLs* et protocoles particuliers.
- * des *transformations* et *routages internes* au cas par cas (selon les besoins)
- * des *définitions logiques des services rendus ou accessibles* (ex : fichier **wsdl** , code java annoté , ...)

ESB: paramétrage & intégration



ESB: basculement transparent d'implémentationValeurs ajoutées couramment configurées au sein d'un ESB**Médiation de service :**

Service intermédiaire pouvant ajouter des contrôles de sécurité, des mesures statistiques, des ajustements de formats, des relocalisations transparentes, ...

Orchestrations/com combinaisons diverses:

Service interne à l'ESB qui configure plusieurs appels vers d'autres services (internes ou externes) et qui agrège/combine quelquefois les résultats.
(ex : composition , diffusion, comparaisons/filtrages, duplication/re-transfert, ...)

ESB: éléments libres & imposés (propriétaires & standards)

Tout ESB se doit de:

- * Offrir un accès interne dans un format local unifié (java, xml/soap/wsdl , ou ...) aux services (internes et externes) enregistrés.
- * Permettre à une application externe de se connecter (au moins via (SOAP, WSDL)) à un service pris en charge par l'ESB (indirectement via un connecteur)

Chaque ESB est libre de:

- * se configurer à sa façon (scripts, console , fichiers de configuration, ...).
- * de gérer à sa guise les enregistrements / déploiements de services.

3.1. Principaux ESB

Liste (non exhaustive) de quelques ESB :

ESB des grandes marques (avec prix quelquefois élevé mais un support généralement sérieux) :

<i>ESB</i>	<i>Editeur</i>	<i>Caractéristiques</i>
Tibco ESB	Tibco	Un des pionniers (anciennement EAI)
WebMethod	Software AG	Autre pionnier (anciennement EAI). Les versions récentes ont été beaucoup améliorées.
OSB (<i>Oracle Service Bus</i>)	Oracle	ESB très complet (<i>attention : pas mal de différences entre certaines versions</i>). <i>Architecture très propriétaire.</i>
WebSphere ESB	IBM	ESB très complet (utilisant norme SCA)
BizTalk	Microsoft	Lien avec norme WCF de .net
Talend ESB	Talend (éditeur ETL)	Suite SOA complète (en interne basé sur composants "open-source")

ESB "Open source" ou mixte (version "community" et version "payante") :

<i>ESB</i>	<i>Editeur</i>	<i>Caractéristiques</i>
<i>OpenESB "has been"</i>	SUN	<i>JB1</i> / intégration possible dans GlassFish V2 (mais pas V3)
<i>ServiceMix [3.x , 4.3]</i>	Apache (branche cxf)	<i>JB1</i> puis <i>OSGi+JB1</i> , <i>ODE/BPEL</i> intégrable
ServiceMix >= 4.4	Apache	OSGi (quasiment plus <i>JB1</i>) , plus d'intégration possible de <i>ODE/BPEL</i> mais intégration possible de <i>jbpm5</i> et <i>activiti</i>
<i>Petals</i>	OW2 (éditeur de Jonas)	frenchy but based on <i>JB1</i> (has been)
FuseESB	FuseSource puis reprise par Jboss/Red-Hat	Version améliorée de ServiceMix (avec support , documentation plus précise,)
Mule ESB	MuleSoft	Bon ESB assez populaire (dès le départ pas <i>JB1</i>) , relativement léger. Intégrable dans Tomcat ou ailleurs. Il existe une version payante/améliorée avec plein de connecteurs. Accompagné de l'IDE "MuleStudio" .
Synapse	Apache (branche axis2)	ESB très light basé sur HTTP
WSO2	WSO2	Basé sur OSGi (comme <i>ServiceMix</i> et <i>FuseESB</i>). Suite SOA assez complète
Spring Intégration	Spring / VmWare	Extension Spring légère (un peu comme <i>MuleESB</i> mais syntaxe Spring et plus léger)

NB : L'ancienne norme *JB1* (de SUN) est tombée à l'eau car trop complexe.

4. Notion de "moteur de services"

Utilités des "moteurs de services (java/bpel,...)"

* **transformations des formats des messages fonctionnels**

(ex : *addition(a,b)* <---> *add(x,y)* ,
chaîne_adresse <----> *adresse xml(rue + codePostal + ville)*)

techniquement , ce type de transformation peut être effectué via **XSLT** (via "interpréteur xslt") ou via **java** (avec dozer ou ...).

* **orchestration de services**

Les technologies "**BPEL**" et "**java/jBpm**" nécessitent des moteurs d'exécutions (interpréteurs) spécifiques pour faire fonctionner des services collaboratifs (*invoquant des services élémentaires selon un algorithme/processus convenu*)

Moteurs de services (exécution, intégration)

Moteur de Services BPEL

*Service n
(script BPEL
interprété)*

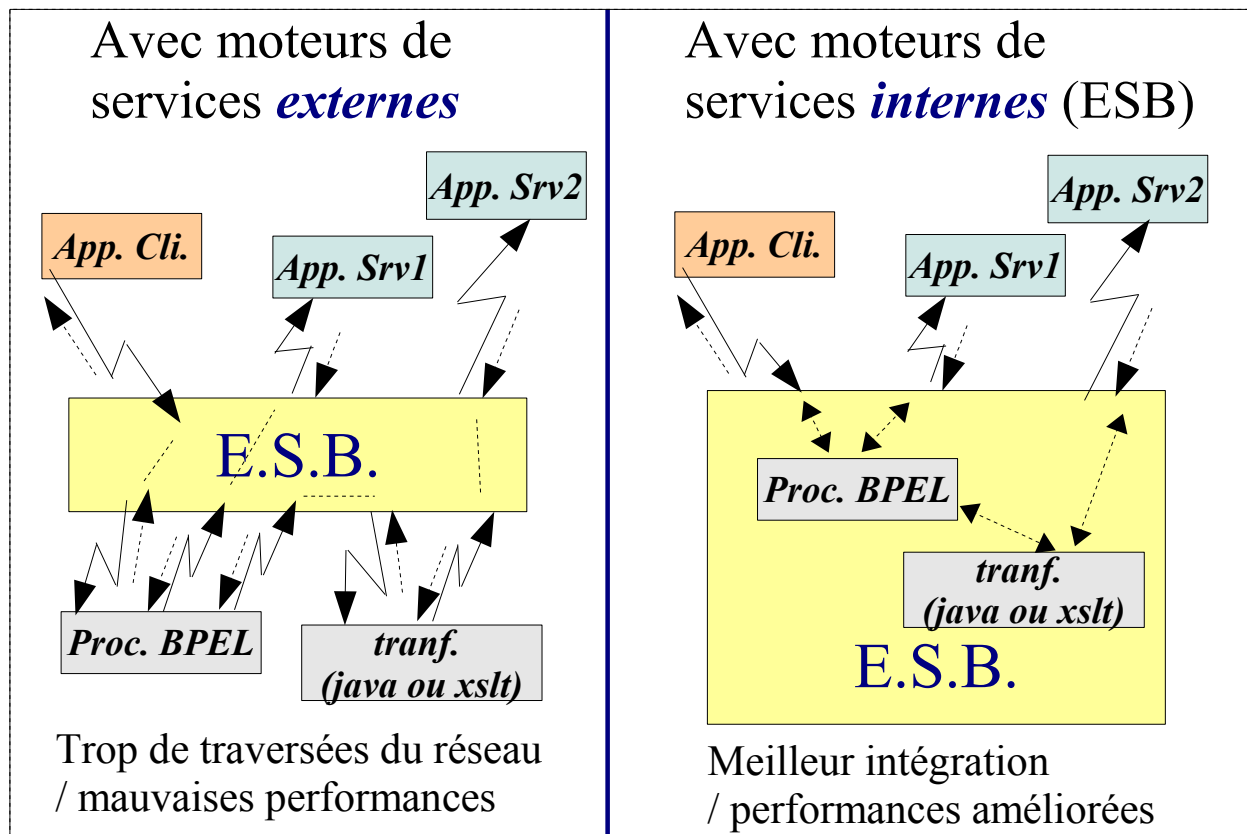
Moteur de Services Java (Serveur d'applications)

*Service Yy
(annotations jax-ws
interprétées)*

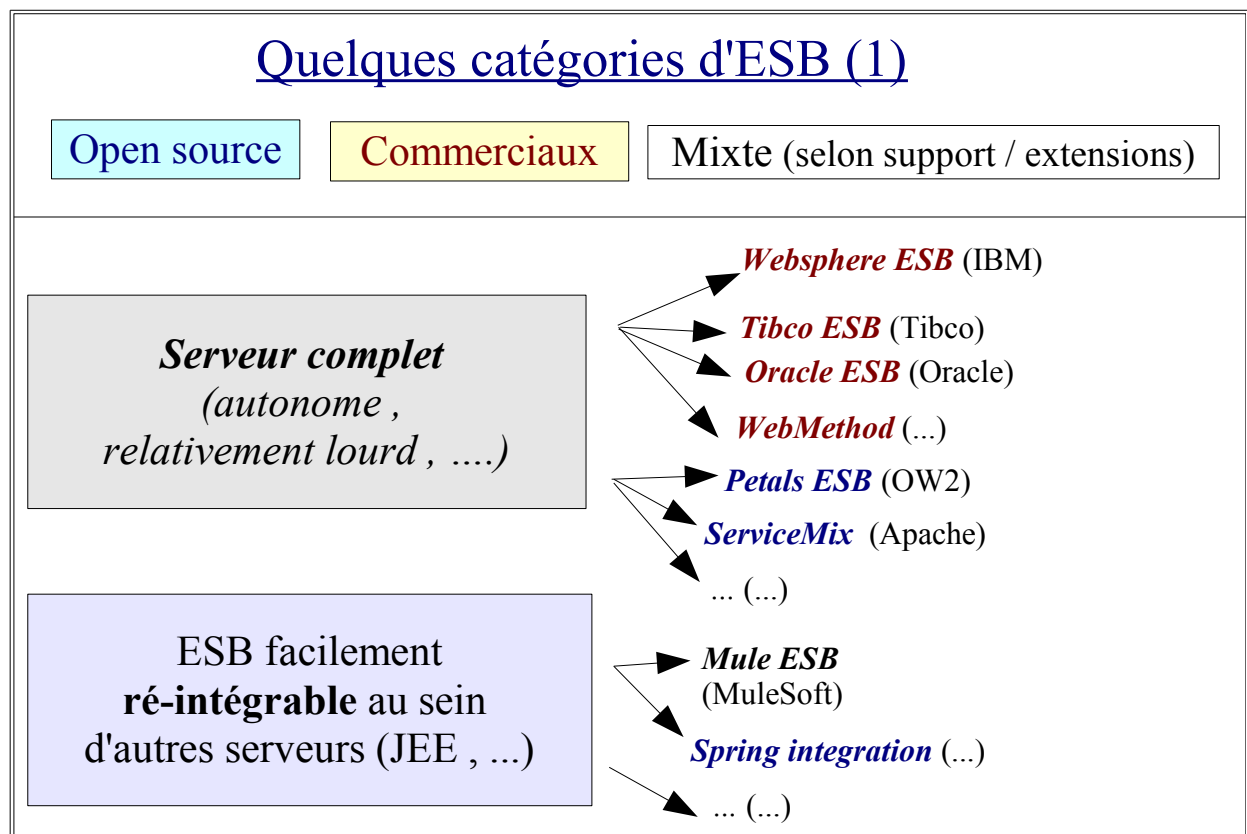
* **Définition logique
WSDL indépendante
du déploiement.**

* ***l'URL directe du point
d'accès au service est liée
au moteur de services
dans lequel le service est
déployé et interprété.***

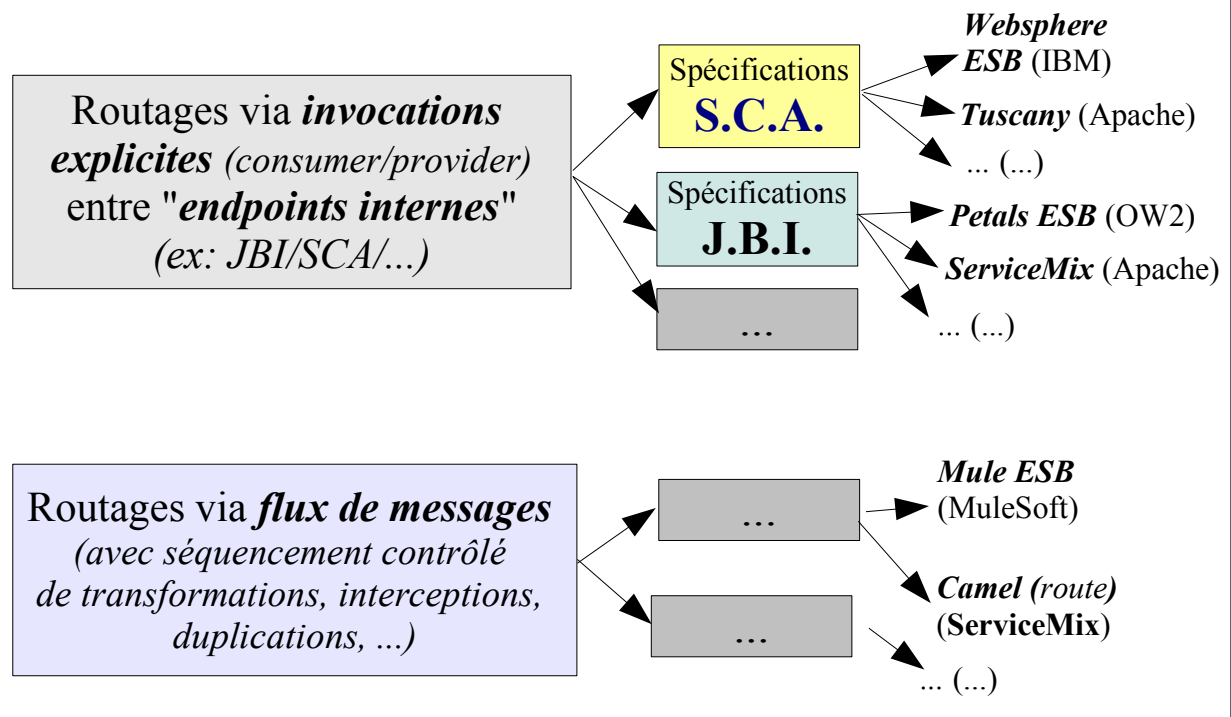
* ***Application "Moteur de
services" liée à l'ESB ?
(interne , externe ou ... ?)***



5. Catégories d'ESB

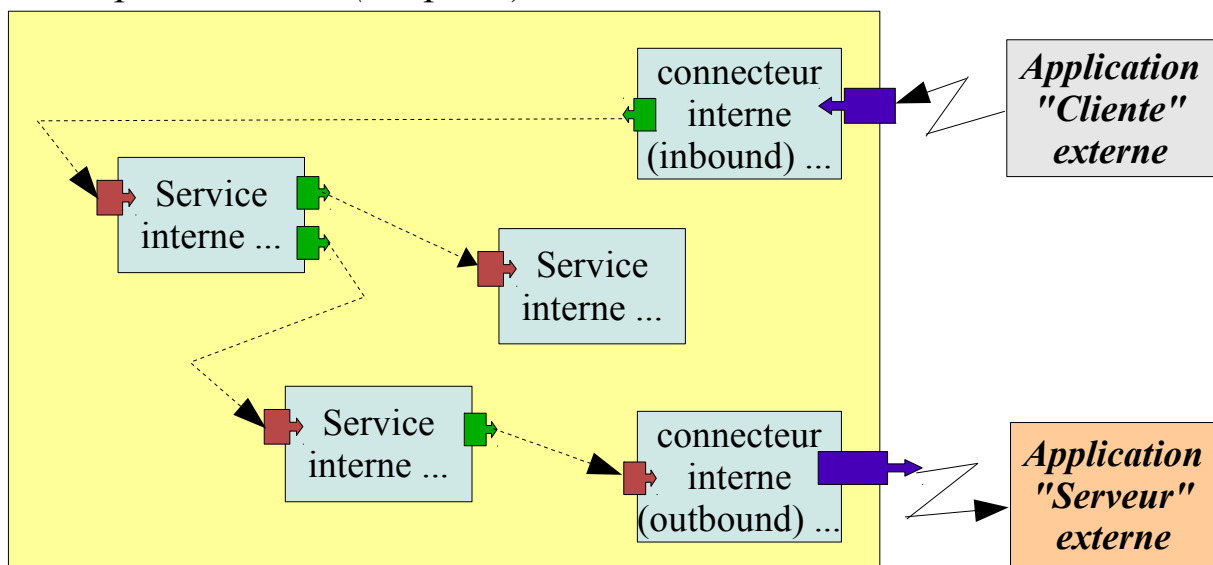


Quelques catégories d'ESB (2)



"Endpoints" internes à un ESB et connexions

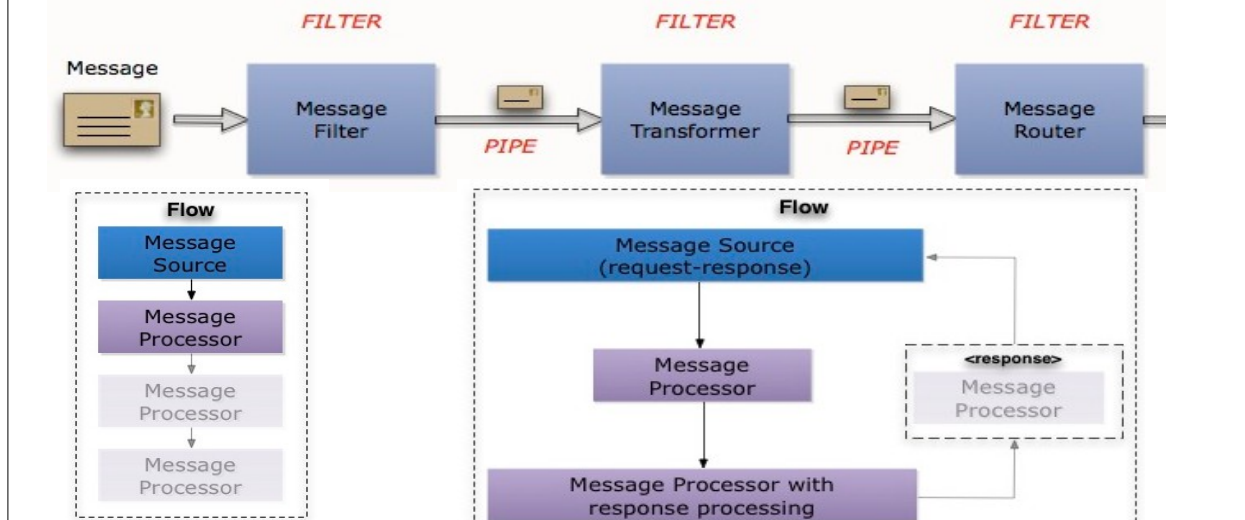
■ = point d'accès (endpoint) interne à l'esb



ESB (de type SCA, JBI ou ...)

ESB configuré(s) via des flux/flots de messages

Au lieu d'expliciter des connexions entre "endPoints internes", *certaines ESB (ex : MuleESB) se configurent en paramétrant une **suite (séquentielle) de traitements à appliquer sur des flux de messages** qui entrent dans l'ESB au niveau d'un point d'accès précis (url , ...)* .



ESB = microcosme , écosystème

*Chaque type d'ESB (SCA, JBI , Mule, ...) peut être vu comme une sorte de **microcosme (ou écosystème)** à l'intérieur duquel seront pris en charge des assemblages de services internes.*

La configuration exacte d'un service interne dépend énormément de l'ESB hôte .

D'un ESB à un autre, des fonctionnalités identiques peuvent se configurer de manières très différentes.

Attention : connecteurs quelquefois très limités et paramétrages quelquefois complexes (au sein de certains ESB) !!!!

V - Mule ESB

1. Mule Esb (présentation)

Mule ESB est un **ESB Open source** développé en **Java** par l'entreprise "... (*MuleSoft*) ...".

Cet **ESB** est volontairement basé sur une **architecture légère** (non JBI) et se configure avec des **fichiers xml** qui ressemblent à ceux de Spring.

Le concept central de Mule ESB est la notion de "**flow**" : **séquence** de *routages* , *transformations* et *services internes* .

Existant depuis longtemps et étant simple à utiliser/configurer, **Mule ESB est l'ESB Open source java le plus populaire (le plus utilisé)** .

Mule ESB peut éventuellement être intégré dans une application Java/web (et donc fonctionner au sein de Tomcat ou Jetty) .

Mule ESB peut être lancé en tant que serveur autonome (standalone) .

Mule ESB existe en deux grandes versions/variantes :

- version "**CE : Community Edition**" gratuite et sans fonctionnalité pointue , sans support
- version "**EE : Enterprise Edition**" payante avec support et fonctionnalités avancées.

Le développement d'une application "Mule ESB" peut s'effectuer avec l'IDE "**MuleStudio**" (Eclipse avec plugins spécifiques) . Bien que l'icône représente une "**tête de Mule**" , cet outil fonctionne tout de même.

Les performances de Mule ESB sont assez bonnes . **On peut "charger la mule" !!!!**

Mule ESB a dès le début fait les choix techniques suivants :

- simplicité de l'environnement technique (quelques ".jar" suffisent)
- simplicité de la configuration (fichier xml avec syntaxe proche de spring)
- optimisation des performances (avec transformations techniques au cas par cas et pas de format interne unique et lourd tel que celui de JBI).

Mule ESB peut donc être considéré comme un ESB assez "light" .

Ceci dit , la version EE (Enterprise Edition) comporte pas mal de connecteurs et l'on peut mettre en œuvre des clusters de serveurs "Mule ESB" .

2. IDE Mule Studio (utilisation)

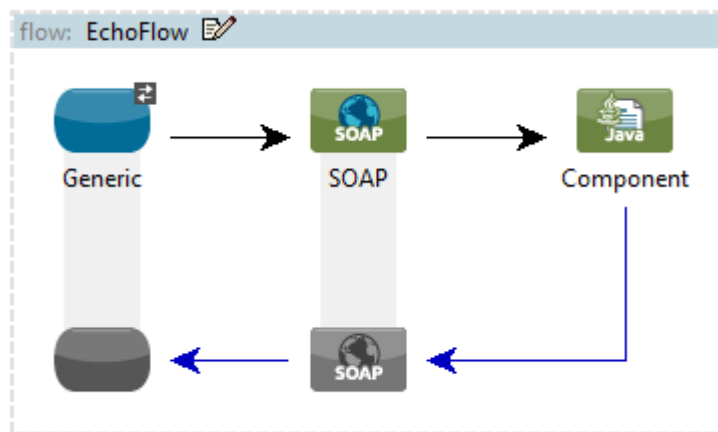
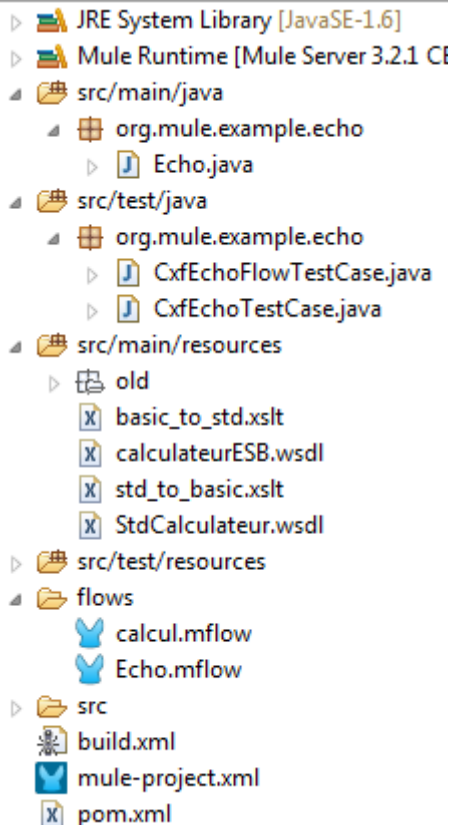
NB : La version 32bits nécessite un jdk 32bits , la version 64bits nécessite un jdk 64bits .

Après avoir installer le produit et créer un projet de type "Mule Application", on peut configurer un ou plusieurs "flow" en mode graphique ou bien en mode source_xml.

2.1. Structure d'une application mule (dans mule studio)

Respectant les conventions "maven",

les éventuelles classes java (avec @WebService ou ...) sont à placer dans **src/main/java**
 les **fichiers xml** (.xsd, .wsdl, .xslt, ...) et ".properties" sont à placer dans **src/main/resources**
 la configuration principale de mule esb s'effectue au sein de fichier(s) flowXXX.mflow.



2.2. Edition d'un "mule flow"

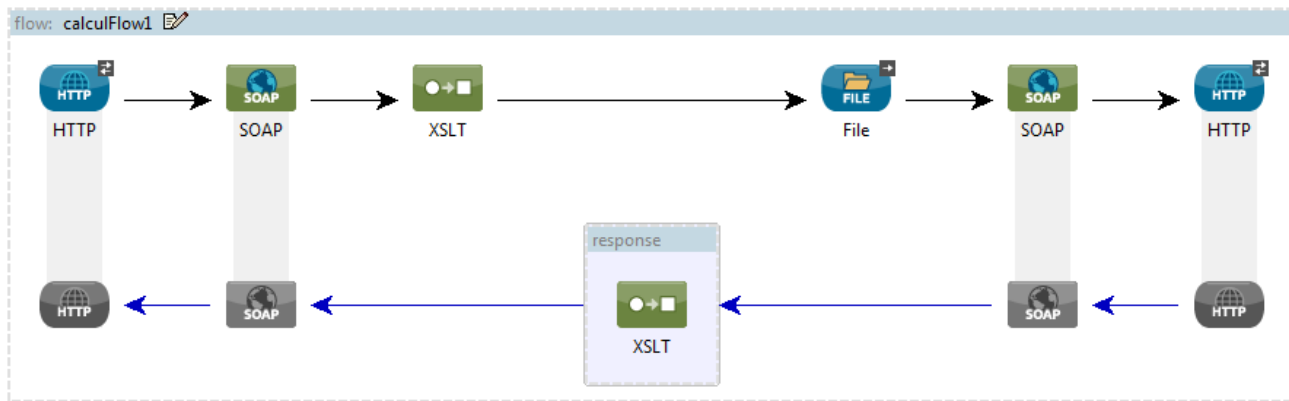
Attention, les commentaires xml (saisis en mode "source/xml") seront automatiquement supprimés par l'IDE dès que l'on basculera en mode "graphique".

Il est donc conseillé de placer des commentaires ou des versions temporaires dans des fichiers "temp1.txt" (du répertoire src/main/resources).

Mode d'édition conseillé :

- 1) effectuer des "drag & drop" depuis la palette en respectant un des ordres convenus :
 "inbound", intermédiaires internes, "outbound" (pour une traversée)
 "inbound", intermédiaires et service(s) interne(s)

- 2) effectuer quelques paramétrages en mode graphique (boîte de dialogue avec onglets)
- 3) effectuer quelques réglages fin en mode source/xml



double clic sur un élément ---> édition des propriétés dans une boîte de dialogue

Pattern Properties

SOAP

The SOAP Component will publish a SOAP web service via JAX-WS Annotations, WSDL, or CXF Simple Service using Apache CXF.

General | Advanced | Documentation

Display

Display Name: SOAP

Generic

Config Reference: [dropdown]

Operation: Proxy service

Inbound Attributes

Binding ID: [text field]

Port: 80

Namespace: http://standard/

Service: StdCalculateurService

Service Class: [text field] ... Import

☐ Validation Enabled ⓘ

Proxy-Service

Payload: body

?

OK Cancel

Edition en mode source/xml :

```

*calcul x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:mulexml="http://www.mulesoft
3 http://www.mulesoft.org/schema/mule/xml http://www.mulesoft.org/schema/mule/xml/current/n
4 http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/current
5 http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/current
6 http://www.mulesoft.org/schema/mule/cxf http://www.mulesoft.org/schema/mule/cxf/current/n
7 http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/s
8 http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/current
9 <flow name="calculFlow1" doc:name="calculFlow1">
10   <http:inbound-endpoint exchange-pattern="request-response"
11     address="http://localhost:8081/StdCalculeteurPort" doc:name="HTTP"/>
12   <cxf:proxy-service namespace="http://standard/" service="StdCalculeteurService"
13     payload="body" wsdlLocation="StdCalculeteur.wsdl" enableMuleSoapHeaders="false" c
14   </cxf:proxy-service>
15   <mulexml:xslt-transformer maxIdleTransformers="2" maxActiveTransformers="5"
16     xsl-file="std_to_basic.xslt" doc:name="XSLT"/>
17   <response>
18     <mulexml:xslt-transformer maxIdleTransformers="2" maxActiveTransformers="5"
19       xsl-file="basic_to_std.xslt" doc:name="XSLT"/>
20   </response>
21   <file:outbound-endpoint path="c:\temp\2" doc:name="File"/>
22   <cxf:proxy-client payload="body" doc:name="SOAP"/>
23   <http:outbound-endpoint exchange-pattern="request-response"
24     address="http://localhost:8080/basic-ws/services/calculateur" doc:name="HTTP"/>
25   </flow>
26 </mule>
27
<
Global Elements Configuration XML

```

2.3. Packaging , lancement et test d'une application mule .

Se placer sur un des "...mflow" et déclencher **"Run as ... / Mule application"** .

- Un environnement mule intégré à MuleStudio est automatiquement lancé pour exécuter l'application mule.
- Des erreurs , warnings ou infos s'affichent dans la console eclipse.
- Un test externe peut être rapidement effectuer avec soap-ui .
- Un clic sur l'icône rouge permet d'arrêter l'application mule .
- ...

```

Problems Javadoc Declaration Console JUnit Error Log
essai1 [Mule Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (25 avr. 2012 07:48:49)

* Agents Running:
*   JMX Agent
*****
[04-25 07:49:02] INFO  DeploymentService [main]:
+++++
+ Started app 'essai1' +

```


3. Configuration Mule Esb (essentiel)

3.1. Généralités

L'essentiel de la configuration de l'ESB Mule s'effectue au sein de fichier(s) xxx.mflow (de l'environnement de développement MuleStudio) ou bien dans le fichier **mule-config.xml**

Il y a en général un namespace xml et un schéma xsd par technologie (http,jms, ...) . L'entête du fichier xml dépend des technologies utilisées .

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
xmlns:http="http://www.mulesoft.org/schema/mule/http"
xmlns:mulexml="http://www.mulesoft.org/schema/mule/xml"
xmlns:file="http://www.mulesoft.org/schema/mule/file"
xmlns:cxfr="http://www.mulesoft.org/schema/mule/cxf"
xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
xmlns:spring="http://www.springframework.org/schema/beans"
xmlns:core="http://www.mulesoft.org/schema/mule/core"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="CE-3.2.1"
xsi:schemaLocation="
http://www.mulesoft.org/schema/mule/http
http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd
http://www.mulesoft.org/schema/mule/xml
http://www.mulesoft.org/schema/mule/xml/current/mule-xml.xsd
http://www.mulesoft.org/schema/mule/file
http://www.mulesoft.org/schema/mule/file/current/mule-file.xsd
http://www.mulesoft.org/schema/mule/cxf
http://www.mulesoft.org/schema/mule/cxf/current/mule-cxf.xsd
http://www.mulesoft.org/schema/mule/jms
http://www.mulesoft.org/schema/mule/jms/current/mule-jms.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.mulesoft.org/schema/mule/core
http://www.mulesoft.org/schema/mule/core/current/mule.xsd ">
...
<flow ...>
    <http:inbound-endpoint .../>
    <cxfr:proxy-service .../>
    <mulexml:xslt-transformer .../>
    <jms:outbound-endpoint .../>
    <file:outbound-endpoint ... />
    ...
</flow>
</mule>
```

On peut intégrer dans la configuration mule toute configuration Spring 3 valide (avec le préfixe spring: en complément) :

```
<spring:bean id/name="idCompSpring" class="package.ClasseJava" >
    <spring:property name="propriété_à_injecter" ref="id_comp_à_injecté"/>
</spring:bean>
```

NB: Certains éléments globaux (dans <mule> ...</mule> et au dehors des <flow> ...</flow>) avec des noms (identificateurs) peuvent être référencés au sein des éléments des "flow" .

Les références vers les éléments globaux sont généralement effectués via des attributs de type xxx-ref="...." .

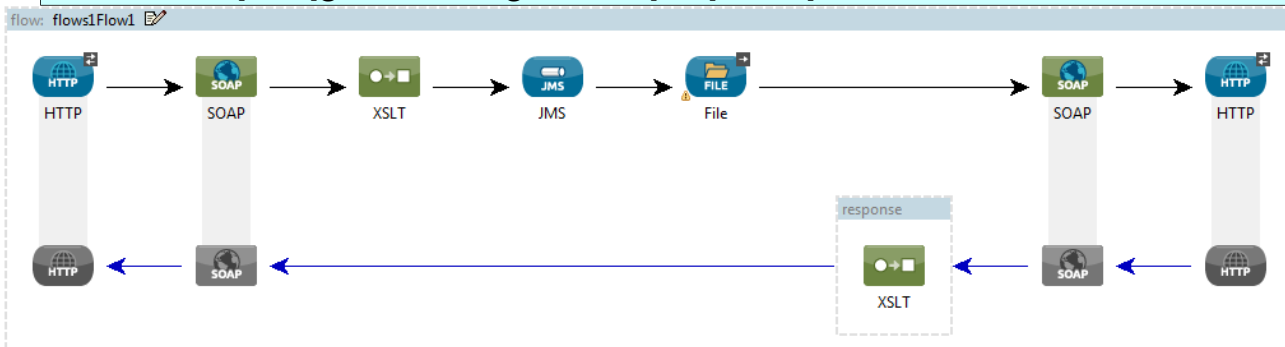
Exemple:


```
<?xml version="1.0" encoding="UTF-8"?>
<mule ...>

<jms:activemq-connector name="jmsConnector" specification="1.1"
brokerURL="tcp://localhost:61616" maxRedelivery="3" doc:name="Active MQ"/>

  <flow name="flows1Flow1" doc:name="flows1Flow1">
    ...
    <jms:outbound-endpoint queue="queue.A"
      connector-ref="jmsConnector"
      transformer-refs="objectToString objectToJms" doc:name="JMS"/>
    ...
  </flow>
</mule>
```

3.2. Exemple (grandes lignes expliquées)



```
<?xml version="1.0" encoding="UTF-8"?>
<mule ...>

<!-- connecteur vers le broker JMS de activeMQ avec URL -->
<jms:activemq-connector name="jmsConnector" specification="1.1"
  brokerURL="tcp://localhost:61616" maxRedelivery="3" />

<!-- transformateur objet vers "String" -->
<object-to-string-transformer name="objectToString" />

<!-- transformateur objet (dont String) vers message JMS -->
<jms:object-to-jmsmessage-transformer name="objectToJms" />

<!-- fin des éléments globaux qui seront référencés dans le(s) flot(s) -->

  <flow name="flows1Flow1" >
    <!-- point d'entrée HTTP sur l'ESB avec URL précisée -->
    <http:inbound-endpoint exchange-pattern="request-response"
      address="http://localhost:8081/StdCalculeurPort" />

    <!-- spécification "SOAP/WSDL" du service accessible via
      le point d'entrée précédent (SOAP-over-HTTP) -->
    <cxfr:proxy-service namespace="http://standard/"
      service="StdCalculeurService" payload="body"
      wsdlLocation="StdCalculeur.wsdl" doc:name="SOAP"/>

    <!-- transformation XSLT du contenu des "body" des requêtes SOAP -->
    <mulexml:xslt-transformer maxIdleTransformers="2" doc:name="XSLT"
      maxActiveTransformers="5" xsl-file="std_to_basic.xslt" />

    <!-- envoi d'une copie des requêtes SOAP (après transformation XSLT) -->
```

```

        dans la file d'attente "queue.A" et en précisant la double
        transformation nécessaire "objectToString objectToJms" -->
<jms:outbound-endpoint queue="queue.A" doc:name="JMS"
connector-ref="jmsConnector"
transformer-refs="objectToString objectToJms" />

<!-- envoi d'une copie des requêtes SOAP transformée dans un nouveau
fichier de nom outputPattern et dans le répertoire path -->
<file:outbound-endpoint path="c:\temp\out2" doc:name="File"
outputPattern="AdditionCall_{function:timestamp:dd-MM-yy-HH-mm-ss}.xml" />

<response>
<!-- transformation XSLT inverse (pour les réponses SOAP) -->
    <mulexml:xslt-transformer maxIdleTransformers="2" doc:name="XSLT"
        maxActiveTransformers="5" xsl-file="basic_to_std.xslt" />
</response>

<!-- spécification de la nature "SOAP" (avec enveloppe à construire
à partir du "body") du service accessible via le point de sortie
suivant -->
<cxfr:proxy-client payload="body" doc:name="SOAP"/>

<!-- point de sortie (après traversée de l'ESB) vers un service WEB
externe donc l'URL HTTP est précisée -->
<http:outbound-endpoint exchange-pattern="request-response"
address="http://localhost:8080/calculateurWS/services/Calculateur" />

</flow>
</mule>

```

NB : Le code de cet exemple nécessite les éléments annexes suivants pour fonctionner :

- un service web externe (ex : Java/Cxf) "Calculateur" avec "addition(a,b)" ... 8080
- un agent (broker) activeMQ (en attente sur le port `tcp://localhost:61616`)
- un répertoire existant "c:\temp\out2" pour accueillir les fichiers générés .
- un fichier "**StdCalculateur.wsdl**" avec "add(x,y) ..." dans src/main/resources
- deux fichiers xslt (dans src/main/resources) pour transformer add(x,y) en addition(a,b) et additionResponse en addResponse .

L'ensemble de cet exemple permet essentiellement d'appeler un service web externe existant avec des noms d'opérations basiques en "addition(a,b)" et "multiplication(a,b)" d'une manière indirecte via une nouvelle version "standard" du service sur l'ESB avec des noms d'opérations en "add(x,y)" et "mult(x,y)" . Les transformations de format sont effectuées en interne via XSLT .

Les copies envoyées en mode "one-way" via "file" et "jms" ont été ajoutées pour simplement expérimenter une combinaison de technologies au sein de l'ESB .

3.3. Configuration des points d'accès HTTP et SOAP

```

<http:inbound-endpoint exchange-pattern="request-response"
address="http://localhost:8081/StdCalculateurPort" />

<http:outbound-endpoint exchange-pattern="request-response"
address="http://localhost:8080/calculateurWS/services/Calculateur" />

```

NB : via ces 2 "endpoint" HTTP , on peut faire traverser n'importe-quelle structure de données au sein de MuleESB .

Il faut ajouter des éléments "SOAP/CXF" de façon à préciser la nature "SOAP" des requêtes et réponses qui sont attendues pour entrer et/ou sortir sur l'ESB :

```
<cx:proxy-service namespace="http://standard/"
    service="StdCalculateurService" payload="body"
    wsdlLocation="StdCalculateur.wsdl" doc:name="SOAP"/>
```

après un point d'entrée (http:inbound-endpoint)

et éventuellement

```
<cx:proxy-client payload="body" doc:name="SOAP"/>
```

avant un point de sortie (http:outbound-endpoint)

NB : il existe beaucoup d'autres configuration SOAP "cx:...." (à approfondir via la documentation de référence)

3.4. Configuration des points d'accès "FILE"

```
<!-- envoi d'une copie des requêtes SOAP transformée dans un nouveau
    fichier de nom outputPattern et dans le répertoire path -->
<file:outbound-endpoint path="c:\temp\out2" doc:name="File"
    outputPattern="AdditionCall_#[function:datestamp:dd-MM-yy-HH-mm-ss].xml" />
```

3.5. Configuration des points d'accès "JMS" (exemple ActiveMq)

```
<!-- connecteur vers le broker JMS de activeMQ avec URL -->
<jms:activemq-connector name="jmsConnector" specification="1.1"
    brokerURL="tcp://localhost:61616" maxRedelivery="3" />

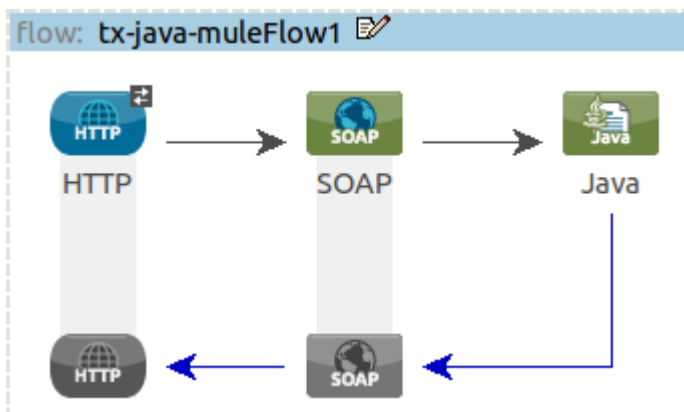
<!-- transformateur objet vers "String" -->
<object-to-string-transformer name="objectToString" />

<!-- transformateur objet (dont String) vers message JMS -->
<jms:object-to-jmsmessage-transformer name="objectToJms" />

<flow>
    ...
    <!-- envoi d'une copie des requêtes SOAP (après transformation XSLT)
        dans la file d'attente "queue.A" et en précisant la double
        transformation nécessaire "objectToString objectToJms" -->
    <jms:outbound-endpoint queue="queue.A" doc:name="JMS"
        connector-ref="jmsConnector"
        transformer-refs="objectToString objectToJms" />
    ...
</flow>
```

XML(Object) → String → TextMessage(JMS) envoyé dans file d'attente

3.6. Configuration d'une transformation fonctionnelle en java



```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<mule ...>

  <spring:beans>
    <spring:bean id="stdCalculatorBean" class="std.CalculatorBean" name="Bean">
      <spring:property name="proxyWsUrl"
        value="http://localhost:8080/wsCalculateur/services/calculateur"/>
    </spring:bean>
  </spring:beans>

  <flow name="tx-java-muleFlow1" doc:name="tx-java-muleFlow1">
    <http:inbound-endpoint exchange-pattern="request-response"
      host="localhost" port="8081" path="tx-java-mule/StdCalculateurPort"
      contentType="text/xml" mimeType="text/xml" doc:name="HTTP" />

    <!-- nb: l'attribut serviceClass de cxf:jaxws-service a une valeur correspondante
      à l'interface du WS Soap -->
    <cxf:jaxws-service doc:name="SOAP" serviceClass="std.Calculator"/>

    <!-- implementation : classe java avec @WebService -->
    <component doc:name="Java">
      <spring-object bean="stdCalculatorBean" />
      <!-- en faisant référence a un composant spring
        on peut avoir de l'injection de dépendance -->
    </component>
  </flow>
</mule>

```

Code java du composant spring (transformant multiplication en mult , etc) :

```

package std;

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

import tp.service.Calculateur;

@WebService(endpointInterface="std.Calculator")
public class CalculatorBean implements Calculator {
    private String proxyWsUrl;//url du service externe (à transformer) à injecter
    private Calculateur calcProxy = null;
    private Service service=null;

    //injection spring (config)
    public void setProxyWsUrl(String proxyWsUrl) {
        this.proxyWsUrl = proxyWsUrl;
    }
}

```

```

public void initProxy(){

    QName SERVICE_NAME = new QName("http://service.tp/", "CalculateurImplService");
    QName PORT_NAME = new QName("http://service.tp/", "CalculateurImplPort");
    service = Service.create(SERVICE_NAME);           //javax.xml.ws.Service
    // Endpoint Address
    //this.proxyWsUrl = "http://localhost:8080/wsCalculateur/services/calculateur";
    // Add a port to the Service , javax.xml.ws.soap.SOAPBinding
    service.addPort(PORT_NAME, SOAPBinding.SOAP11HTTP_BINDING,proxyWsUrl);

    calcProxy = (Calculateur) service.getPort(PORT_NAME, Calculateur.class);
}

@Override
public double add(double x,double y) {
    System.out.println("add(x="+x+",y="+y+""); //trace facultative
    if(calcProxy==null)
        initProxy();
    double res = calcProxy.addition(x,y);
    System.out.println("return res=addition(a,b)=" + res); //trace facultative
    return res;
}

@Override
public double mult(double x,double y) {
    System.out.println("mult(x="+x+",y="+y+""); //trace facultative
    if(calcProxy==null)
        initProxy();
    double res = calcProxy.multiplication(x,y);
    System.out.println("return res=multiplication(a,b)=" + res); //trace facultative
    return res;
}
}

```

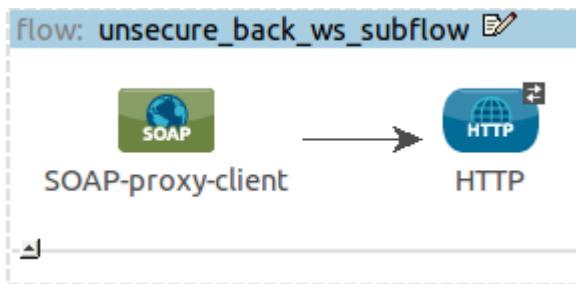
plus deux interfaces java avec @WebService pour ServiceExterne et ServiceTransformé .

3.7. Contrôle d'authentification géré via Mule-ESB

L'exemple suivant montre deux façons d'ajouter une authentification sur un service web qui initialement n'en a pas :

- soit via **basic_http**
- soit via **wss**

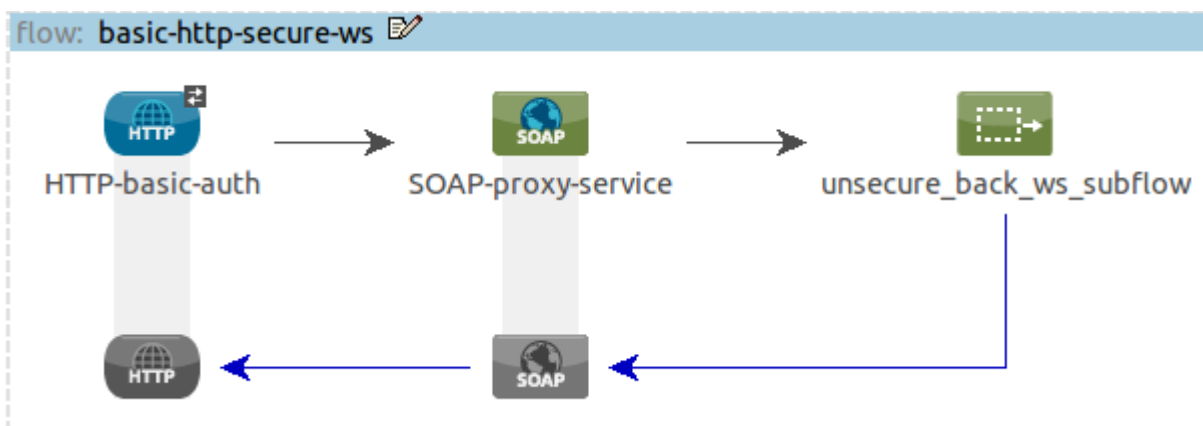
Subflow pour la partie existante non sécurisée (en arrière plan) :



```

<flow name="unsecure_back_ws_subflow" doc:name="unsecure_back_ws_subflow" >
  <cx:proxy-client payload="envelope" doc:name="SOAP-proxy-client"/>
  <http:outbound-endpoint exchange-pattern="request-response"
    address="http://localhost:8080/wsCalculateur/services/calculateur"
    doc:name="HTTP"/>
</flow>
  
```

Ajout d'authentification basique http (ok avec mule 3.4):



```

<?xml version="1.0" encoding="UTF-8"?>
<mule ...
  xmlns:mule-ss="http://www.mulesoft.org/schema/mule/spring-security"
  xmlns:ss="http://www.springframework.org/schema/security"
  xmlns:spring="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="...
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-current.xsd
    http://www.mulesoft.org/schema/mule/spring-security
    http://www.mulesoft.org/schema/mule/spring-security/3.1/mule-spring-security.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-3.1.xsd ">
  
```

<!-- NB: <mule-ss:security-manager> et <ss:authentication-manager> servent à paramétrer "memory-provider" référencé par <spring-sec:http-security-filter> de <http:inbound-endpoint> en mode basic http authentication -->

```

<mule-ss:security-manager>
  <mule-ss:delegate-security-provider name="memory-provider"
    delegate-ref="authenticationManager" />
</mule-ss:security-manager>
<spring:beans>
  
```

```

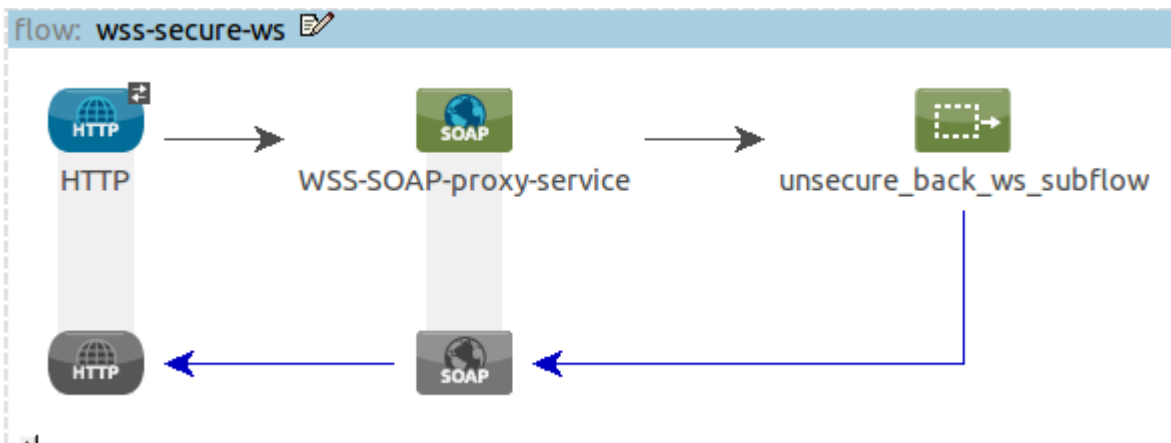
<ss:authentication-manager alias="authenticationManager">
  <ss:authentication-provider>
    <ss:user-service id="userService">
      <ss:user name="user1" password="pwd1" authorities="ROLE_ADMIN"/>
      <ss:user name="user2" password="pwd2" authorities="ROLE_ANON"/>
    </ss:user-service>
  </ss:authentication-provider>
</ss:authentication-manager>
</spring:beans>

<flow name="basic-http-secure-ws" doc:name="basic-http-secure-ws">
  <http:inbound-endpoint exchange-pattern="request-response"
    address="http://localhost:8083/secure-ws-mule/BasicHttpSecureCalculator" doc:name="HTTP-
    basic-auth" >
    <mule-ss:http-security-filter realm="mule-realm" securityProviders="memory-provider" />
    <!-- <mule-ss:http-security-filter is ok with mule 3.4 but bug with mule 3.3 -->
  </http:inbound-endpoint>

  <cxfr:proxy-service namespace="http://service.tp/" service="CalculateurImplService"
    payload="envelope" wsdlLocation="calculateur.wsdl" doc:name="SOAP-proxy-service">
  </cxfr:proxy-service>
  <flow-ref name="unsecure_back_ws_subflow" doc:name="unsecure_back_ws_subflow"/>
</flow>
</mule>

```

Ajout alternatif d'authentification "wss"



```

<?xml version="1.0" encoding="UTF-8"?>

<mule ...>

<flow name="wss-secure-ws" doc:name="wss-secure-ws">
  <http:inbound-endpoint exchange-pattern="request-response"
    address="http://localhost:8083/secure-ws-mule/WSSCalculator" doc:name="HTTP"/>

  <cxfr:proxy-service namespace="http://service.tp/" service="CalculateurImplService"
    payload="envelope" wsdlLocation="calculateur.wsdl"

```

```

doc:name="WSS-SOAP-proxy-service">
  <xf:ws-security>
    <xf:ws-config>
      <xf:property key="action" value="UsernameToken"/>
      <xf:property key="passwordCallbackClass"
        value="tp.ws_security.PasswordCallback" />
    </xf:ws-config>
  </xf:ws-security>
</xf:proxy-service>
<flow-ref name="unsecure_back_ws_subflow" doc:name="unsecure_back_ws_subflow"/>
</flow>
</mule>

```

```

package tp.ws_security;

import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.ws.security.WSPasswordCallback;

public class PasswordCallback implements CallbackHandler
{
    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException
    {
        WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];

        if (pc.getIdentifier().equals("user1"))
        {
            pc.setPassword("pwd1");
        }
        else if (pc.getIdentifier().equals("user2"))
        {
            pc.setPassword("pwd2");
        }
    }
}

```

3.8. Autres configurations

Voir la documentations de référence de Mule ESB 3 (et les exemples livrés avec la version "standalone").

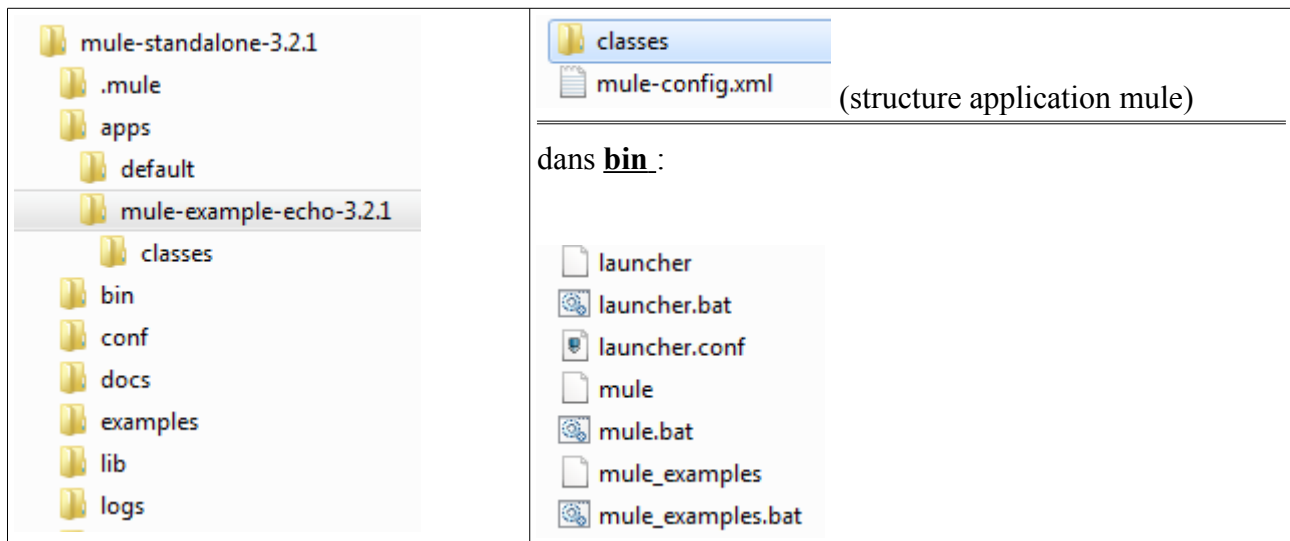
4. Environnement Mule Esb (standalone ou intégré)

Mule ESB peut (au choix) fonctionner de trois manières différentes :

- en tant que serveur autonome "**Mule ESB Standalone**"
- en tant que **sous partie d'une application java/web** (avec plein de mule-xy.jar dans WEB-INF/lib)
- en tant que **service partagé installé** avec toutes les librairies "mule-xy.jar" nécessaires *dans*

un serveur d'application (ex: Tomcat) et mis à disposition de mini "web-app" ne comportant que le code "mule" (config xml et éventuelles classes java associées)

4.1. Mule ESB "Standalone"



Le lancement/arrêt du serveur s'effectue via bin/**mule start** et bin/**mule stop** et peut être automatisé en tant que service windows ou daemon linux.

Le déploiement d'application s'effectue (à chaud) en recopiant des archives "**appliXY.zip**" dans le répertoire "**apps**".

Dès qu'une nouvelle application est détectée et reconnue/chargée sans erreur, le fichier ".zip" est supprimé et remplacé par un répertoire "appliXY" (contenu extrait du ".zip").

Au même moment un petit fichier "**appliXY-anchor.txt**" est créé et servira ultérieurement à déclencher une suppression de l'application (undeploy).

Pour enlever/supprimer une application fonctionnant dans "Mule ESB standalone", il ne faut pas supprimer directement le sous répertoire de l'application dans "apps" mais **supprimer** le fichier "**appliXY-anchor.txt**". Mule détectera cette suppression et déclenchera un "**stop + undeploy**" bien ordonné de l'application.

Le fichier "**appXY.zip**" à déployer doit comporter :

- le fichier principal "mule-config.xml" (configuration du flow) à la racine
- d'éventuels fichiers annexes (".properties", ".xml", ".wsdl", ".xsd", ".xslt")
au même niveau (racine) ou dans classes ???
- des packages et des classes **java** dans un **sous répertoire "classes"**

Pour **construire** le fichier "**appXY.zip**" à déployer, on peut (au choix) utiliser une des 2 solutions suivantes :

- déclencher le menu contextuel "**export / Mule archive**" de l'IDE spécialisé "**Mule Studio**"
- déclencher un script "**ant**" ou "**maven**" (à écrire ou ajuster).

Exemple de configuration maven (pom.xml) :

....

4.2. Mule ESB intégré dans une application Java-web

Structuration spécifique "Mule" de l'application :

- Rapatrier tous les "**mule-xy.jar**" nécessaires dans **WEB-INF/lib** (éventuellement via une liste de dépendances "maven").
- Placer les fichiers de configuration dans l'endroit habituel correspondant à la racine du "classpath" de l'application ("src" ou "src/main/resources")
- Placer les éventuelles classes java associées à la configuration "Mule" dans l'endroit habituel correspondant à la racine du "classpath" de l'application ("src" ou "src/main/java")

a des fins de tests :

```
DefaultMuleContextFactory muleContextFactory = new DefaultMuleContextFactory();

SpringXmlConfigurationBuilder configBuilder =
    new SpringXmlConfigurationBuilder("mule-config.xml");
/*SpringXmlConfigurationBuilder configBuilder =
    new SpringXmlConfigurationBuilder(new String[] { "mule-config.xml", "config2.xml" });*/

muleContext = muleContextFactory.createMuleContext(configBuilder);

muleContext.start();
```

VI - ESB "Apache ServiceMix" (OSGi , camel)

1. Présentation de l'ESB "ServiceMix"

1.1. Principales fonctionnalités et spécificités

ServiceMix est un ESB Open source codé en java .

Il est actuellement à considérer comme le deuxième Bon ESB open source java (après MuleESB qui est déjà populaire depuis quelques années).

ServiceMix est un ESB de type "serveur autonome complet" et n'est pour l'instant pas prévu pour être intégrable dans Tomcat (contrairement à Mule ESB qui est plus léger) .

1.2. Variantes

Esb "**ServiceMix**" de l'éditeur "**Apache**" = version de base (gratuite , sans support).

FuseESB → version améliorée avec support / services / documentation enrichie , ...

1.3. Evolutions

Les premières versions au point de **ServiceMix (3.x)** étaient à l'origine entièrement basées sur les spécifications **JB1** .

A partir de la **version 4** , la structure de l'ESB **ServiceMix** a été entièrement refondue sur un **cœur OSGi**. ServiceMix >=4 s'appuie en interne sur le container OSGi "**Karaf**" de la marque "Apache"

Les spécifications OSGi en version 4.2 on introduit le déploiement de bundles OSGi basés sur les spécifications "**Blueprint**" (dérivées de **Spring DynamicModule**) .

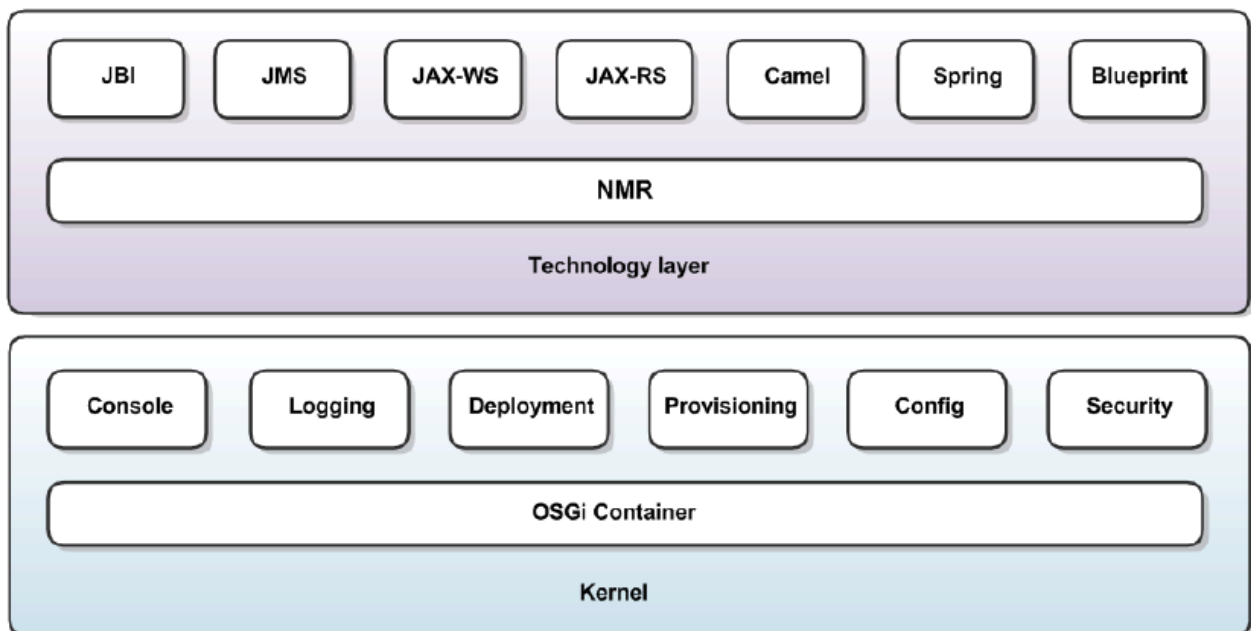
Ceci permet de déployer facilement des configurations complètes basées sur des fichiers de type "*Spring*". Une configuration "**Blueprint + cxf + camel**" pour **servicemix 4** est alors quasiment aussi **simple** qu'une configuration "MuleESB" (et bien plus simple que ce qui était auparavant attendu par la norme JBI)

Versions	Technologies	Caractéristiques
3.x	JB1 (pas osgi)	Époque où l'on croyait en l'avenir de JBI
4.1 , 4.2 , 4.3	JB1 sur Osgi ou Osgi sans JBI (ODE-BPEL intégré via JBI)	Transition "jbi" / "osgi"
4.4	Plus Osgi que JBI (ODE-BPEL pas intégré)	À fond Osgi ("jbi" considéré "has been")
5.x (attendue en 2013?)	Osgi avec support de NMR et ODE-BPEL	À priori reprise de la partie "NMR" de JBI (sans toute la lourdeur de JBI).

1.4. Arborescence de servicemix 4

	<p>Répertoire "bin" --> script ".sh" ou ".bat"</p> <p>Répertoire deploy --> où il faut déposer les "xxx_sa.zip"</p> <p>Répertoire "etc" --> configurations (ex: <i>org.ops4j.pax.logging.cfg</i> comporte le paramétrage des logs de servicemix "INFO" / "DEBUG" / ...)</p> <hr/> <p>démarrage de smx 4 ---> bin/servicemix[.bat] (en mode console/shell OSGi) ou bien bin/start[.bat] (en mode tâche de fond)</p> <p>arrêt de smx 4 ---> "osgi:shutdown" dans <i>la fenêtre (shell)</i> <i>"servicemix"</i> ou bien bin/stop[.bat] (si en mode tâche de fond)</p>
--	--

1.5. Structure de servicemix 4 (OSGi , NMR et JBI)



JBI n'est plus le cœur de servicemix 4, JBI est maintenant considéré comme une simple technologie optionnelle.

1.6. Servicemix 4 et OSGi

Principales commandes OSGi :

osgi:list affiche la *liste des bundles osgi déployés* avec leurs *numéros* et leurs *états* (install , active , ...)

```
[ 227] [Active] [Created] [ 58] camel-jetty (2.0.3)
[ 237] [Active] [Created] [ 60] blueprint-osgi-camelRoutes
<1.0.0.SNAPSHOT>
karaf@root>
```

osgi:list | grep camel

osgi:shutdown arrête le serveur OSGi (servicemix).

osgi:start / **osgi:stop** / **osgi:uninstall** *<numero_bundle>* démarre , arrête ou bien désinstalle le bundle osgi identifié par le numéro renseigné .

log:display → affiche les dernières lignes de log

log:display-exception → affiche les dernières exceptions

log:set **DEBUG** → fixe le niveau des logs à DEBUG

Depuis la version 4.4 certaines fonctionnalités optionnelles de servicemix (nmr , jbi , ...) ne sont pas installées par défaut et nécessite une installation depuis la console osgi .

features:list ---> avec états (*uninstalled* or *installed*)

features:list | grep cxf

features:install *<feature-name>*

exemples :

features:install **webconsole** (à utiliser via l'url **http://localhost:8181/system/console** (smx/smx))

features:install **nmr**

2. OSGi (présentation)

OSGi (*Open Services Gateway initiative*) est une **spécification d'environnement d'exécution java modulaire** .



OSGi est le fruit d'une **alliance** (collaboration) entre les grands éditeurs du monde java (IBM, Oracle , Jboss , ...) et est en soit une spécification indépendante de telle ou telle marque.

OSGi s'appuie sur une **JVM java** et complète celle ci en apportant une très bonne modularité via une structure globale découpée en modules (bundles) relativement indépendants .

A l'origine prévu pour les (petits) **systèmes embarqués** , OSGi est maintenant également utilisé au niveau des (gros) **serveurs d'applications JEE** .

2.1. Principaux apports de OSGi

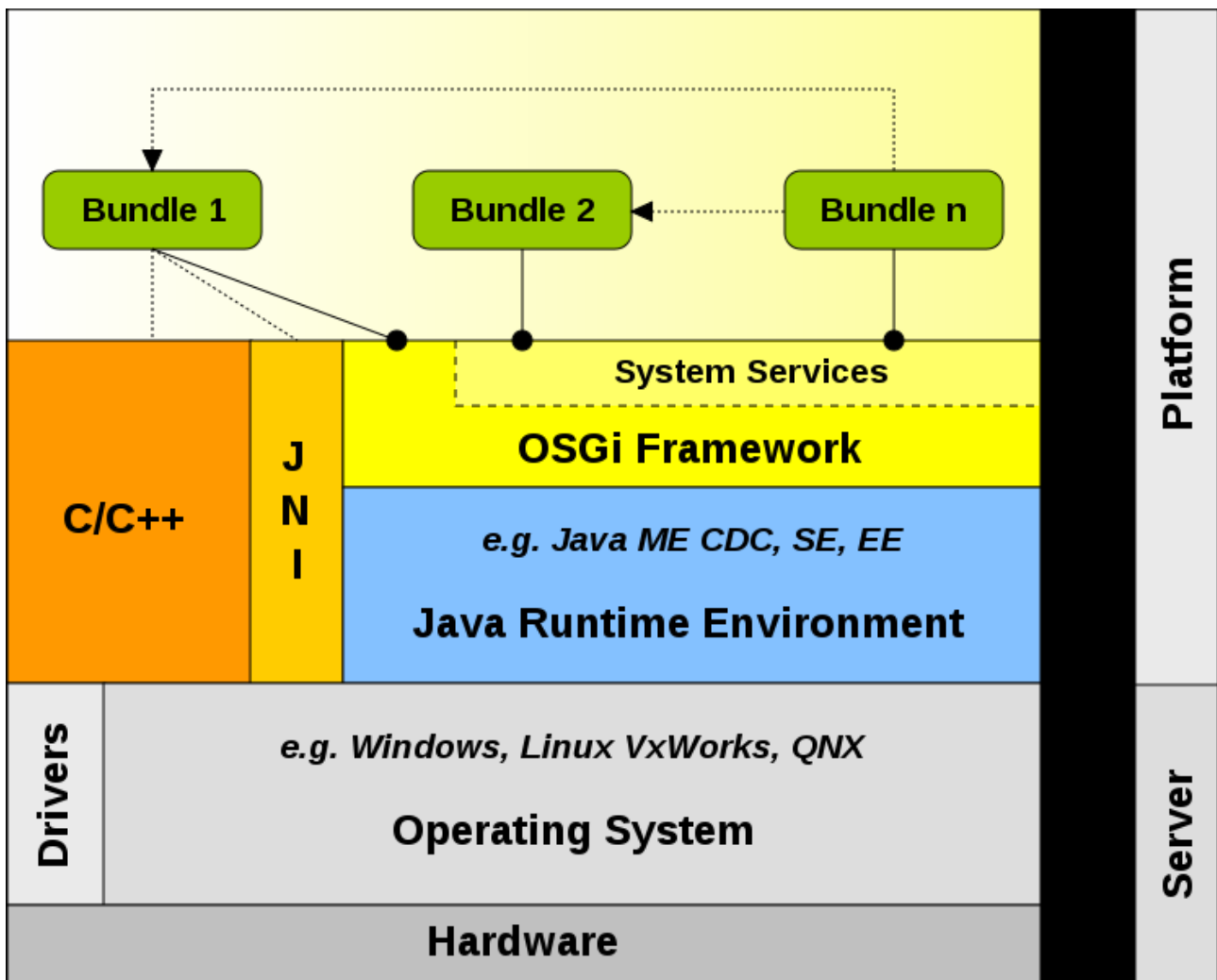
- Possibilité d'**arrêter / remplacer / redémarrer un (sous) module "à chaud"** (sans arrêter tout le processus d'exécution (serveur ou ...)).
- Possibilité de **gérer très finement les "classpath" de chacun des modules en permettant des réutilisations partielles contrôlées (via import/export)** .
==> Ceci permet **globalement de très bien optimiser la gestion de la mémoire** .
- Possibilité de faire cohabiter plusieurs versions d'une même librairie .
- **Mise en relation automatique (selon le modèle publication/souscription) des services offerts par un module avec les besoins des autres modules.**

2.2. grandes lignes des spécifications osgi

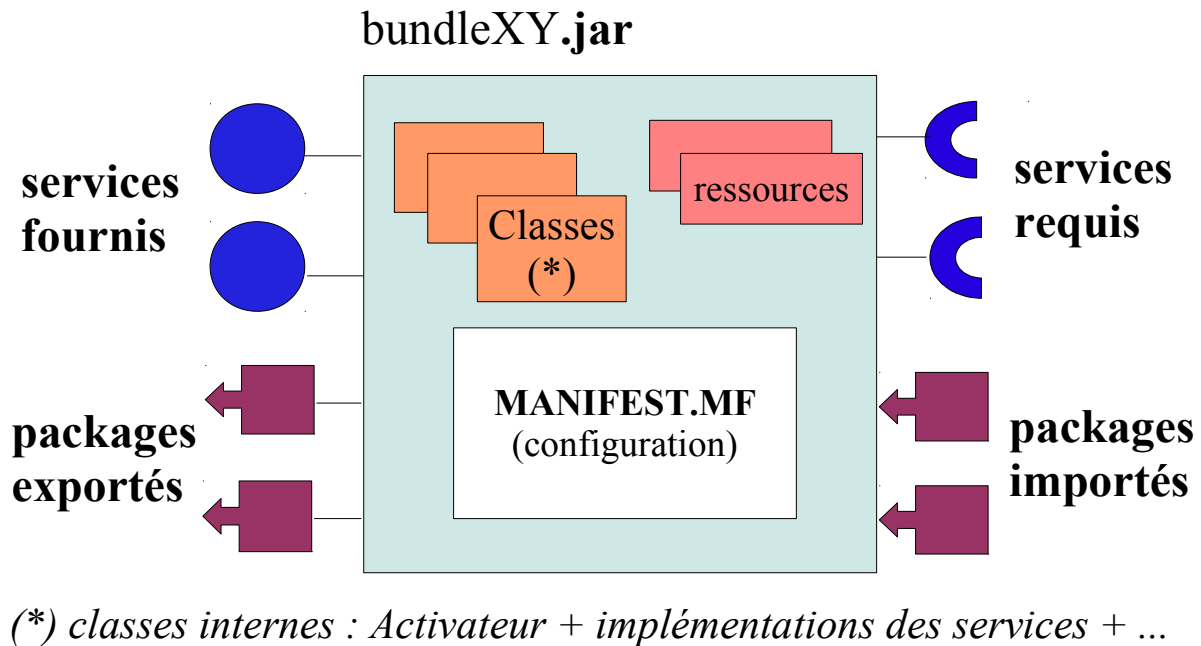
Modèle en couches :

- services (publication/souscription)
- cycle de vie (activator)
- modules/bundles avec "classloader" (avec import/export des packages)
- JVM

JVM découpée en bundles relativement autonomes



Anatomie d'un bundle OSGi



2.3. Principales implémentations et utilisations

Principales implémentations de OSGi :

<i>Implémentation de OSGi</i>	<i>Marque/Editeur</i>	<i>Caractéristiques</i>
Felix	Apache	
Equinox	Eclipse	

Liste (non exhaustive) de produits qui utilisent OSGi :

- Jboss AS 7
- Glassfish 3
- ServiceMix 4
-

3. Spring Dynamic Module et OSGi Blueprint

3.1. Fonctionnalités de Spring Dynamic Module

"**Spring DM**" (*Spring Dynamic Module*) est une **variation** du **framework Spring** spécialement **adaptée** à la mise en œuvre de **bundle OSGi** .

Le framework Spring habituel est dédié aux applications JEE (fonctionnant sous Tomcat ou dans d'autres serveurs d'applications JEE).

Spring DM est une **variante allégée** (*basée sur le cœur de Spring-Framework 2.5*) et qui peut fonctionner au sein d'un **container OSGi** de type "*karaf*" ou "*equinox*" .

A l'aide de "**Spring DM**" on peut alors facilement concevoir un **bundle applicatif OSGi** bâti à partir de **composants reliés entre eux par des injections de dépendances** et configuré via un **fichier de configuration xml** à la syntaxe spring habituelle (`<beans></beans>`).

Fonctionnalités "**Spring DM**" spécifiques à l'environnement **OSGi** :

- import/export et injection de dépendances vis à vis d'autres modules OSGi
- ...

3.2. Variante officielle "OSGi Blueprint" de OSGi 4.2

Bien que fonctionnant très bien sous servicemix 4 , Spring DM n'est pas une composante officielle des spécifications OSGi 4.2 .

D'autre part , le projet "Spring DM" est officiellement arrêté en version 1.0 .

"Gemini blueprint" (de la fondation eclipse) est le successeur annoncé de Spring DM.

Si l'on tient à se conformer au **standard "OSGi 4.2"** , on peut éventuellement utiliser une **variante de Spring DM normalisée par les spécifications OSGi** qui s'intitule "**OSGi Blueprint**" .

Les fonctionnalités essentielles sont à peu près les mêmes.

Quelques extensions non normalisées de Spring DM ne sont pas reprises par Blueprint.

La **principale différence** tient dans les "**namespaces xml**" (en entête des fichiers de configuration).

Les versions récentes de servicemix (ex : 4.4) supportent aussi bien les bundles "**Spring DM**" que les bundles "**blueprint**" .

L'avenir nous dira peut être quelle sera la version la plus utilisée (standard de fait qui fonctionne ou bien norme officielle suivie ou pas ?) .

3.3. Structure et configuration d'un bundle "Spring DM"

Arborescence (**maven**) pour **Spring-DM**

```

src
|
main
|
java
|   ....java
resources
|   ...xml , .xslt , ...
META-INF
|   spring
|       beans.xml

```

Exemple de fichier de configuration "Spring DM" :

```

META-INF/spring/beans.xml
<?xml version="1.0" encoding="UTF-8"?>

<!-- META-INF/spring/*.xml for Spring DM (Dynamic Module) for OSGi -->

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxws="http://cxf.apache.org/jaxws"
  xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

  <jaxws:endpoint id="calculateurEndpoint"
    implementor="tp.service.CalculateurImpl"
    address="/calculateur"/>
  <!-- url in smx 4.4.2 : http://localhost:8181/cxf/calculateur?wsdl -->
</beans>

```

Exemple de configuration maven pour un bundle "Spring DM" dans serviceMix :

.../...

pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd"> <modelVersion>4.0.0</modelVersion>
  <parent> ... </parent>
  <groupId>tp</groupId>
  <artifactId>springDM-osgi-app1</artifactId>
  <version>1.0-SNAPSHOT</version>

  <packaging>bundle</packaging>

  <dependencies>
    <dependency>
      <groupId>org.apache.geronimo.specs</groupId>
      <artifactId>geronimo-ws-metadata_2.0_spec</artifactId>
      <version>1.1.3</version>
    </dependency>
    <!-- <dependency>
      <groupId>org.apache.servicemix</groupId>
      <artifactId>servicemix-utils</artifactId>
      <version>1.5.0</version>
    </dependency> -->
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions> <!-- important pour nouveau packaging=bundle -->
        <version>2.3.7</version>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
            <Bundle-Description>${project.description}</Bundle-Description>
            <b>Import-Package</b>
              javax.jws,
              javax.wsdl,
              javax.xml.namespace,
              org.springframework.beans.factory.config
            </b>
            <!-- org.apache.servicemix.util, not imported -->
            <b>Private-Package</b>tp.service</b>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>

```

3.4. Structure et configuration d'un bundle "Blueprint"

Arborescence (**maven**) pour **Blueprint**

```

src
|
main
|
java
...java
resources
|
...xml , .xslt , ...
OSGI-INF
|
blueprint
|
blueprint.xml

```

Exemple de fichier de configuration "Blueprint" :

META-INF/blueprint/blueprint.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- OSGI-INF/blueprint/*.xml for blueprint (standardized version of Spring DM in OSGi R4.2)
      Aries est un projet apache de type implémentation du container blueprint osgi -->

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:cxf="http://cxf.apache.org/blueprint/core"
  xmlns:camel-cxf="http://camel.apache.org/schema/blueprint/cxf"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxws="http://cxf.apache.org/blueprint/jaxws"
  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
http://cxf.apache.org/blueprint/jaxws http://cxf.apache.org/schemas/blueprint/jaxws.xsd
http://cxf.apache.org/blueprint/core http://cxf.apache.org/schemas/blueprint/core.xsd
http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/camel-
blueprint.xsd">

  <cxf:bus id="calculateurServiceBus"> </cxf:bus>
  <bean id="calculateurImpl" class="tp.service.CalculateurImpl" />

  <jaxws:endpoint id="calculateurEndpoint" implementor="#calculateurImpl"
    address="/calculateurWS">
  </jaxws:endpoint>
  <!-- url in smx 4.4.2: http://localhost:8181/cxf/calculateurWS?wsdl -->

```

`</blueprint>`

Autres exemples de namespaces et schémas xsd en entête :

```

<!-- xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0"
xmlns:jaxrs="http://cxf.apache.org/blueprint/jaxrs"
xsi:schemaLocation="...
http://cxf.apache.org/blueprint/jaxrs http://cxf.apache.org/schemas/blueprint/jaxrs.xsd
http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0
http://aries.apache.org/schemas/blueprint-cm/blueprint-cm-1.1.0.xsd"

<!-- <cm:property-placeholder persistent-id="xyz"
update-strategy="reload">
</cm:property-placeholder>

```

Exemple de configuration maven pour un bundle "blueprint" dans serviceMix :

pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd"> <modelVersion>4.0.0</modelVersion>
  <parent>... </parent>
  <groupId>tp</groupId>
  <artifactId>blueprint-osgi-app1</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>bundle</packaging>

  <dependencies>
    <!-- <dependency>
      <groupId>org.osgi</groupId>
      <artifactId>org.osgi</artifactId>
      <version>3.0.0</version>
    </dependency>
    <dependency>
      <groupId>org.osgi</groupId>
      <artifactId>org.osgi.enterprise</artifactId>
      <version>4.2.0</version>
    </dependency> -->
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions> <!-- important pour nouveau packaging=bundle -->
        <version>2.3.7</version>
        <configuration>
          <instructions>

```

```

    <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
    <Bundle-Description>${project.description}</Bundle-Description>
    <Import-Package>
        javax.jws,
        javax.wsdl,
        javax.xml.namespace,
        org.osgi.service.blueprint
    </Import-Package>
    <Private-Package>tp.service</Private-Package>
</instructions>
</configuration>
</plugin>
</plugins>
</build>

</project>

```

4. Camel

4.1. Présentation de camel

Camel est une **technologie d'intégration** *java* et *open source* de l'éditeur "Apache" .

Camel apporte les **principales fonctionnalités suivantes** :

- **intégration** des principales **technologies asynchrones** (*JMS* , SMTP , File ,)
- intégration de quelques technologies synchrones (Http, SOAP via cxf , ...)
- **design patterns d'intégration "EIP"** (filtrage , composition et enrichissement de message, routage conditionné , interception , logs , ...)
- **gestion** simple (syntaxiquement compacte) des **routes** (pour les *messages* véhiculés)
- **configuration "java DSL"** et/ou "xml proche spring" .

En d'autres termes , camel permet de facilement "intégrer" et "piloter" tout un tas de technologies qui tournent autour de la gestion des messages dans un environnement middleware .

Camel est souvent associé à *ActiveMQ* (middleware JMS) ou à l'ESB *ServiceMix* .

4.2. Intégration dans l'ESB ServiceMix

Camel constitue une des pièces maîtresses de l'ESB ServiceMix.

Dans une intégration JBI (smx 3 ou 4) , camel peut être utilisé pour configurer des routes entre des "endpoint jbi" d'autres technologies (ex : router une invocation BPEL vers un endpoint JMS/JBI) .

Dans une intégration OSGi (smx >= 4) , camel peut être utilisé en intégrant directement d'autres technologies compatibles OSGi (ex : cxf via camel ,) sans passer par JBI .

4.3. Syntaxes "DSL" et "XML" (configuration)

Configuration "xml" (à intégrer selon container "jbi" ou "spring DM" ou "blueprint" ou ...) :

```
...
<camelContext xmlns="http://camel.apache.org/schema/blueprint">

  <route>
    <from uri="cxf:bean:EsbCalculateurPort" />
    <convertBodyTo type="java.lang.String"/>
    <to uri="xslt:tx1.xslt" />
    <wireTap uri="file://target/inbox/" />
    <to uri="log:calculateurTapInputAfterXslt" />
    <to uri="http://localhost:8181/cxf/calculateurWS?bridgeEndpoint=true"/>
    <convertBodyTo type="java.lang.String"/>
    <to uri="log:calculateurTapResponse" /> <!-- automatic wireTap = effect of
                                          InOut / InOnly MEP -->
  </route>

  <route>
    <from uri="timer:test" />
```

```

    <wireTap uri="log:tap"/>
    <to uri="log:test" />
  </route>
</camelContext>
...

```

Exemple de syntaxe "Java DSL" (Domain Specific Language) :

```

CamelContext context = new DefaultCamelContext();

context.addRoutes(new RouteBuilder() {
    public void configure() {
        from("test-jms:queue:test.queue")
            .to("file://test");
    }
});

```

La syntaxe "Java DSL" permet d'encoder des paramétrage de "routes" dans du code java avec pour avantage la possibilité d'effectuer simplement des *routages conditionnés* (via des "if").

Autre avantage potentiel de "camel java DSL": héritage de configuration (et variantes) .

4.4. Utilisation de camel pour configurer des routes de JBI

Pendant la période de transition "JBI → OSGi", on peut utiliser camel pour configurer simplement des routes entre des "endpoints JBI existants" (ex : smx-http , smx-file , ODE/BPEL , ...)

Exemple :

pom.xml

```

...
<artifactId>basic-ws-camel-su</artifactId>
<packaging>jbi-service-unit</packaging>
<properties>
    <componentName>servicemix-camel</componentName> <!-- pour jbi.xml su sa -->
</properties>
...

```

src/main/resources/camel-context.xml

```

<?xml version="1.0" encoding="UTF-8"?>

<!-- ce fichier doit s'appeler camel-context.xml (et pas xbean.xml)
syntaxe jbi:endpoint:NameSpace[sep]ServiceName[sep]EndPointName
    ou [sep] = : si namespace en urn:...
    et [sep] = / si namespace en http://...
BIEN PENSER a DOUBLER LE / ==> // si le namespace se termine par /
-->

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd

```



```
http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd">
```

```
<camelContext xmlns="http://camel.apache.org/schema/spring">

  <route>
    <from uri="jbi:endpoint:http://basic//CalculateurService/CalculateurPortCamel"/>
    <!-- cette configuration a pour effet de créer le endpoint "CalculateurPortCamel"
         pour le service de namespace "http://basic/" et de nom "CalculateurService" -->
    <to uri="jbi:endpoint:http://basic//CalculateurService/CalculateurPortCxfSe" />
    <!-- cette route configurée ici via camel redirige vers le endpoint
         "CalculateurPortCxfSe" prise en charge par un autre service unit de jbi -->
  </route>
</camelContext>
```

```
</beans>
```

serviceXY (d'une partie JBI "A")

targetService = CalculateurService (de namespace "http://basic/")

targetEndpoint = CalculateurPortCamel (virtuel/indirect)

Routage JBI via camel

```
<from uri="jbi:endpoint:... /CalculateurPortCamel">
<to uri="jbi:endpoint:... /CalculateurPortCxfSe"> ou ...
```

serviceZZ (d'une partie JBI "B")

service = CalculateurService (de namespace "http://basic/")

endpoint = CalculateurPortCxfSe (réel/concret)

→ routage JBI indirect (et flexible) via camel .

5. Integration "camel+cxfr" dans un bundle OSGi de serviceMix

5.1. Exemple avec explications

L'exemple de configuration suivante correspond à un *service "calculateur" accessible sur l'ESB servicemix 4.4* et *adapté* vis à vis d'un service externe/existant via une *transformation XSLT*.

Le **service existant** est (en *version "a"*) pris en charge par une **application "spring+cxfr"** fonctionnant dans **Tomcat**. Son url est "*http://localhost:8080/wsCalculateur/services/calculateur*", son namespace xml est "*http://service.tp/*" et les méthodes sont **addition(a,b)** et **multiplication(a,b)**.

En *version simplifiée "b" (sans dépendance externe)* le service existant est pris en charge par un *autre bundle blueprint osgi* et l'url est alors "*http://localhost:8181/cxf/calculateurWS*".

Le service transformé et exposé sur l'ESB aura l'url "*http://localhost:8181/cxf/camelCalculator*" et les méthodes **add(x,y)** et **mult(x,y)**.

Liste des fichiers du bundle blueprint osgi :

- **pom.xml** (configuration maven pour "blueprint" avec camel+cxfr)
→ voir exemple dans le chapitre "blueprint".
- src/main/resources/**tx1.xslt** transformation xslt
- src/main/resources/**calculateur.wsdl** structure wsdl du service web exposé sur l'esb
(avec add(x,y) et mult(x,y))
- src/main/resources/OSGI-INF/blueprint/**blueprint.xml** config osgi blueprint avec "camel+cxfr"

Configuration "camel + cxfr" au sein de blueprint.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:cxf="http://cxf.apache.org/blueprint/core"
  xmlns:camel-cxf="http://camel.apache.org/schema/blueprint/cxf"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxws="http://cxf.apache.org/blueprint/jaxws"
  xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
http://cxf.apache.org/blueprint/jaxws http://cxf.apache.org/schemas/blueprint/jaxws.xsd
http://cxf.apache.org/blueprint/core http://cxf.apache.org/schemas/blueprint/core.xsd">

<!-- configuration camel-cxf du point d'entrée sur l'ESB.
attention : si endpoint de type point d'entree sur ESB alors il faut fournir une url
de type "/camelCalculator" qui sera relative a "http://localhost:8181/cxf"
et surtout pas d'url absolue en "http" -->
<camel-cxf:cxfEndpoint id="EsbCalculateurPort"
  address="/camelCalculator" xmlns:s="http://service.tp/"
  endpointName="s:EsbCalculateurPort" serviceName="s:CalculatorService"
  wsdlURL="calculateur.wsdl" >
```

```

<camel-cxf:properties>
  <entry key="dataFormat" value="MESSAGE"/> <!-- POJO , MESSAGE , PAYLOAD -->
  <entry key="setDefaultBus" value="false"/>
</camel-cxf:properties>
</camel-cxf:cxfEndpoint>

<camelContext xmlns="http://camel.apache.org/schema/blueprint">

  <route>
    <from uri="cxf:bean:EsbCalculeteurPort" /> <!-- référence au camel-cxf:cxfEndpoint
                                              ci dessus configurant le point d'entrée sur l'esb-->
    <convertBodyTo type="java.lang.String"/>
    <to uri="xslt:tx1.xslt" /> <!-- déclenche la transformation xslt -->

    <wireTap uri="file://target/inbox/" /> <!-- copie envoyée dans un nouveau fichier →

    <to uri="log:calculateurTapInputAfterXslt" /> <!-- log in smx/data/log/servicemix.log →

    <to uri="http://localhost:8181/cxf/calculateurWS?bridgeEndpoint=true"/>
      <!-- bridge/proxy vers service initial avec méthodes addition(a,b) , ... →

    <!-- les éléments de la route qui figurent après la sortie / appel synchrone
         ne sont implicitement utilisés que pour traiter la réponse qui revient : →

    <convertBodyTo type="java.lang.String"/>
    <to uri="log:calculateurTapResponse" />
    <!-- NB log: ou "... :" automatic wireTap = effect of InOut / InOnly MEP -->

  </route>

</camelContext>

</blueprint>

```

NB :

Les balises **cxf:cxfEndpoint** (de camel-cxf) et **jaxws:endpoint** (de jaxws/cxf) sont basées sur des schémas XML différents (bien que les paramétrages soient proches).

Ces deux balises configurent des "endpoints" de manières différentes :

- cxf:cxfEndpoint instancie et associe un "WS endpoint" à un point d'une route de Camel
- jaxws:endpoint instancie et associe un "WS endpoint" à une classe Java (@WebService)

VII - Modélisation SOA (problématiques , bpmn , ...)

1. Modélisation et problématiques SOA

Eléments importants d'une bonne modélisation SOA :

- Services (à avant tout voir de manière fonctionnelle)
- Structure de données échangées (entrées,sorties,exceptions/faults) .
Le vocabulaire "ValueObject" / "Data Transfert Object / View" est ici assez approprié.
- Catégorie/secteur métier englobant tel ou tel service.
- Dépendances fonctionnelles entre les services (et avec structures de données produites/consommées indirectement concernées).
- Processus métiers avec logiques/règles métiers.
- Quelques éléments techniques fondamentaux (volume, perf , synchrone ou asynchrone,)
-

1.1. Orchestration ou bien chorégraphie ?

- Le terme d'**orchestration de service** correspond à une **logique d'interaction de type "maître / subordonnés (promptement coopératifs ou pas)"**. Un **service de haut niveau pilote des services de bas niveaux** en invoquant des opérations dans un ordre bien établi et/ou selon une logique conditionnelle .
- Le terme de **chorégraphie** (SOA/services) correspond en plus à une **interaction plus étroite/collaborative** ou chaque membre **agit de façon plus spontanée (idéalement pas trop dé-synchronisée)**.

==> Techniquement parlant, dans la plupart des cas , l'orchestration suffit (c'est plus simple à mettre en œuvre et plus fiable). La plupart des technologies existantes (bpel , ...) s'occupent de l'orchestration (et pas de la chorégraphie).

==> Fonctionnellement, on a souvent besoin de modéliser des chorégraphies (avec plusieurs couloirs) pour bien montrer une collaboration de principe entre plusieurs parties prenantes (ex : "client" , "logistique" , "fournisseur") qui partagent un même but (ou sous but) et qui sont quelquefois mutuellement engagées au sein d'une logique de partenariat (quelquefois contractuelle).

1.2. UML et/ou BPMN ?

UML est assez universel et très approprié pour la modélisation orientée objet (des services et des structures de données utilisées/échangées par les services) .

BPMN (*Business Process Modeling Notation*) est très approprié pour modéliser les processus métiers (mieux que les diagrammes d'activités d'UML).

UML et BPMN se complémentent plutôt bien .

1.3. en Mode SaaS (Software as a Service) ?

SaaS (*Software as a Service*) est **étroitement lié** à l'**architecture SOA** .

Au lieu que chaque entreprise développe elle même (et sans assez de moyen) ses propres services, il vaut mieux louer des services offerts par des éditeurs expérimentés dans tel ou tel domaine fonctionnel.

Le mode "Saas" apporte quelques nouvelles problématiques :

- compte client / authentications/habilitations sécurisées
- éventuel partage (partiel) de données "publiques" entre plusieurs clients
- base de données mutualisée (avec séparation claire de la propriété des différents "clients") ou
-

1.4. Performances et sécurité (en mode SOA)

Un des problèmes très souvent rencontrés dans une architecture SOA est un **mauvais temps de réponse** (mauvaises performances).

Ceci est du au fait que chaque **intermédiaire** (*technique et/ou fonctionnel*) introduit potentiellement (selon l'architecture exacte) de nouveaux besoins en termes de :

- transfert de données (multiples traversées du réseaux ?)
- traitements CPU (transformations de formats de données / de protocoles)
- reformulations/réinterprétations , encodage/décodage (ex : java <-->xml)
- éventuels cryptages/décryptages (selon sécurité)

Il est clair que l'on ne peut pas avoir le beurre et l'argent du beurre. Autrement dit , la flexibilité/agilité a un prix.

Cependant , toute mise en œuvre d'une architecture SOA devrait idéalement être accompagnée d'une étude annexe d'optimisation des performances (au cas par cas) .

Le second problème récurrent de SOA est la **sécurité** :

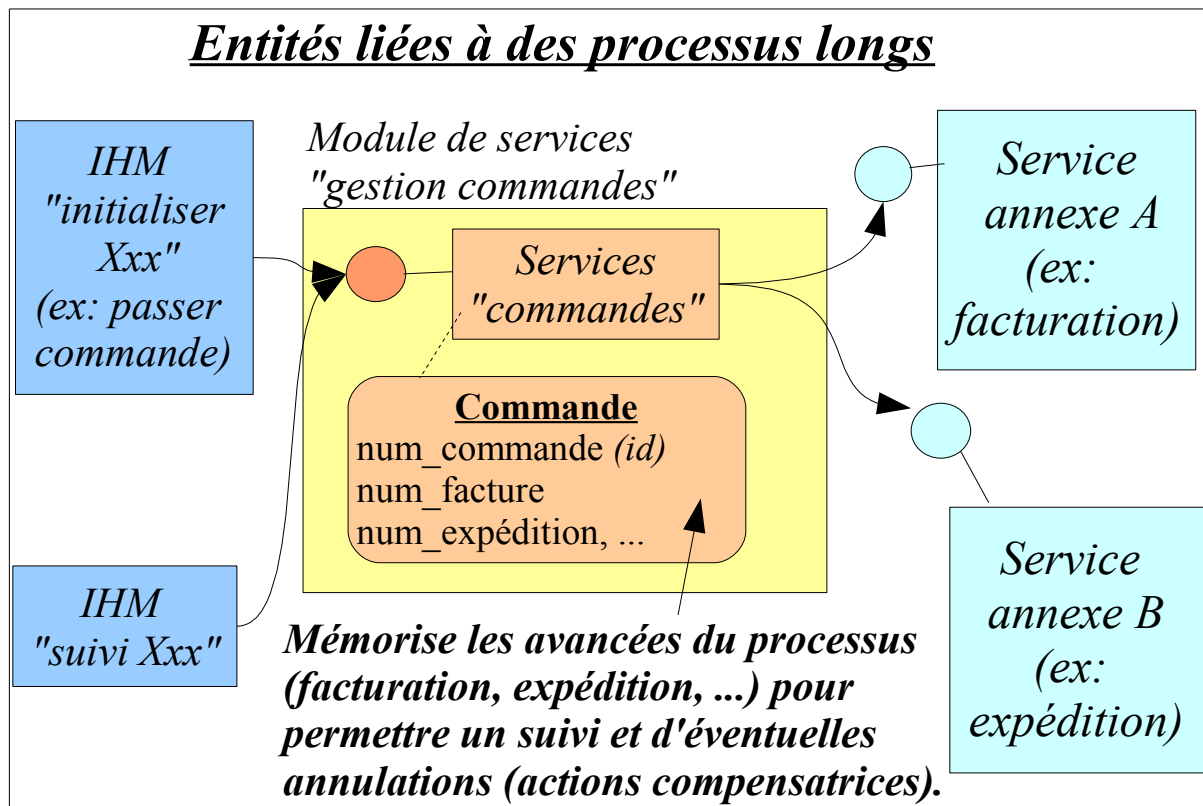
par défaut un service web peut être appelé par n'importe quel client qui connaît son URL .

...

2. Transactions longues et compensations

De nombreux échanges SOA sont effectués en mode asynchrone lorsque certains traitements ou sous processus (ex: livraison) sont longs.

Les éventuelles transactions associées, longues elles aussi, ne peuvent pas se permettre d'exiger un verrouillage ou une isolation temporaire des données mises en jeu sur une longue période. En cas d'échec d'une transaction longue , il faut prévoir des mécanismes de compensation (action inverse annulant l'action d'origine : exemple= remboursement si livraison non effectuée).



Autre exemple classique : entité "dossier" avec num_dossier,

3. Intégrité référentielle & MDM

3.1. Stabilités des éléments référencés

Dans un système à contraintes d'intégrités centralisés (telle qu'une base de données unique prise en charge par un SGBDR), il est souvent impossible de supprimer involontairement/accidentellement une entité qui est encore référencée par une autre.

A l'inverse dans un système plutôt décentralisé comme internet ou SOA , une entité peut être supprimée dans un système (application) sans que les autres systèmes soient au courant (ex: URL devenue invalide, service utilisant un "vieil" id pour référencer une entité qui existait et n'existe plus.

==> Les spécifications (cahier des charges , modélisations,) devraient idéalement spécifier clairement certains éléments tels que la durée de validité d'une référence (et/ou des régulières vérifications ,).

3.2. MDM (Master Data Management)

Pour adresser le problème de l'intégrité référentielle au sein de systèmes décentralisés , on se réfère souvent au concept de "**MDM**" (*Master Data Management*) .

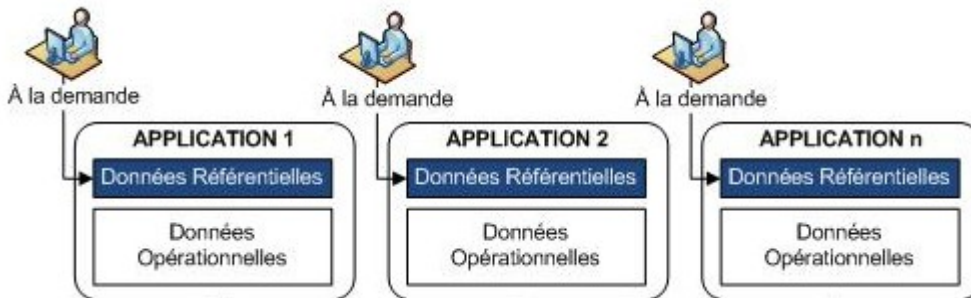
MDM est quelquefois dénommé *GDR (Gestion de Données Référentielles)* en français .

MDM (ou GDR) est surtout utilisé pour gérer les données fondamentales (*de références*) d'une grande entreprise :

- les données « clients/fournisseurs »
- les données « produits »
- les données « financières ».

MDM en mode "Maître/secondaire" :

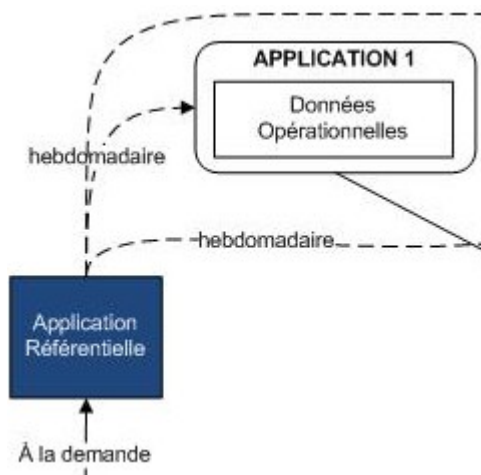
sans MDM :



avec risques d'incohérences.

avec MDM :

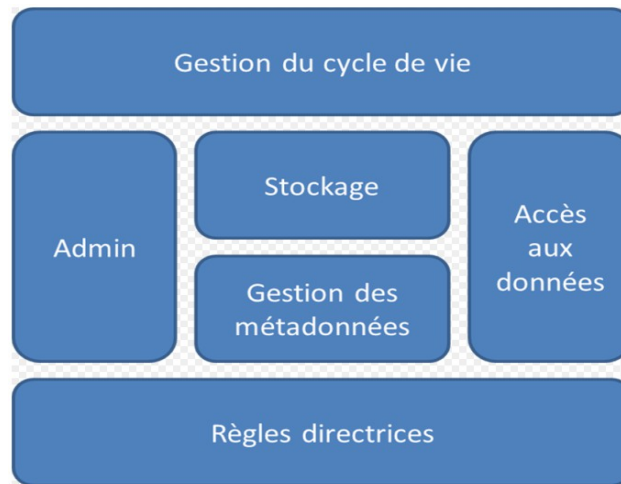
- Les données référentielles sont encodées et maintenues en un seul endroit, ce qui diminue le coût opérationnel lié à la maintenance et à l'encodage.
- Le système est le maître des données. Il les contrôle en sélectionnant quelles données il transmet à quel système.
- Le système contient une seule version active (il peut exister plusieurs versions inactives, tant passées que futures ; c'est même recommandé pour une meilleure flexibilité de la solution). Il est donc le garant de la seule version de la vérité et en cas de litige, sa version tient lieu de version officielle.



(ici en mode "push") .

Un mode "pull" ou bien "push-pull" est envisageable.

Fonctionnalités interne d'un système MDM :



MDM au cas par cas (selon modélisation) :

[http://www.dsi.cnrs.fr/description_referentiel/donnees/Documents/Livre blanc MDM Solucom.pdf](http://www.dsi.cnrs.fr/description_referentiel/donnees/Documents/Livre%20blanc%20MDM%20Solucom.pdf)

Distinguer :

- données maîtres (partagées , avec différentes vues , stratégiques, sous système propriétaire difficile à identifier)
- données procédurales (fruit d'une procédure , d'un processus métier , factuel , date précise, ...)

et bien modéliser les cycles de vie (lorsque c'est possible)
et les dépendances "référence principale" / "vue / copies" .

4. Quelques repères d'urbanisation

4.1. Urbanisation du S.I. (présentation)

L'objectif principal consiste à *faire évoluer les différentes parties du SI de façon convergente* (en respectant les mêmes grandes lignes directrices) et à *faire cohabiter les parties existantes* au sein d'une *structure globalement découpée en zones/quartiers/blocs*.

Ce découpage en **modules** et sous modules **autonomes** vise à :

- garantir une certaine liberté d'implémentation
- clarifier les zones d'échanges (communications) entre les différentes parties du SI.

Plus particulièrement, l'urbanisation vise :

- à renforcer la capacité à construire et à intégrer des sous-systèmes d'origines diverses,
- à renforcer la capacité à faire interagir les sous-systèmes du SI et les faire interagir avec d'autres SI ([interopérabilité](#)),
- à renforcer la capacité à pouvoir remplacer certains de ces sous-systèmes (interchangeabilité).

et de manière générale pour le SI à :

- favoriser son évolutivité, sa pérennité et son indépendance,
- renforcer sa capacité à intégrer des solutions hétérogènes ([progiciels](#), éléments de différentes plate-formes, etc.).

4.2. Zones , quartiers , blocs

L'urbanisation consiste à découper le SI en modules autonomes, de taille de plus en plus petite :

- les **zones**,
- les **quartiers** (et les **îlots** si nécessaire),
- les **blocs** (blocs fonctionnels).

Exemple :

- zone production bancaire
 - quartier gestion des crédits
 - îlot gestion des crédits immobiliers
 - bloc fonctionnel gestion d'un impayé

Entre chaque module (zone, quartier, îlot, bloc) se dessinent des **zones d'échange d'informations** qui permettent de *découpler* les différents modules pour qu'ils puissent évoluer séparément tout en conservant leur capacité à interagir avec le reste du système.

4.3. Les différents types de zones

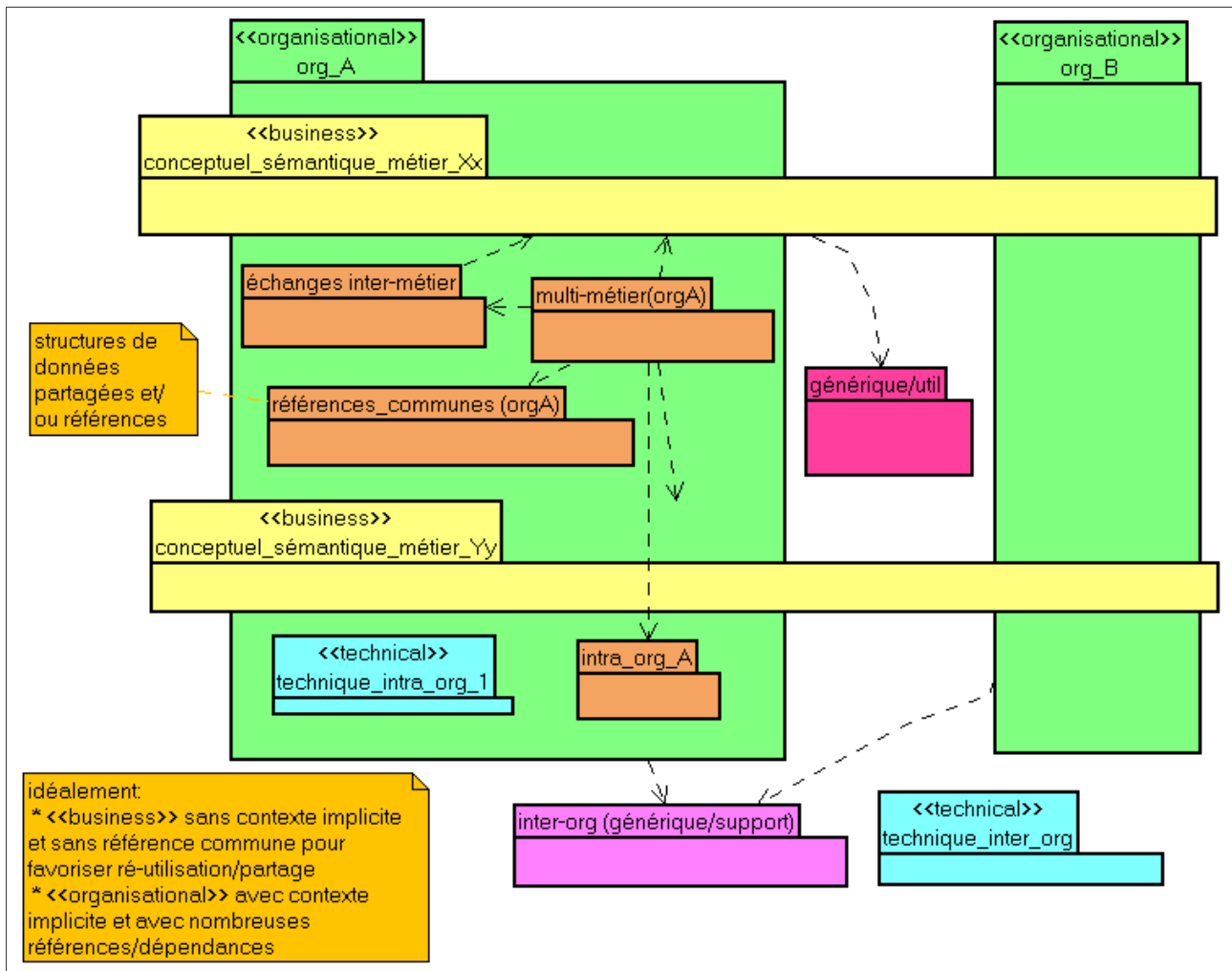
Zones	Caractéristiques
<i>échanges</i> avec l' <i>extérieur</i> du SI	<u>acquisition</u> /émission de/vers les <u>partenaires</u> : <u>clients</u> , <u>fournisseurs</u> , etc.
<i>activités opérationnelles</i>	gestion des opérations bancaires, gestion des opérations commerciales, gestion des opérations logistiques internes, etc.
gestion des <u>données de référence</u> communes à l'ensemble du SI	les <u>référentiels</u> de <u>données structurées</u> (<u>données clients</u> , <u>catalogue de produits</u> et <u>services</u> , etc.)
gestion des <i>gisements de données</i>	ensemble des <i>informations produites quotidiennement</i> , communes à l'ensemble du SI (données de production, etc.)
<i>activités de support</i>	<u>comptabilité</u> , <u>ressources humaines</u> , etc.
aide à la <u>décision</u> et le <i>pilotage</i>	<u>informatique décisionnelle</u> .

5. Quintessence d'une bonne modélisation SOA

5.1. Distinguer le "pur fonctionnel/métier" de l'organisationnel

Distinguer au besoin :

- domaines purement métiers (comptabilité , R&D , marketing , ...) avec services métiers élémentaires
- domaines organisationnels (départements , centres de gestion ,) avec contextes
- domaines purement techniques (facilement réutilisables , sans contexte , ...)



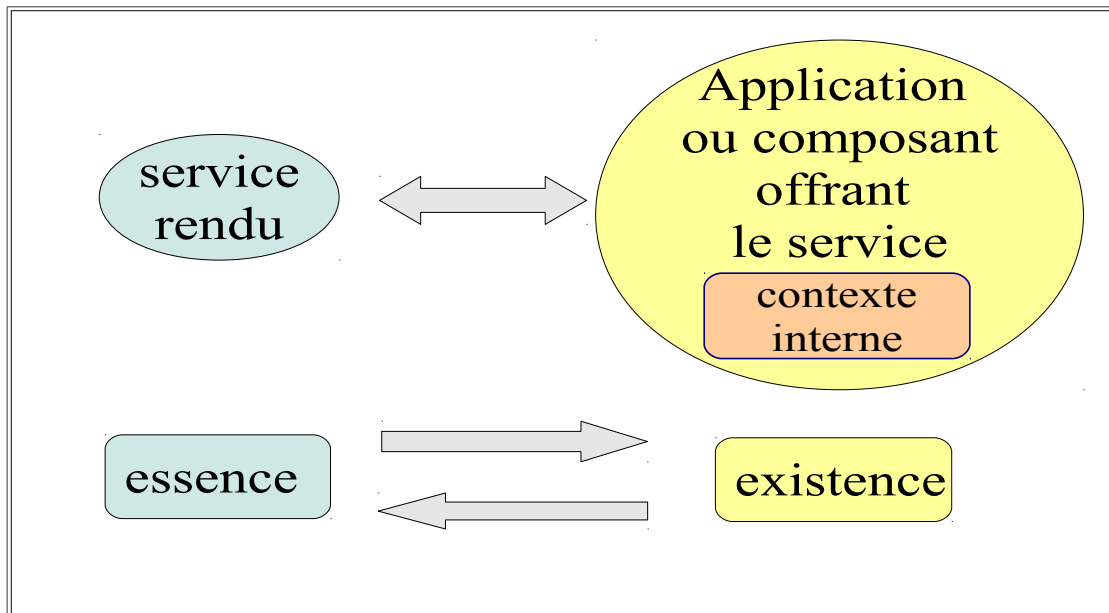
5.2. Bien modéliser (d'une façon ou d'une autre) les contextes

A part quelques cas triviaux (ex: service de calcul élémentaire) , la plupart des services sont intelligibles (sans ambiguïté) que si l'on précise un minimum certains contextes :

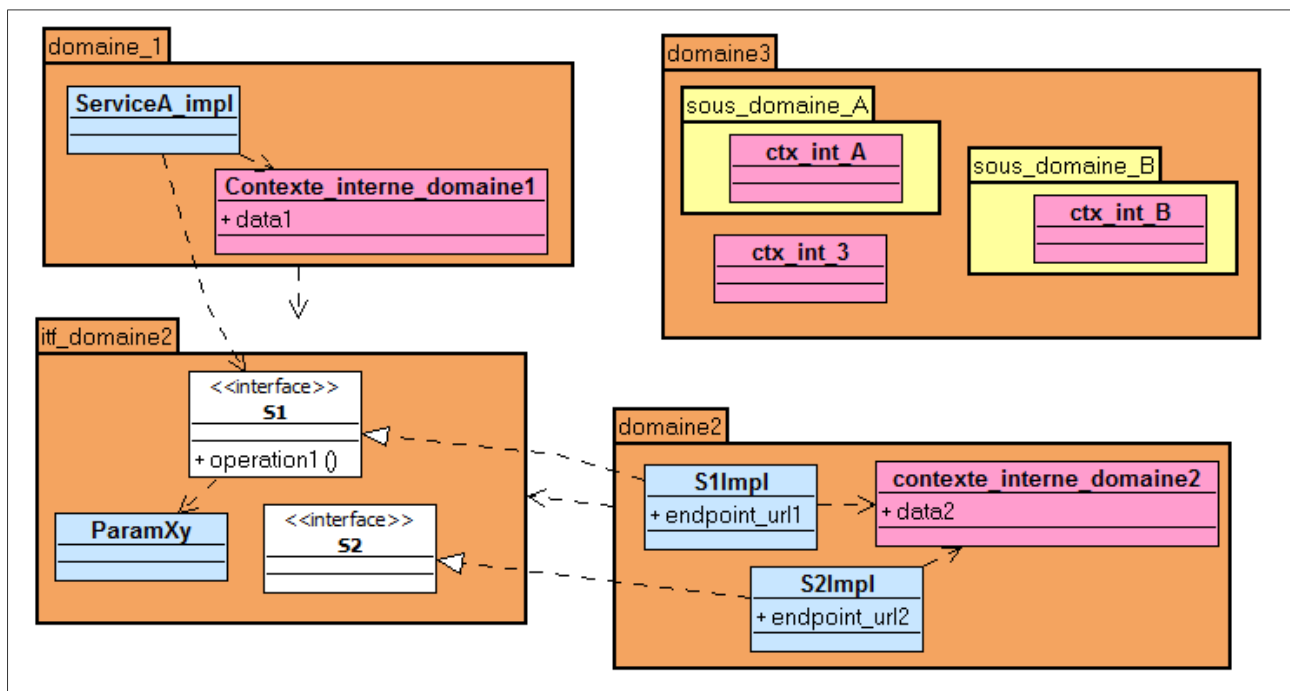
- contexte de persistance (base de données de l'application offrant le service)
- contexte d'exécution (mémoire de l'état d'avancement d'un processus métier : processus long en partie asynchrone)
- contexte implicite/explicite , privé/public , opaque ,

5.3. Penser à une structuration logique (future implémentation) à base de composants offrant des services

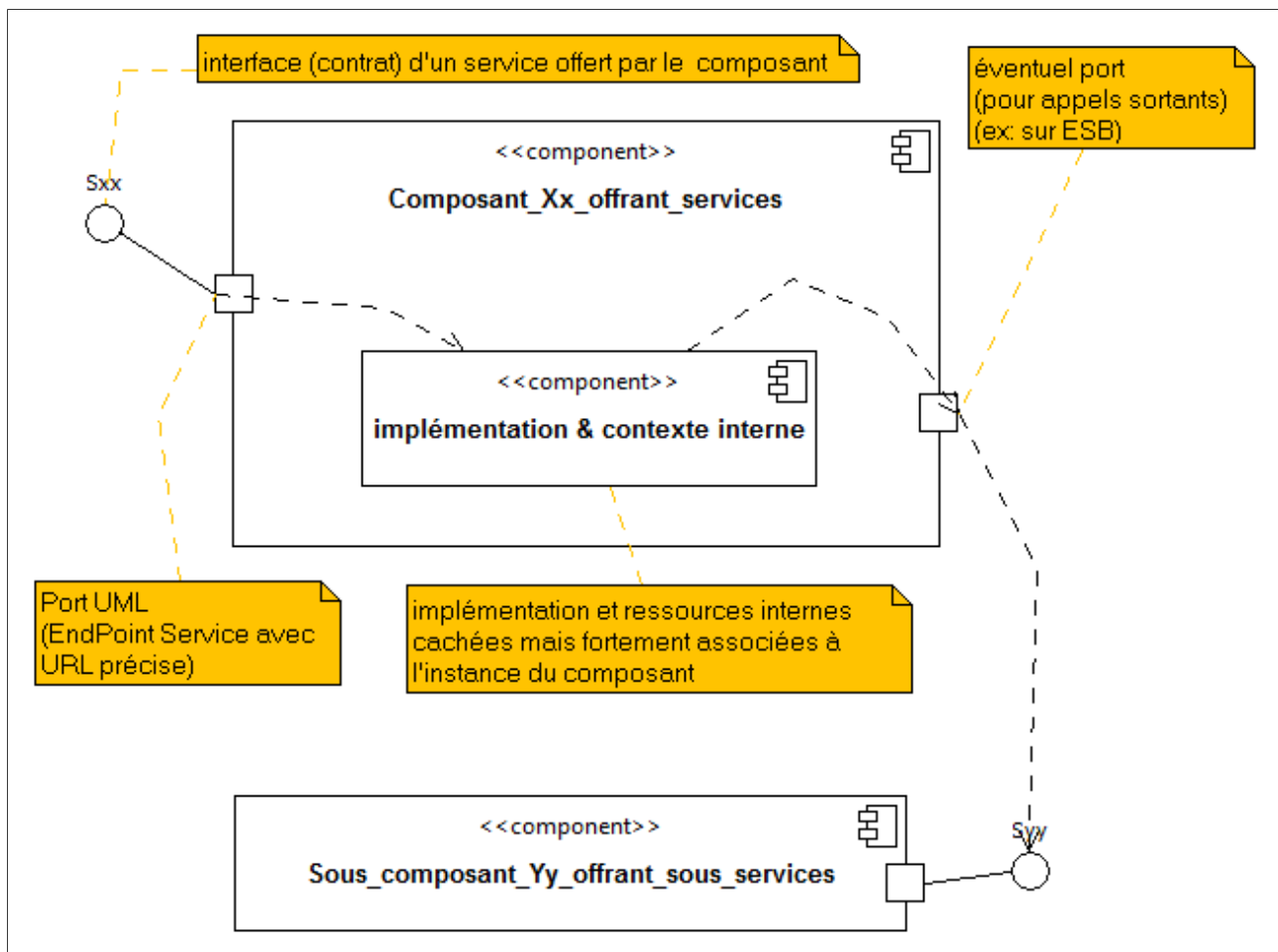
Beaucoup de contextes peuvent être concrètement pris en charge par des composants informatiques (de petites ou grandes tailles) .



En UML, cette notion de contexte peut se modéliser via des diagrammes de classes (avec packages)

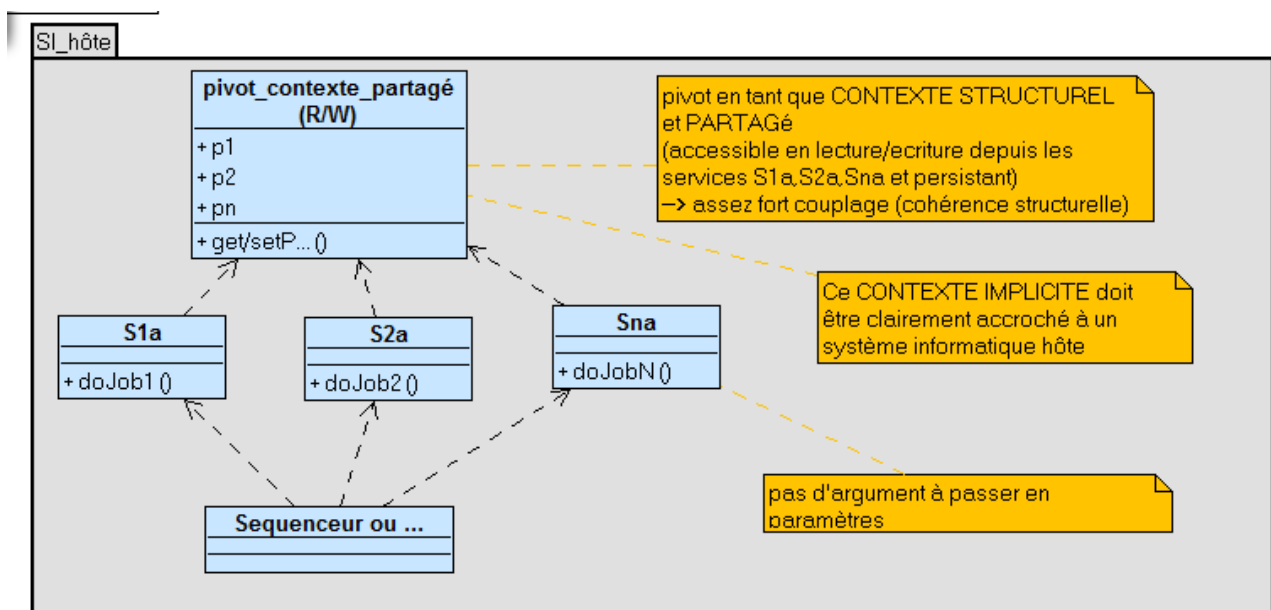


ou bien via des diagrammes de composants .../...



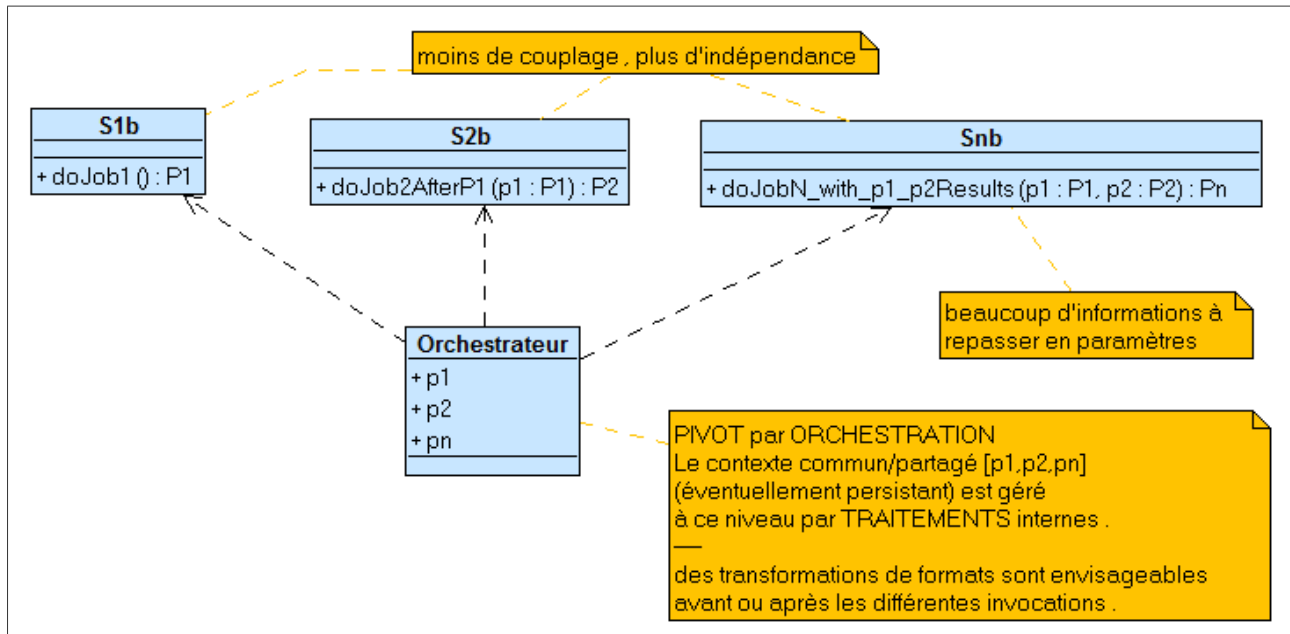
5.4. Bien choisir les types de pivots

Pivot en tant que contexte structurel partagé :



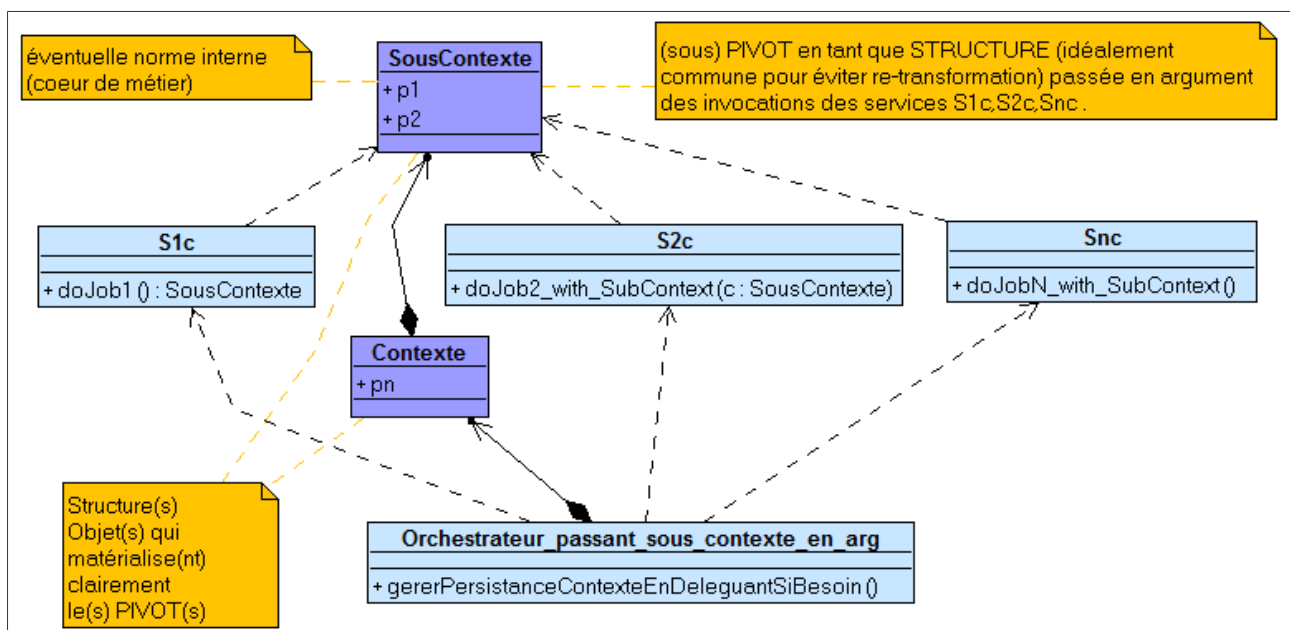
→ envisageable que si services clairement cloisonnés à un sous système informatique précis .

Pivot par orchestration :



→ adapté si les services à orchestrer sont placés dans des sous systèmes informatiques différents .

Pivot en tant que structure de données (éventuellement partagée) passée en argument :



6. BPMN

6.1. Présentation de BPMN

BPMN signifie *Business Process Management Notation*

- Il s'agit d'un formalisme de modélisation spécifiquement adapté à la modélisation fine des processus métiers (sous l'angle des activités) et prévu pour être transposé en BPEL ou jBpm.
- Un diagramme **BPMN** ressemble beaucoup à un diagramme d'activité UML . Les notions exprimées sont à peu près les mêmes.

Les différences entre UML et BPMN sont les suivantes:

- la syntaxe des diagrammes d'activités UML est dérivée des diagrammes d'états UML et est plutôt orientés "technique de conception" que "processus métier"
- à l'inverse la syntaxe des diagrammes BPMN est plus homogène et plus parlante pour les personnes qui travaillent habituellement sur les processus métiers.
- UML étant très généraliste et avant tout associé à la programmation orientée objet, il n'y a pas beaucoup de générateurs de code qui utilisent un diagramme d'activité UML
- à l'inverse la plupart des éditeurs BPMN sont associés à un générateur de code BPEL ou jBpm ou Les éléments fin d'une modélisation BPMN ont été pensées dans ce sens.

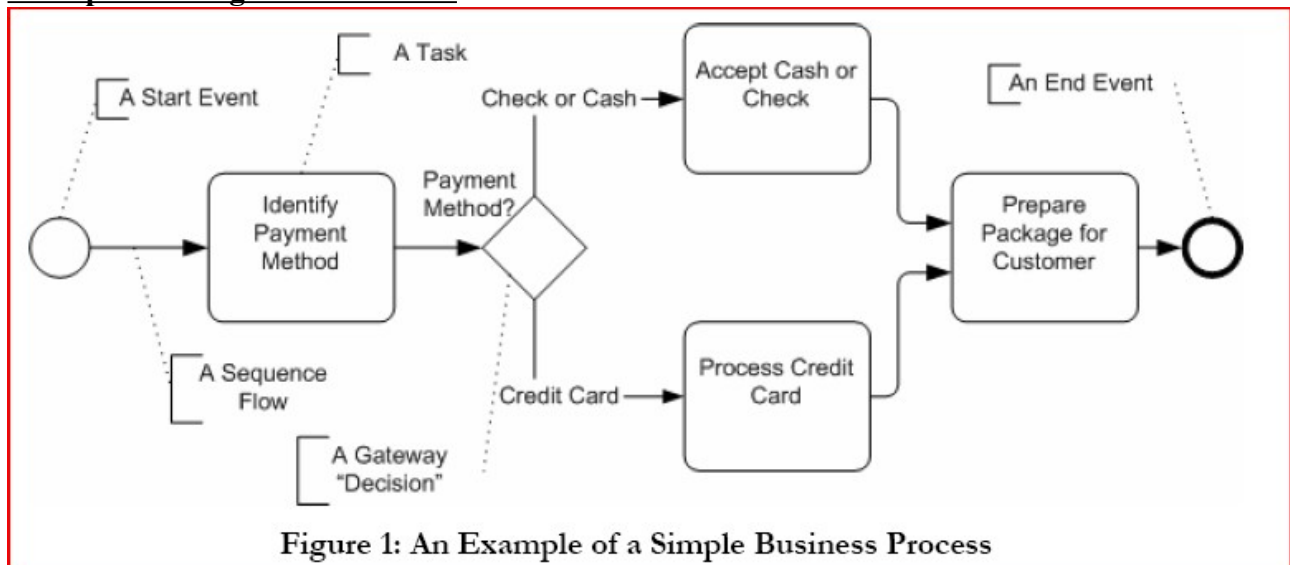
Outils concrets pour la modélisation BPMN :

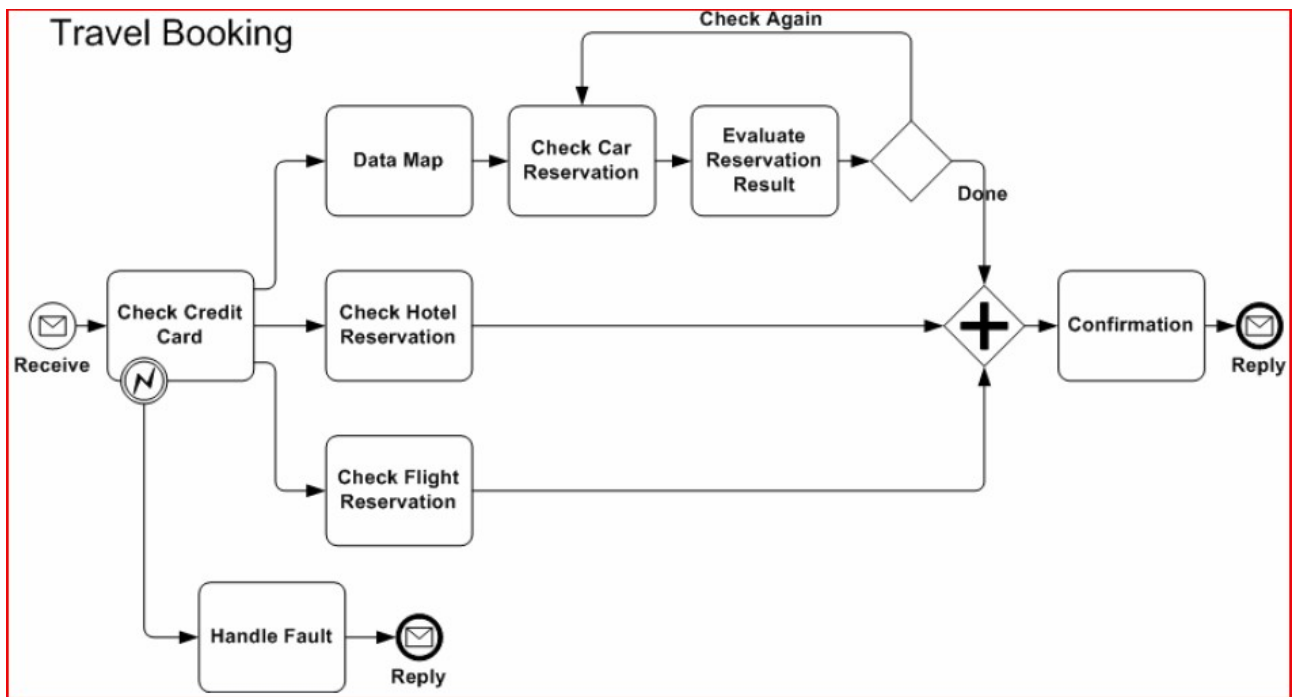
- **Intalio BPMN** .
- **Bizagi Process Modeler** (version gratuite)
- **Editeur BPMN2 intégré à l'IDE eclipse** (drools / jbp5).

NB : Certains outils BPMN 1.x (tels que **Bizagi**) sont capables d'effectuer des imports/exports au format **XPDL** . **XPDL** signifie "*Xml Process Definition Language*" : c'est une sorte de sérialisation Xml de BPMN.

Des outils BPMN récents (supportant **BPMN 2**) peuvent quelquefois directement utiliser xml comme format natif (ex : **Jboss drools/jbp5**) .

Exemples de diagrammes BPMN:





6.2. Principales notations BPMN :

Flow objects :

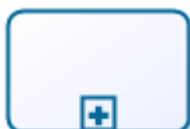
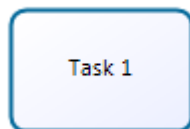
Event (événement) :

- **Start** (début)
- **Intermediate** (intermédiaire)
- **End** (fin)



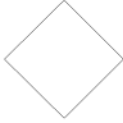




Activity (activité) :

- **Task** (tâche)
- **Sub-Process** (sous processus)






Gateway (décision / contrôle du flow)



Le losange (gateway) peut comporter alternativement plusieurs types/marqueurs :

Exclusive		Le mode exclusif désigne une seule route de sortie possible (selon condition/décision)
Data-Based		
Event-Based		Le mode inclusif désigne plusieurs (au moins 2) routes de sortie possibles (1 branche principale/obligatoire et d'autres branches facultatives) qui se rejoignent généralement par la suite
Inclusive		
Complex		
Parallel		
		Parallel (=fork)

Connecting objects (connexions) :

- **sequence flow** (séquence ordonnée) 
- **message flow** (entre 2 participants) 
- **association** (entre tâche et données) 

Swimlane (couloir/partition d'activités)

- **pool** (un par participant : *Processus* ou *Partenaire* ou ...) 
- **lane** (sous partition) 

















Data Objects :

- **Data** (document xml , objet en mémoire , ...) 

- **Data Store** (*extraction/persistance en base,...*)



6.3. Types précis d'événements BPMN :

Start	None  Message  Timer  Rule  Link  Multiple 	
Intermediate	Idem "Start" + Error  Compensation 	
End	None  Message  Error  Compensation  Link  Terminate  Multiple  et  Cancel	

VIII - BPEL (présentation , mode synchrone)

1. BPEL (présentation)

BPEL (*Business Process Execution Language*)

BPEL (*BPEL4WS* renommé *WS-BPEL*) a été créé par le consortium OASIS et vise à encoder en Xml des services Web de haut niveau qui orchestrent ou pilotent des services Web de plus bas niveaux.

On parle généralement en terme de "*processus collaboratif*" pour désigner le type de services produit via BPEL .

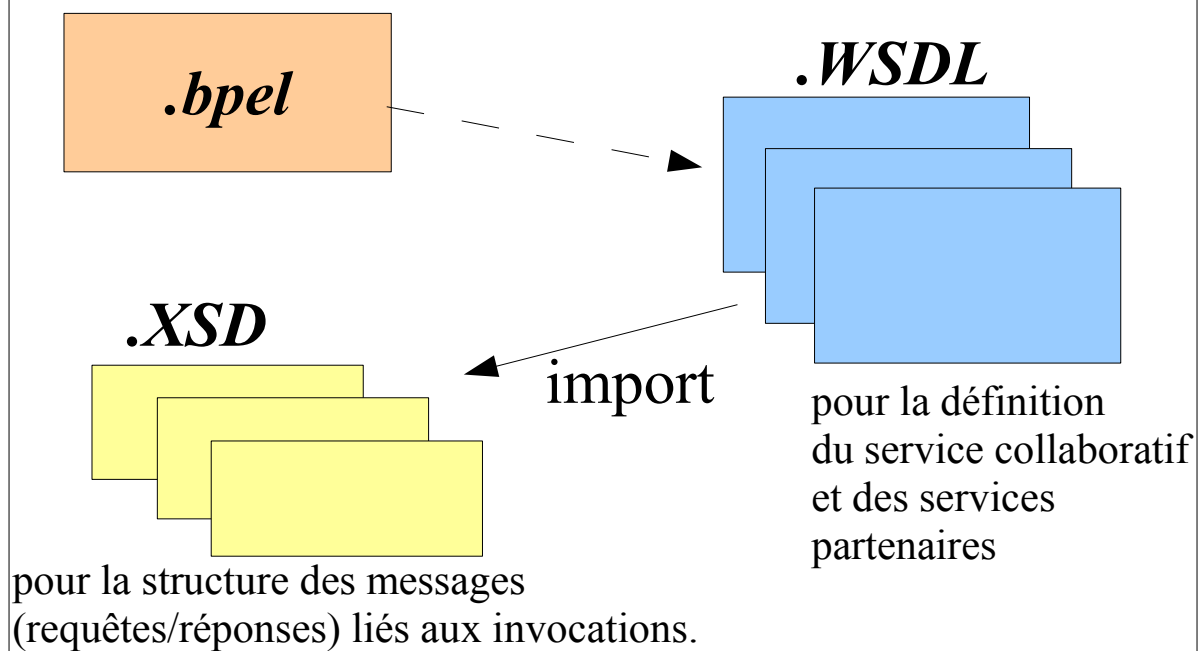
NB: Deux grandes versions :

- * BPEL 1.1 (2003) et BPEL 2.0 (2007)
- * En pratique quelques variantes d'interprétations selon le moteur d'exécution (ODE, Orchestra , ...)

Principales fonctionnalités de BPEL:

- **invoquer** des services externes (partenaires)
- gérer finement le *contenu des messages de requêtes et de réponses* avant et/ou après les différentes invocations.
- ajouter une logique métier de haut niveau (tests , vérifications, *copies et calculs de valeurs à retransmettre*).
- **orchestrer / séquencer** de façon adéquate les invocations(*synchrone ou asynchrone*) de services partenaires.

Technologies XML utilisées par BPEL:



Quelques implémentations : moteurs BPEL

- **ODE** (open source , apache)
à intégrer dans *Tomcat* ou *ServiceMix*
- **BPEL-SE** de OpenESB et GlassFish V2 (SUN)
- **Orchestra** (Open source, OW2)
- **BizTalk Server** (Microsoft)
- **WebSphere Process Server** (IBM)
- **Oracle BPEL Process Manager**
- **SAP Exchange Infrastructure**
- **ActiveVOS**
- ...

1.1. Structure BPEL

Structure générale d'un processus BPEL:

```
<?xml ... ?>
<process ...>
  <import ...="...wsdl"/>
  <import ...="...wsdl" />
  <partnerLinks>
    <partnerLink .../>
  </partnerLinks>
  <variables>
    <variable .../>
  </variables>
  ... instructions ...
</process>
```

**déclarations logiques
des services partenaires**
(types précis définis dans
wsdl , partenaire spécial =
le process bpel lui même)

déclarations de variables
(messages transmis , parties
recopiées ou calculées)
(types précis définis dans
schémas ".xsd" des ".wsdl")

Séquence type d'un processus BPEL:

```

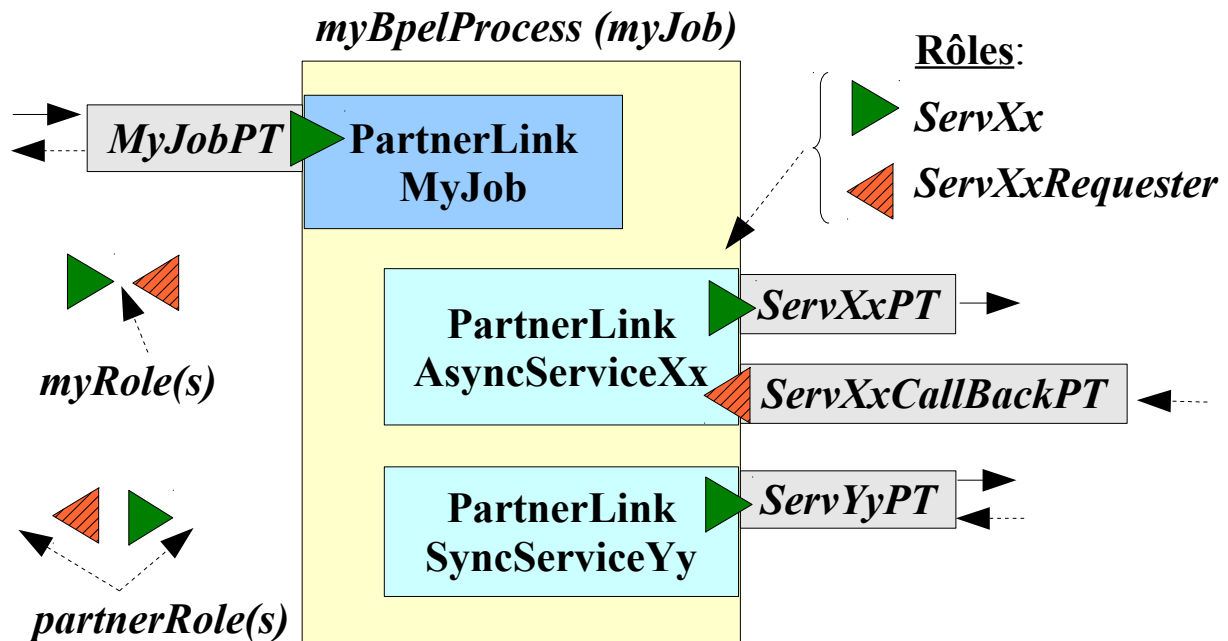
<process ...> ....
  <sequence>
    <receive partnerLink=".."
      operation="..." variable="..." />
    <assign>
      <copy> from ... to ... </copy>
    </assign>
    <invoke partnerLink=".."
      operation=".." inputVariable=
        "..." outputVariable="..." /> ...
    <reply partnerLink=".."
      operation="..." variable="..." />
  </process>

```

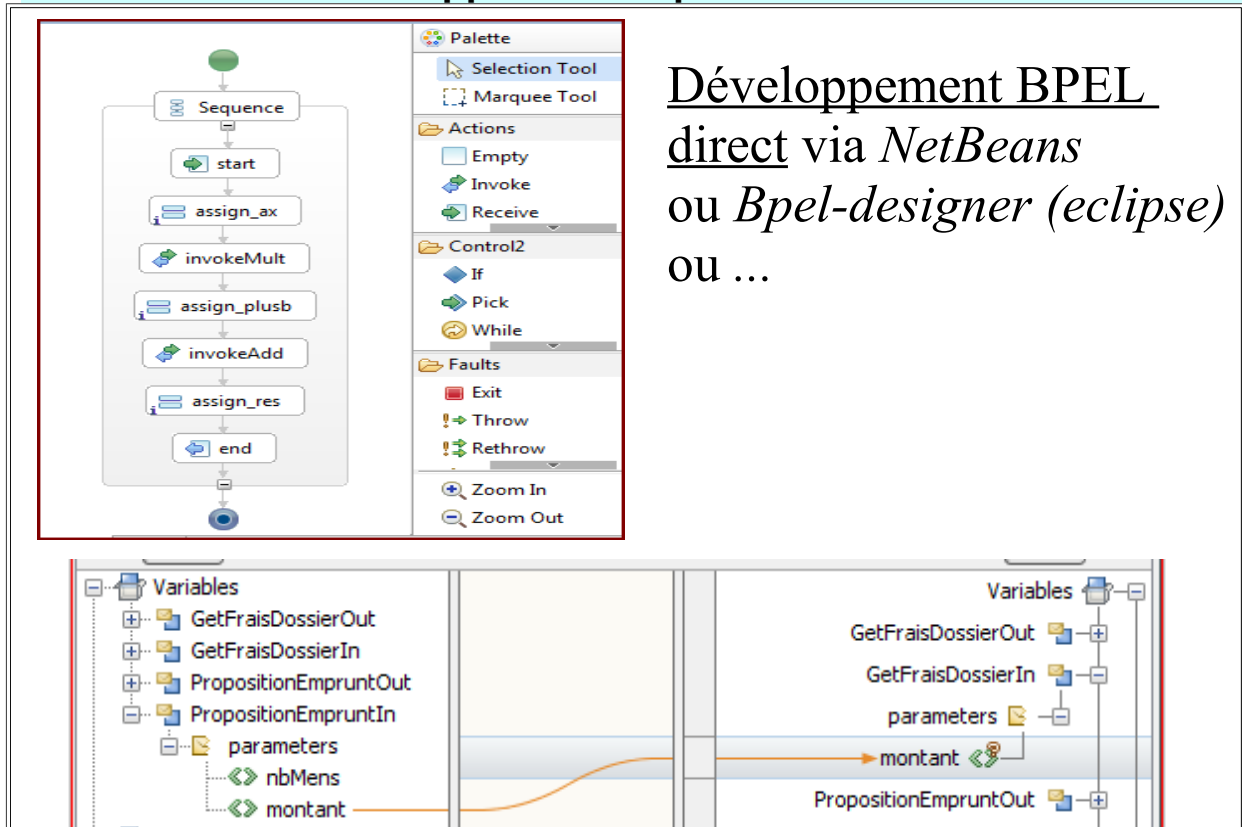
*réception d'une
requête et affectation
du message
dans une variable.*

*invocation d'une
opération
sur un service Web
partenaire*

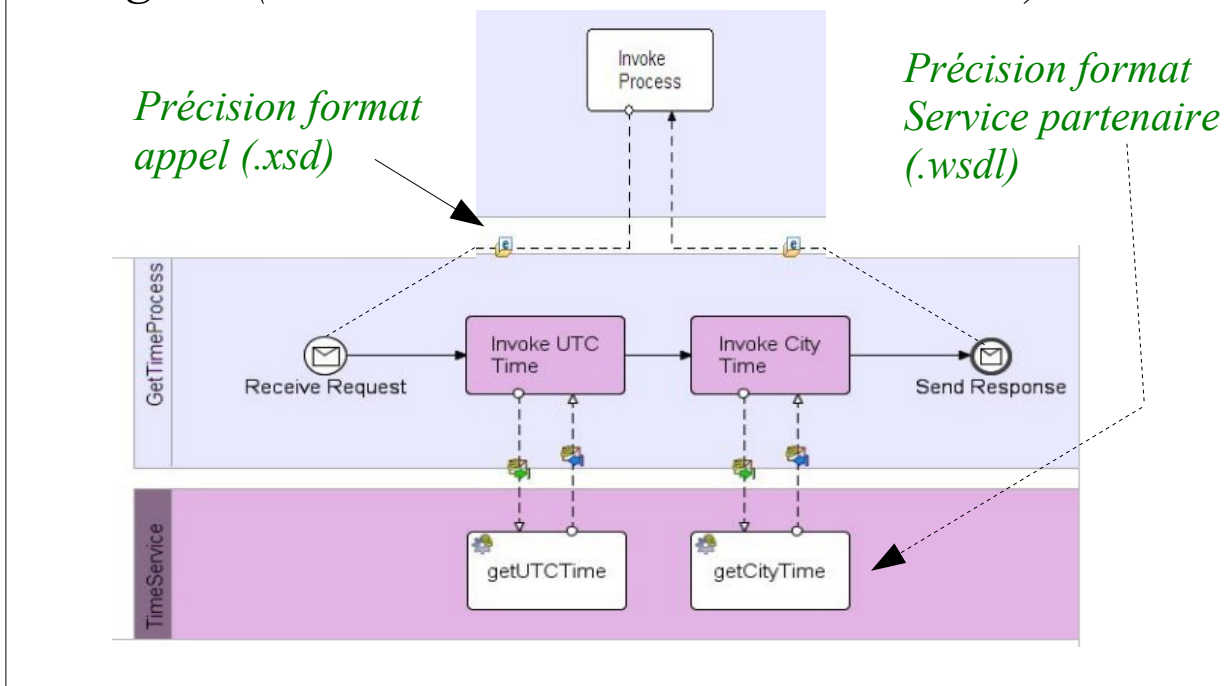
BPEL – PartnerLinks & PortType:



1.2. Vues sur le développement de processus BPEL

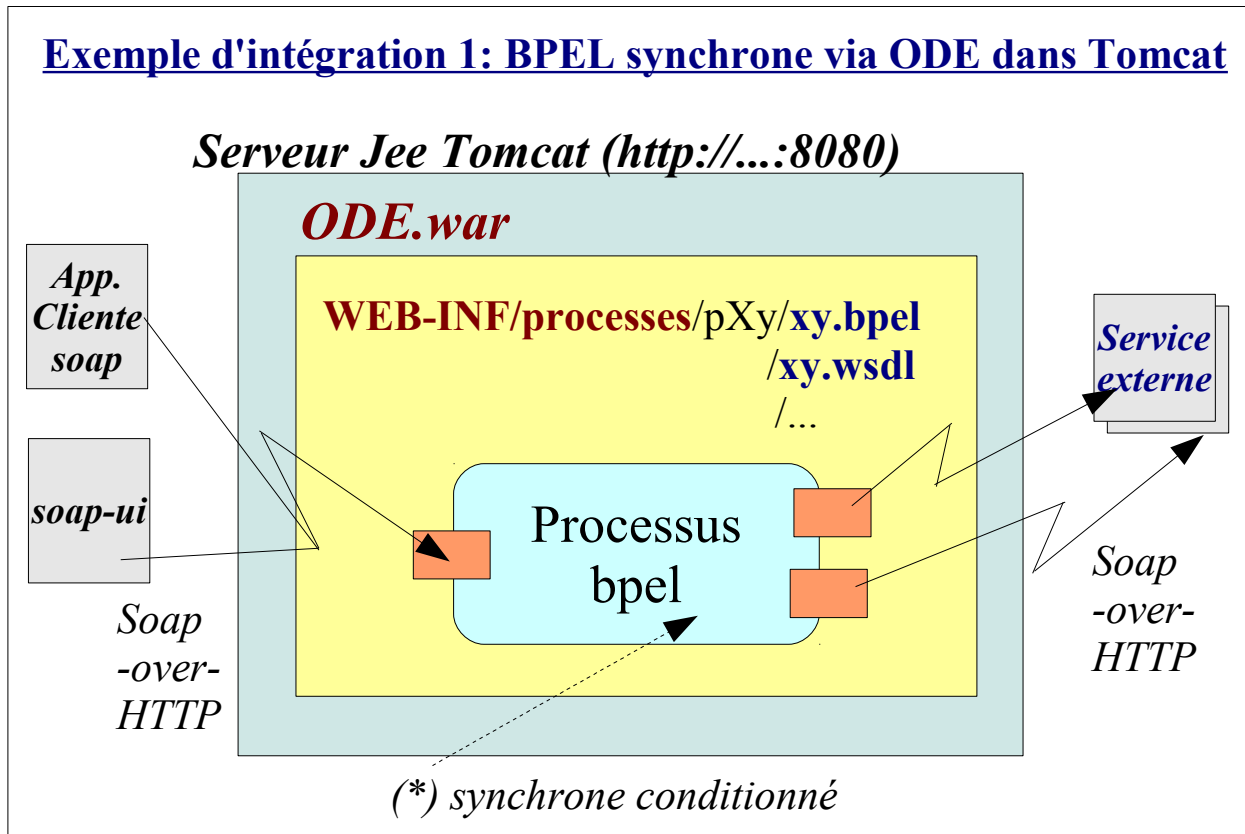


Développement BPEL indirect depuis *Intalio BPMNs Designer* (*modélisation BPMN → code BPEL*)



1.3. BPEL en mode synchrone et exemple d'intégration

Exemple d'intégration 1: BPEL synchrone via ODE dans Tomcat



2. BPEL (détails)

2.1. Éléments WSDL & XSD importés

BPEL - Importation des définitions WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="..." targetNamespace="..."
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="..." xmlns:ns0="...." ...>

  <import namespace="http://yyy/xyz"
    location="xxx.wsdl"
    importType=
      "http://schemas.xmlsoap.org/wsdl/" />

  ....
</process>
```

Éléments WSDL pour BPEL (1/2): yyy.wsdl

```
<definitions targetNamespace="http://xxx/wsdl/yy"
  xmlns="http://schemas.xmlsoap.org/wsdl/" ...
  xmlns:tns="http://xxx/wsdl/yy"
  xmlns:ns="http://xxx/schema/yyy" >
  <types>
    <xsd:schema targetNamespace="http://xxx/wsdl/yy">
      <xsd:import namespace="http://xxx/schema/yyy"
        schemaLocation="yyy.xsd"/>
    </xsd:schema>
  </types> ...
  <message name="...." >
    <part name="parameters" type="ns:abcType"/>
  </message>
  ...
```


Elements XSD (via wsdl) pour BPEL:

yyy.xsd

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xxx/schema/yyy"
  xmlns:tns="http://xxx/schema/yyy" elementFormDefault="qualified">
  <xsd:element name="operation1Req" type="tns:op1ReqType" />
  <xsd:element name="operation1Resp" type="tns:op1RespType" />
  <xsd:complexType name="op1ReqType">
    <xsd:sequence> <xsd:element name="x" type="xsd:int"/>
      <xsd:element name="y" type="xsd:double"/> </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="op1RespType">
    <xsd:sequence> <xsd:element name="a" type="xsd:double"/>
      </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Eléments WSDL pour BPEL (2/2):

yyy.wsdl

```

<definitions ... xmlns:tns="http://xxx/wsdl/yy" ...
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype" >
  <types>... </types> <message name="...." >... </message> ...
  <portType name="serviceXyPortType">
    <operation name="operation1">
      <input name="input1" message="tns:operation1Request"/>
      <output name="output1" message="tns:operation1Reply"/>
    </operation> .... </portType>
  <binding ....> </binding> ... <service ....> </service>
  <plnk:partnerLinkType name="ServiceXyPartnerLinkType">
    <plnk:role name="ServiceXyPortTypeRole"
      portType="tns:serviceXyPortType"/>
  </plnk:partnerLinkType> </definitions>

```

2.2. partnerLink(s)

BPEL – Déclaration des "partnerLink"

```

<process ...> ....
<partnerLinks>
  <partnerLink name="ServiceXyPartnerLink"
    xmlns:ns0="http://xxx/xyz"
    partnerLinkType="ns0:ServiceXyPartnerLinkType"
    partnerRole="ServiceXyPortTypeRole"/>
  <partnerLink name="MyJobPartnerLink"
    xmlns:tns="http://yyy/myjob"
    partnerLinkType="tns:MyJobPartnerLinkType"
    myRole="myJobPortTypeRole"/>
</partnerLinks> ....
</process>

```

2.3. variables bpeL

BPEL – Déclaration des variables

```

<process ...>...
<variables>
  <variable name="operation1ServXYOut"
    xmlns:ns0="http://xxx/xyz"
    messageType="ns0:operation1Reply"/>
  <variable name="operation1ServXYIn"
    xmlns:ns0="http://xxx/xyz"
    messageType="ns0:operation1Request"/>
  <variable ... messageType="xsd:String" />
</variables> ....
</process>

```

2.4. principales activités

BPEL – Liste des principales "*activity*"

receive	réception message
reply	réponse du processus
invoke	invocation opération – service
assign / copy	affectation de variable(s)
sequence	suite séquentielle d'activités
flow	suite d'activités concurrentes (en //)
....	

BPEL – receive & reply:

```

<process ...> ....
  <sequence>
    <receive operation="operationI"
      partnerLink="MyJobPartnerLink"
      variable="operationIServXyIn"
      createInstance="yes"
      xmlns:tns="http:yyy/myjob"
      portType="tns:MyJobPartnerLinkType" />
    ...
    <reply operation="..."
      partnerLink="MyJobPartnerLink"
      variable="...Out" portType="..." />
  </sequence> </process>

```

*réception
d'une
requête et
affectation
du message
dans une
variable.*

*réponse
globale du
processus
BPEL*

BPEL – assign , copy [from, to]:

```

<process ...> .... <sequence> ...
<assign>
  <copy>
    <from>$op1ServYIn.parameters/ns0:montant</from>
    <to>$opAServZIn.parameters/ns1:somme</to>
  </copy>
  <copy>
    <from> ( $aaIn.parameters/ns0:x div
              $bbIn.parameters/ns0:y ) </from>
    <to>$op1ServYOut.parameters/ns0:a</to>
  </copy>
</assign>
...

```

BPEL – invoke:

```

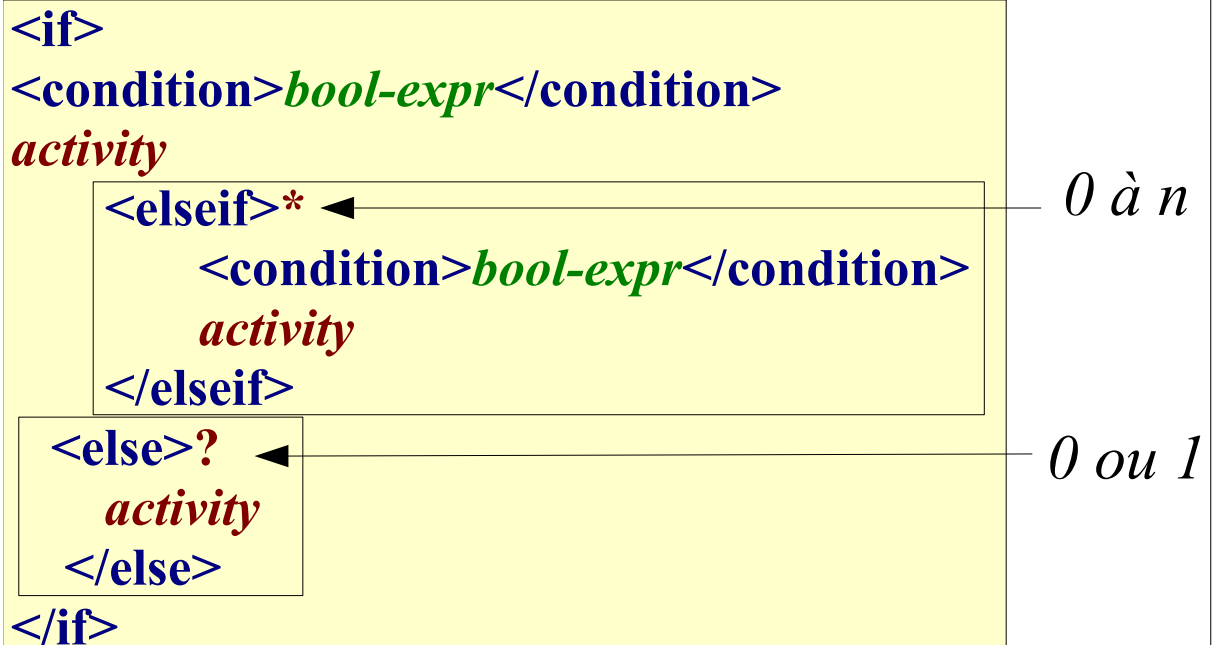
<process ...> ....
  <sequence> ...
    <invoke operation="operationZ"
      partnerLink="ServiceXyPartnerLink"
      inputVariable="...In"
      outputVariable="...Out"
      xmlns:ns0="http://xxx/xyz"
      portType="ns0:ServiceXyPartnerLinkType"/>
    ...
  </sequence> ...
</process>

```

*invocation
d'une
opération
sur un
service
Web
partenaire*

NB: en mode *asynchrone*,
 <invoke operation="callbackXy"
 .../> remplace <reply .../>
 après opérations longues .

2.5. tests et boucles

BPEL – test (if/else):**BPEL – boucles *while* et *repeatUntil***

```

<while>
<condition>bool-expr</condition>
activity
</while>

```

```

<repeatUntil>
activity
<condition>bool-expr</condition>
</repeatUntil>

```

exemple: \$var1.xx = 10 ou > , < , >= , <= , != ,
not(...) , ... **and/or** ... (syntaxe xpath 1.0 / xslt)

BPEL – boucle *forEach* (de n à m)

```
<forEach counterName="i">  
<startCounterValue>1</startCounterValue>  
<finalCounterValue>10</finalCounterValue>  
<scope>activity*<scope>  
</forEach>
```

*la variable bpel i doit être de type
xsd:integer ou équivalent*

*D'autres options sont disponibles pour forEach
==> approfondir si besoin la norme WS-BPEL2*

2.6. activités concurrentes (flow) + synchronisations (link)

BPEL flow – activités concurrentes (en //)

```
<flow>
```

```
  <links>?
```

```
    <link name="lien_xy"/>+
```

```
  </links>
```

```
  activity+
```

```
</flow>
```

liaisons (facultatives)
= dépendances
(avant/après) entre
activités pour exprimer
des synchronisations

activités en //

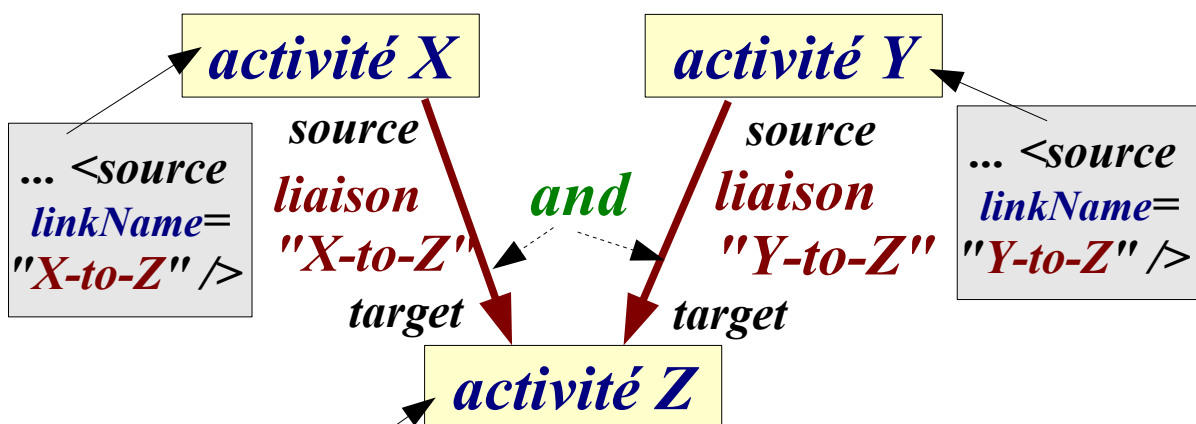
```
<invoke ou ...> ...
```

```
  <targets et/ou sources>?
```

```
    <target ou source linkName="lien_xy"/>+
```

```
  </targets et/ou sources> ... </invoke ou ...>
```

BPEL flow – link (source , target) , join



```
<invoke ...>
```

```
  <targets>
```

```
    <joinCondition>$X-to-Z and $Y-to-Z </joinCondition>
```

```
    <target linkName="X-to-Z" />
```

```
    <target linkName="Y-to-Z" /> ..
```

2.7. Exemple BPEL complet

SimpleCalculator.wsdl (service invoqué)

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://basic/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  targetNamespace="http://basic/" name="SimpleCalculatorImplService">
  <types>
    <xsd:schema xmlns:tns="http://basic/" elementFormDefault="unqualified"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="1.0"
      targetNamespace="http://basic/">
      <xsd:element name="addition" type="tns:addition" />
      <xsd:element name="additionResponse" type="tns:additionResponse" />
      <xsd:element name="multiplication" type="tns:multiplication" />
      <xsd:element name="multiplicationResponse"
        type="tns:multiplicationResponse" />
      <xsd:complexType name="addition">
        <xsd:sequence>
          <xsd:element name="a" type="xsd:double"></xsd:element>
          <xsd:element name="b" type="xsd:double"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="additionResponse">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:double" />
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="multiplication">
        <xsd:sequence>
          <xsd:element name="a" type="xsd:double"></xsd:element>
          <xsd:element name="b" type="xsd:double"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="multiplicationResponse">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:double" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <message name="multiplication">
    <part name="parameters" element="tns:multiplication"></part>
  </message>
  <message name="multiplicationResponse">
    <part name="parameters" element="tns:multiplicationResponse"></part>
  </message>
  <message name="addition">
    <part name="parameters" element="tns:addition"></part>
  </message>
  <message name="additionResponse">
    <part name="parameters" element="tns:additionResponse"></part>
  </message>

```



```

</message>
<portType name="SimpleCalculator">
  <operation name="multiplication">
    <input message="tns:multiplication"></input>
    <output message="tns:multiplicationResponse"></output>
  </operation>
  <operation name="addition">
    <input message="tns:addition"></input>
    <output message="tns:additionResponse"></output>
  </operation>
</portType>

<!-- binding & service facultatif pour usage BPEL -->

<plnk:partnerLinkType name="SimpleCalculatorPartnerLinkType">
  <plnk:role name="SimpleCalculator" portType="tns:SimpleCalculator"/>
</plnk:partnerLinkType>

```

```
</definitions>
```

MyCalculator.wsdl (service rendu par processus bpe)

```

<?xml version="1.0"?>
<wsdl:definitions name="MyCalculator"
  targetNamespace="urn:/MyCalculator.wsdl"
  xmlns:tns="urn:/MyCalculator.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:smix="http://servicemix.org/wsdl/jbi/">

  <wsdl:types>
    <xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
      targetNamespace="urn:/MyCalculator.wsdl"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="urn:/MyCalculator.wsdl">
      <xsd:element name="axb" type="tns:axbRequest"/>
      <xsd:element name="axbResponse" type="tns:axbResponse" />
      <xsd:complexType name="axbRequest">
        <xsd:sequence>
          <xsd:element name="a" type="xsd:double"/>
          <xsd:element name="x" type="xsd:double"/>
          <xsd:element name="b" type="xsd:double"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="axbResponse">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:double"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>

```

```

</wsdl:types>

<wsdl:message name="axbRequest">
  <wsdl:part name="parameters" element="tns:axb"/>
</wsdl:message>

<wsdl:message name="axbResponse">
  <wsdl:part name="parameters" element="tns:axbResponse"/>
</wsdl:message>

<wsdl:portType name="MyCalculatorPortType">
  <wsdl:operation name="axb">
    <wsdl:input message="tns:axbRequest" name="axb"/>
    <wsdl:output message="tns:axbResponse" name="axbResponse"/>
  </wsdl:operation>
</wsdl:portType>

<plnk:partnerLinkType name="MyCalculatorPartnerLinkType">
  <plnk:role name="Provider" portType="tns:MyCalculatorPortType"/>
</plnk:partnerLinkType>

<!--

This is an abstract interface/portType definition. Note the lack of
binding and service: these are defined by the HTTP binding component.
See MyCalculator-http/MyCalculator.wsdl for details.

-->

</wsdl:definitions>

```

MyCalcultor.bpel (calcul $a \cdot x + b$ en invoquant multiplication puis addition)

NB: les parties <!-- <bpel:copy> <bpel:from> <bpel:literal> --> ne sont nécessaires que pour certains moteurs BPEL tel que ODE .

```

<?xml version="1.0"?>
<bpel:process name="MyCalculatorBpel"
  targetNamespace="urn:/HeaderTest.bpel"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:tns="urn:/HeaderTest.bpel"
  xmlns:basic="http://basic/"
  xmlns:MyCalculator="urn:/MyCalculator.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <bpel:import location="MyCalculator.wsdl"
    namespace="urn:/MyCalculator.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <bpel:import location="SimpleCalculator.wsdl"
    namespace="http://basic/"
    importType="http://schemas.xmlsoap.org/wsdl/" />

  <bpel:partnerLinks>

```

```

<bpel:partnerLink name="MyCalculatorPartnerLink"
  partnerLinkType="MyCalculator:MyCalculatorPartnerLinkType"
  myRole="Provider" />
<bpel:partnerLink name="SimpleCalculatorPartnerLink"
  partnerLinkType="basic:SimpleCalculatorPartnerLinkType"
  partnerRole="SimpleCalculator"/>
</bpel:partnerLinks>

<bpel:variables>
  <bpel:variable name="axbRequest" messageType="MyCalculator:axbRequest"/>
  <bpel:variable name="axbResponse"
    messageType="MyCalculator:axbResponse"/>
  <bpel:variable name="addRequest" messageType="basic:addition"/>
  <bpel:variable name="addResponse" messageType="basic:additionResponse"/>
  <bpel:variable name="multRequest" messageType="basic:multiplication"/>
  <bpel:variable name="multResponse"
    messageType="basic:multiplicationResponse"/>
</bpel:variables>

<bpel:sequence>
  <bpel:receive
    name="start"
    partnerLink="MyCalculatorPartnerLink"
    portType="MyCalculator:MyCalculatorPortType"
    operation="axb"
    variable="axbRequest"
    createInstance="yes"/>

  <bpel:assign name="assign_ax">
    <!-- <bpel:copy>
      <bpel:from>
        <bpel:literal>
          <basic:multiplication>
            <a/>
            <b/>
          </basic:multiplication>
        </bpel:literal>
      </bpel:from>
      <bpel:to>$multRequest.parameters</bpel:to>
    </bpel:copy> -->
    <bpel:copy>
      <bpel:from>$axbRequest.parameters/MyCalculator:a</bpel:from>
      <bpel:to>$multRequest.parameters/a</bpel:to>
    </bpel:copy>
    <bpel:copy>
      <bpel:from>$axbRequest.parameters/MyCalculator:x</bpel:from>
      <bpel:to>$multRequest.parameters/b</bpel:to>
    </bpel:copy>
  </bpel:assign>

  <bpel:invoke name="invokeMult"
    partnerLink="SimpleCalculatorPartnerLink"
    operation="multiplication"
    portType="basic:SimpleCalculatorPortType"

```

```

    inputVariable="multRequest"
    outputVariable="multResponse"/>

<bpel:assign name="assign_plusb">
  <!-- <bpel:copy>
    <bpel:from>
      <bpel:literal>
        <basic:addition>
          <a/>
          <b/>
        </basic:addition>
      </bpel:literal>
    </bpel:from>
    <bpel:to>$addRequest.parameters</bpel:to>
  </bpel:copy> -->
  <bpel:copy>
    <bpel:from>$multResponse.parameters/return</bpel:from>
    <bpel:to>$addRequest.parameters/a</bpel:to>
  </bpel:copy>
  <bpel:copy>
    <bpel:from>$axbRequest.parameters/MyCalculator:b</bpel:from>
    <bpel:to>$addRequest.parameters/b</bpel:to>
  </bpel:copy>
</bpel:assign>

<bpel:invoke name="invokeAdd"
  partnerLink="SimpleCalculatorPartnerLink"
  operation="addition"
  portType="basic:SimpleCalculatorPortType"
  inputVariable="addRequest"
  outputVariable="addResponse"/>

<bpel:assign name="assign_res">
  <!-- <bpel:copy>
    <bpel:from>
      <bpel:literal>
        <MyCalculator:axbResponse>
          <MyCalculator:return/>
        </MyCalculator:axbResponse>
      </bpel:literal>
    </bpel:from>
    <bpel:to>$axbResponse.parameters</bpel:to>
  </bpel:copy> -->
  <bpel:copy>
    <bpel:from>$addResponse.parameters/return</bpel:from>
    <bpel:to>$axbResponse.parameters/MyCalculator:return</bpel:to>
  </bpel:copy>
</bpel:assign>

<bpel:reply name="end"
  partnerLink="MyCalculatorPartnerLink"
  portType="MyCalculator:MyCalculatorPortType"
  operation="axb"
  variable="axbResponse"/>
</bpel:sequence>

```

</bpel:process>

Remarque importante:

dans l'écriture

```
<bpel:copy>
  <bpel:from>$axbRequest.parameters/MyCalculator:a</bpel:from>
    <bpel:to>$multRequest.parameters/a</bpel:to>
</bpel:copy>
```

a est préfixé par MyCalculator (préfixe local associé au namespace "urn:/MyCalculator.wsdl") car le schéma xsd inclus dans MyCalculator.wsdl comporte la propriété **elementFormDefault="qualified"**

et à l'inverse a n'est pas préfixé dans \$multRequest.parameters/a car le schéma xsd inclus dans SimpleCalculator.wsdl comporte (explicitement ou implicitement par défaut) la propriété **elementFormDefault="unqualified"**

IX - ODE et BPEL_Designer

1. ODE (Moteur BPEL d'Apache)

Le moteur **ODE** existe en 2 versions:

- une version sous forme d'application java/web (.war) à installer dans un conteneur web (ex: tomcat)
- une version sous forme de "Service_Engine" JBI à installer dans un ESB JBI tel que ServiceMix .

1.1. Installation du moteur ODE (en version web) sous Tomcat

Télécharger **ODE (en version war)** et extraire le contenu de l'archive ZIP sur c:\

recopier **ode.war** de [c:\apache-ode-war-1.3.5](#) vers [c:/.../Tomcat5_ou_6/webapps](#)

Vérifier l'installation via l'url suivante: <http://localhost:8080/ode>

Structure et déploiement d'un processus métier (bpel)

Le déploiement d'un processus bpel au sein de "Tomcat + ODE" s'effectue en déposant un nouveau sous répertoire **my_xxx_process** (de nom quelconque) dans le répertoire **Tomcat/webapps/ode/WEB-INF/processes** .

Pour structurer ce répertoire, il faut suffir d'adopter les conventions/dispositions suivantes:

my_xxx_process

my_process.bpel

deploy.xml

service1.wsdl

service2.wsdl

Seul le fichier **deploy.xml** est spécifique à ODE.

1.2. Installation du moteur ODE (version jbi) dans ServiceMix 3.x

Démarrer **ServiceMix 3.3**

Recopier le fichier **ode-jbi-1.2.zip** dans le répertoire **hotdeploy** de **servicemix 3.3**

Pour arrêter l'ensemble --> surtout pas de ctrl-c mais fermer simplement la fenêtre servicemix.

NB: a chaque redémarrage de "servicemix+ode", il est souhaitable de "désinstaller puis réinstaller" les "service_assembly" (xxx_sa.zip) de façon à ce que les éléments soient bien synchronisés pour garantir un fonctionnement correct .

NB: il vaut mieux intégrer les schémas dans le WSDL car les chemins relatifs vers des sous fichiers ".xsd" sont mal interprété avec la version ODE 1.2 .

1.3. Installation du moteur ODE (version jbi) dans ServiceMix 4.x

Les versions 4.x de serviceMix sont basées sur OSGi . Cela implique une installation particulière de ODE :

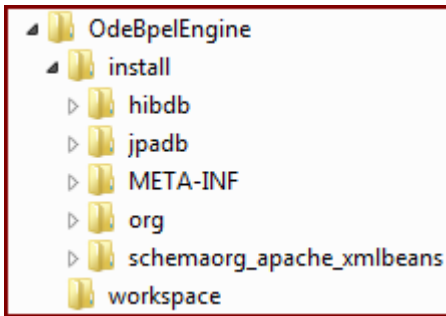
Démarrer **ServiceMix 4.3** (via *bin/servicemix.bat*)

Au sein de la console OSGi de serviceMix , recopier et lancer une par une les deux commandes suivantes:

```
features:addUrl mvn:org.apache.ode/ode-jbi-karaf/1.3.5/xml/features
```

```
features:install ode
```

Pour vérifier l'installation de ODE au sein de servicemix 4 , on peut vérifier la présence des nouveaux répertoires suivants au sein du répertoire **data/jbi** de servicemix :



1.4. Exemple de fichier deploy.xml (spécifique à ODE)

```
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
  xmlns:pns="urn:/HeaderTest.bpel"
  xmlns:sns="urn:/MyCalculator.wSDL"
  xmlns:basic="http://basic/" >

  <process name="pns:MyCalculatorBpel">
    <active>true</active>
    <provide partnerLink="MyCalculatorPartnerLink">
      <service name="sns:MyCalculatorService" port="MyCalculatorPort"/>
    </provide>
    <invoke partnerLink="SimpleCalculatorPartnerLink">
      <service name="basic:SimpleCalculatorImplService"
        port="SimpleCalculatorImplPort"/>
    </invoke>
  </process>
</deploy>
```

2. Plugin eclipse "BPEL Designer"

2.1. Installation du plugin

En juillet 2011 , l'URL de l'update site (pour eclipse 3.6) est

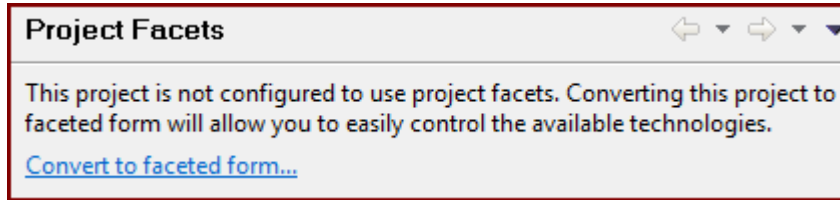
<http://download.eclipse.org/technology/bpel/update-site/>

--> Installation classique via le menu "Help / Software update / find and Install ..."

--> Prévoir un redémarrage d'eclipse en fin d'installation du plugin .

2.2. Utilisation du plugin "BPEL designer"

NB: Les assistants du plugin "BPEL designer" ne fonctionnent correctement qu'au sein d'un projet eclipse ayant la facette technologique "BPEL 2.0 Facet"

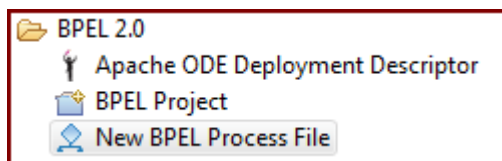


Project Facet	Version	
<input type="checkbox"/> Application Client module	6.0	▼
<input type="checkbox"/> Axis2 Web Services		
<input checked="" type="checkbox"/> BPEL 2.0 Facet	2.0	
<input type="checkbox"/> CXF 2.x Web Services	1.0	
<input type="checkbox"/> Dynamic Web Module	3.0	▼
<input type="checkbox"/> EAR	6.0	▼
<input type="checkbox"/> EJB Module	3.1	▼
<input type="checkbox"/> EJBDoclet (XDoclet)	1.2.3	▼
<input checked="" type="checkbox"/> Java	1.6	▼

NB: Le répertoire "**bpelContent**" créé par le plugin pour ranger les fichier "deploy.xml", xxx.wsdl et xxx.bpel" n'est à utiliser qu'au sein d'un projet qui n'est pas pris en charge par maven.

Au sein d'un projet "**maven**", les fichiers de configurations ".xml" sont attendus dans le répertoire "**src/main/resources**".

Pour déclencher les assistants du plugin "**BPEL Designer**", il faut activer le menu "**File/ New .. / Other ...**" / **BPEL 2 / new BPEL Process File**" ou bien ouvrir un fichier ".bpel" existant .



NB: dans le cas où l'on choisit le modèle "*synchrone*", l'assistant "*new BPEL process File*" crée un fichier xxx.bpel et un fichier "xxxArtifacts.wsdl" correspondant à un début de "*Hello world*".

Bien que *basique*, ce point de départ est *syntactiquement intéressant* .

2.3. Edition préalable d'un schéma XSD

Les structures de données manipulées par un processus BPEL sont définies (*en mode document/literal*) dans la partie "schema" d'un fichier WSDL.

Cette partie "schema" peut être soit directement incluse dans la partie haute d'un fichier WSDL, soit placée dans un "sous fichier" XSD lui-même importé dans un fichier WSDL .

Attention: Les "sous fichiers XSD" ne sont pas supportés par tous les environnements d'exécution

(Moteur BPEL , ESB , ...) . A vérifier !!! (par exemple ODE 1.3.5 supporte bien des fichiers XSD séparés tandis que "servicemix-http" ne le gère pas bien).

Néanmoins, lorsque les sous fichiers ".XSD" sont supportés , la structuration est globalement plus claire.

Dans ces deux cas, eclipse offre des assistants pour éditer un schéma XSD :

- onglet "Design" (pour la visualisation)
- onglet "Xml" et vues "outline" et "properties" (pour les modifications)

Rappel: pour une méthode *xxx(p1,p2)* la structure attendue dans le schéma est:

- 1 complexType "Xxx" avec sequence d'éléments "p1" et "p2" (avec types)
- 1 complexType "XxxResponse" avec element "return" typé dans une sequence
- 1 element "xxx" de type "tns:Xxx"
- 1 element "xxxResponse" de type "tsn:XxxResponse"

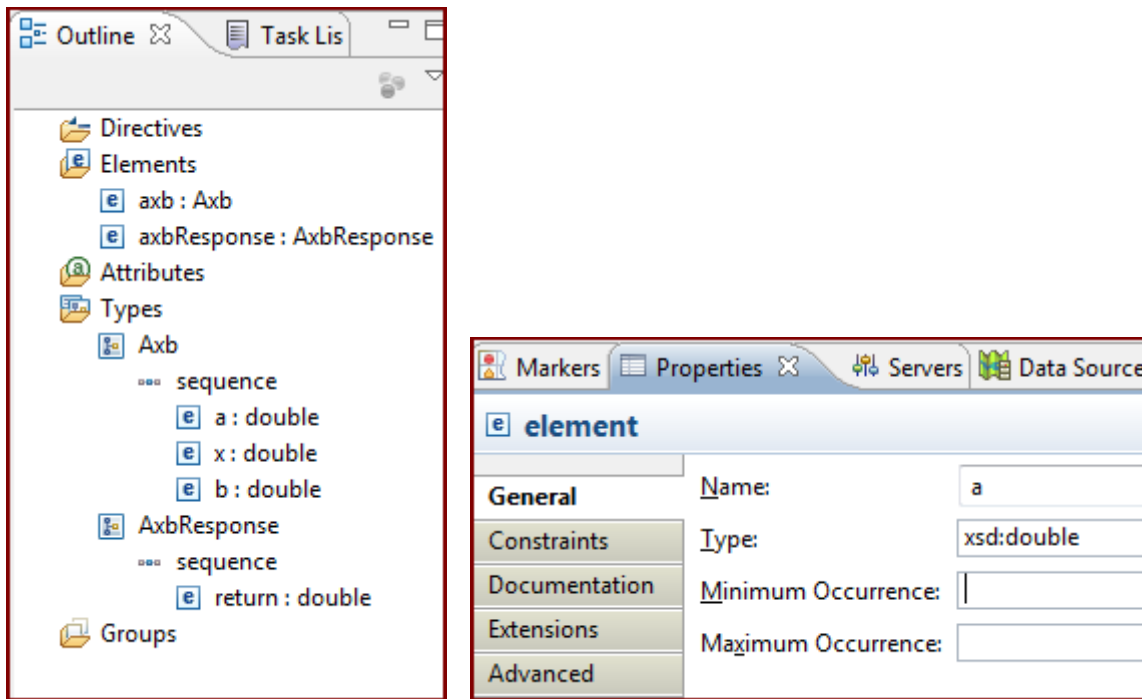
Exemple:

New / other .../ XML / Xml Schema (si nécessaire)

axb.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema attributeFormDefault="unqualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="http://axb/" xmlns:tns="http://axb/">
  <xsd:element name="axb" type="tns:Axb" />
  <xsd:element name="axbResponse" type="tns:AxbResponse" />
  <xsd:complexType name="Axb">
    <xsd:sequence>
      <xsd:element name="a" type="xsd:double" />
      <xsd:element name="x" type="xsd:double" />
      <xsd:element name="b" type="xsd:double" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="AxbResponse">
    <xsd:sequence>
      <xsd:element name="return" type="xsd:double" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Vues "outline" et "properties"

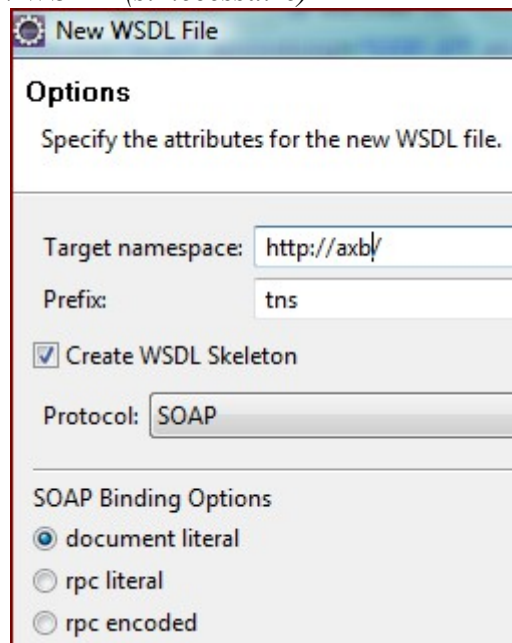


NB: depuis un élément sélectionné dans la vue outline, on peut déclencher tout un tas de menus contextuels "add ...".

2.4. Edition préalable d'un fichier WSDL

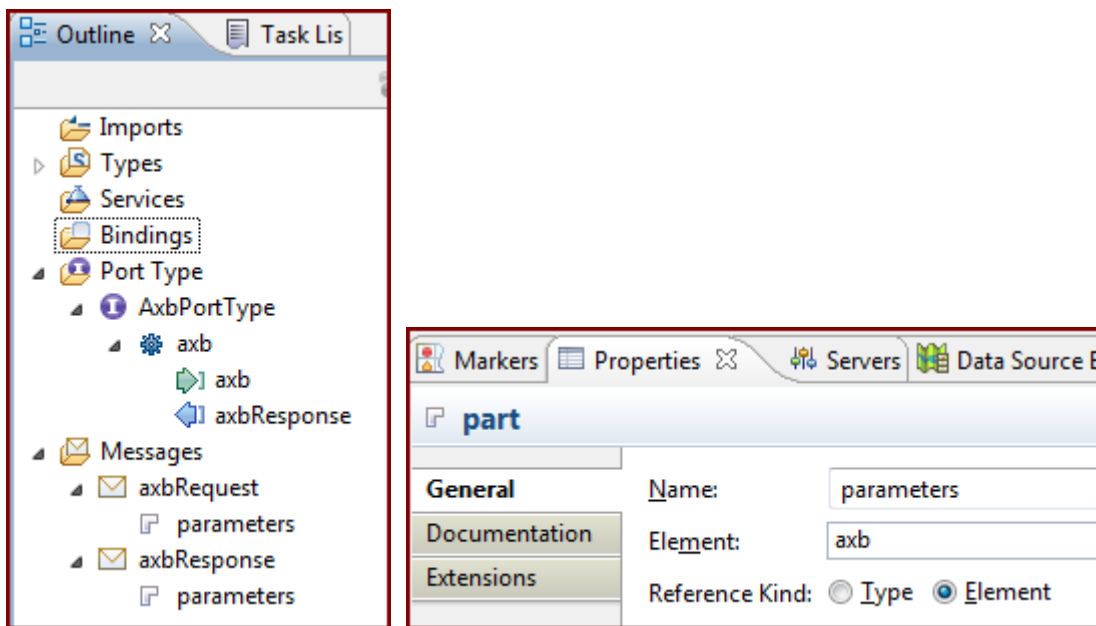
En combinant une édition directe XML (pratique pour le copier/coller et le rechercher/remplacer), et une édition indirecte via les vues "outline" et "properties", on peut rapidement générer un fichier WSDL abstrait nécessaire à un processus BPEL.

New / Other ... / WebServices / WSDL (si nécessaire)



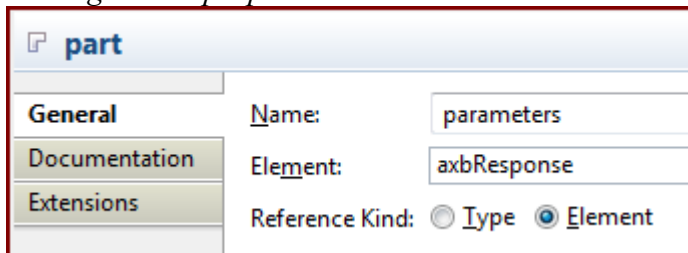
NB: L'assistant "new WSDL" génère un fichier WSDL exemple avec un embryon de toutes les parties.

Il est conseillé d'éditer le fichier en peaufinant très bien la partie haute (types/schema) puis en rédéfinissant/reconstruisant ensuite les parties basses en fonction de la partie haute.



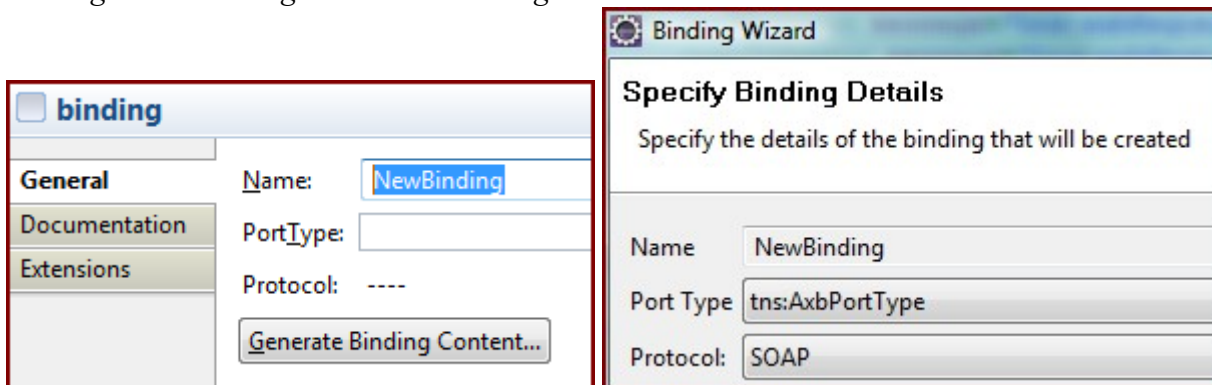
Assistants particulièrement utiles (WSDL):

Message Part "properties":



Attention : "Add PortType" génère quelquefois trop de choses (nouveau schema + nouveaux messages + nouveau PortType)

Bindings/ Add Binding / Generate Binding Content ...



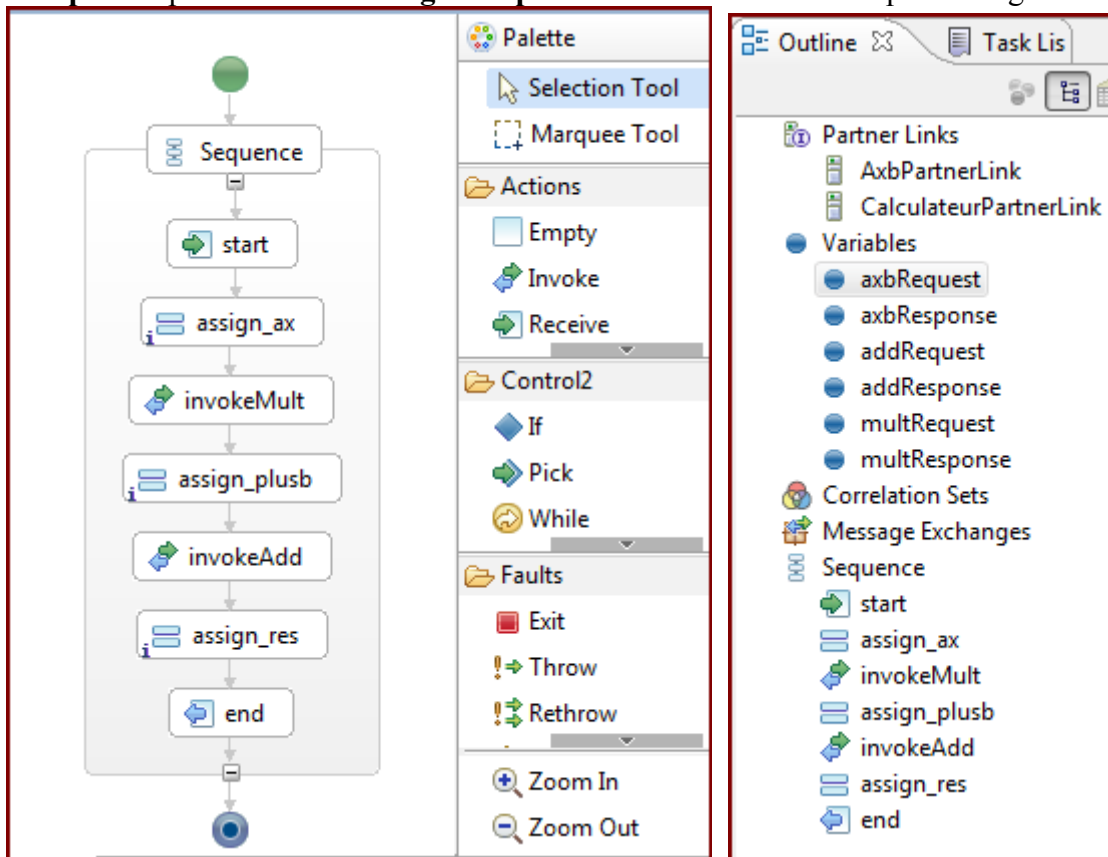
Services / Add Service / ... (ajoute un port en même temps)

Paramétrage du port (via la vue "properties"):

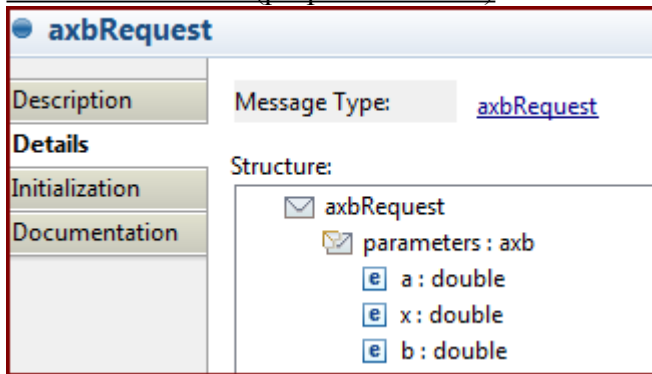
port	
General	Name: AxbPortEsb
Documentation	Binding: AxbServiceSoapBinding
Extensions	Address: http://localhost:8181/AxbPortEsb/
	Protocol: SOAP

2.5. Edition d'un processus BPEL

Une sous **vue graphique** du processus ("design"),
une **palette** pour effectuer des "**glisser/poser**" et une vue "**outline**" pour naviguer dans le "bpel".

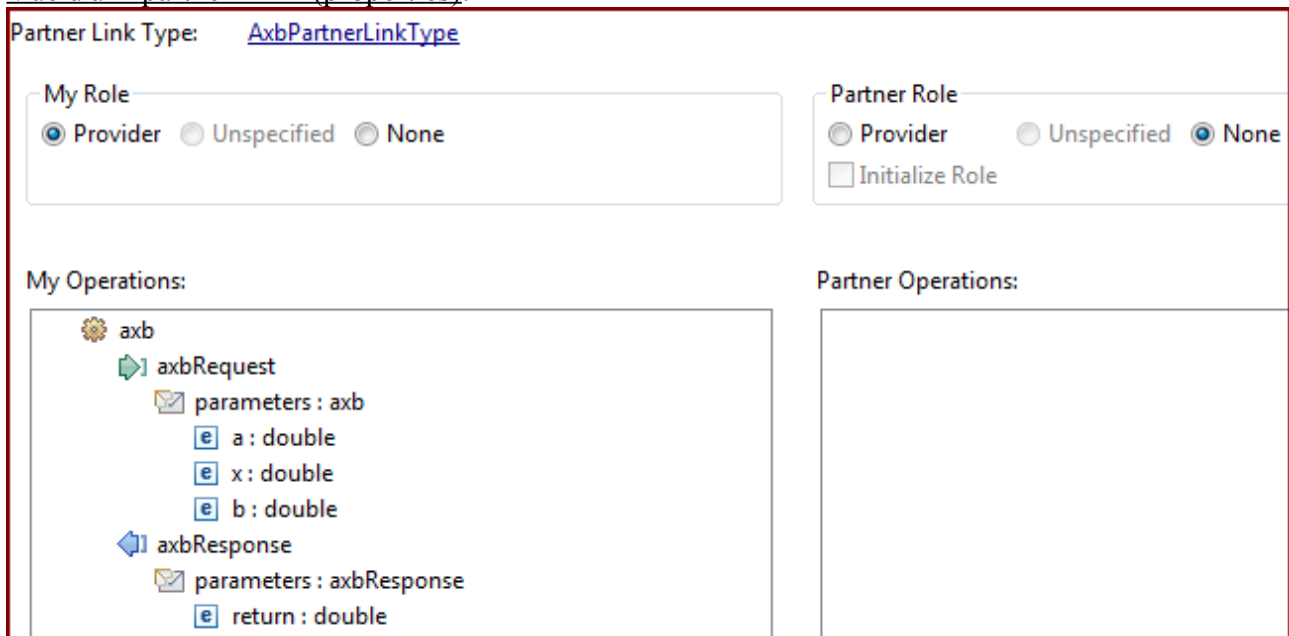


Vue d'une variable (properties/details):



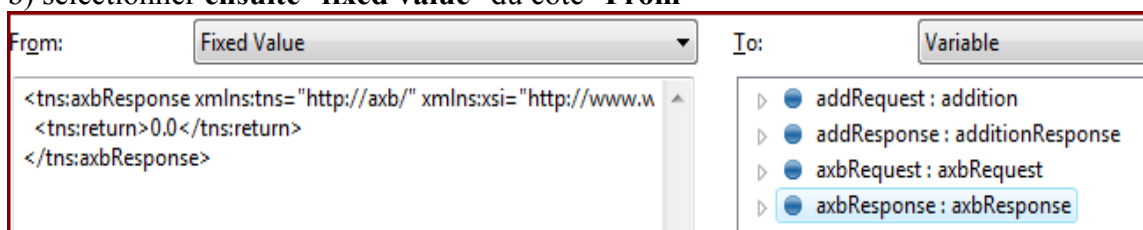
Attention : les **variables bpel** doivent **basées** sur les **messages wsdl** (pas sur les éléments xsd) .
 → vérifier un code source de type `<bpel:variable messageType="..." />` .

Vue d'un "partnerLink" (properties):



2.6. Zoom sur l'edition d'une affectation (assign)

- 1) Déposer un "assign" via un glisser/poser , Renommer cet "assign"
- 2) Dans **properties/details** , New
- 3) Si nécessaire (selon moteur BPEL) pour initialiser une variable encore jamais utilisée:
 - a) **sélectionner d'abord une variable du côté 'To' et la sous partie "payload" ou "parameters"**
 - b) **sélectionner ensuite "fixed value" du côté "From"**



- 4) New , From (**expression**) , To (**expression**)

puis saisir ensuite les expressions avec l'auto-complétion (Ctrl-Espace)

Exemple: *Ctrl-Espace* , puis choix d'une variable

==> `$axbRequest`

Saisir "."

==> `$axbRequest.parameters` ou `$axbRequest.payload` ou ...

Saisir éventuellement "/" puis choisir une sous partie (qualifiée ou pas)

==> `$axbRequest.parameters/namespace:a` ou `$axbRequest.parameters/a`

Remarque importante:

En appuyant plusieurs fois de suite sur "Ctrl-Space" on permute le mode d'auto-complétion :

- choix de variable
- choix d'opérateur (+, -, *, div, mod, and, or,)
- choix d'expression XPath 1.0 (concat, substring, round,)

Type de Code généré:

```
<bpel:assign name="assign_res" validate="no">
  <bpel:copy>
    <bpel:from>
      <bpel:literal>
        <axb:axbResponse xmlns:axb="http://axb/">
          <axb:return/>
        </axb:axbResponse>
      </bpel:literal>
    </bpel:from>
    <bpel:to><![CDATA[$axbResponse.parameters]]></bpel:to>
  </bpel:copy>
  <bpel:copy>
    <bpel:from>
      <![CDATA[$addResponse.parameters/return]]>
    </bpel:from>
    <bpel:to><![CDATA[$axbResponse.parameters/axb:return]]></bpel:to>
  </bpel:copy>
</bpel:assign>
```

Dans le cas d'une utilisation d'ODE , il est préférable d'ouvrir le fichier **deploy.xml** via "Open with / XML Editor" car l'éditeur spécialisé ODE n'est surtout utile qu'en visualisation .

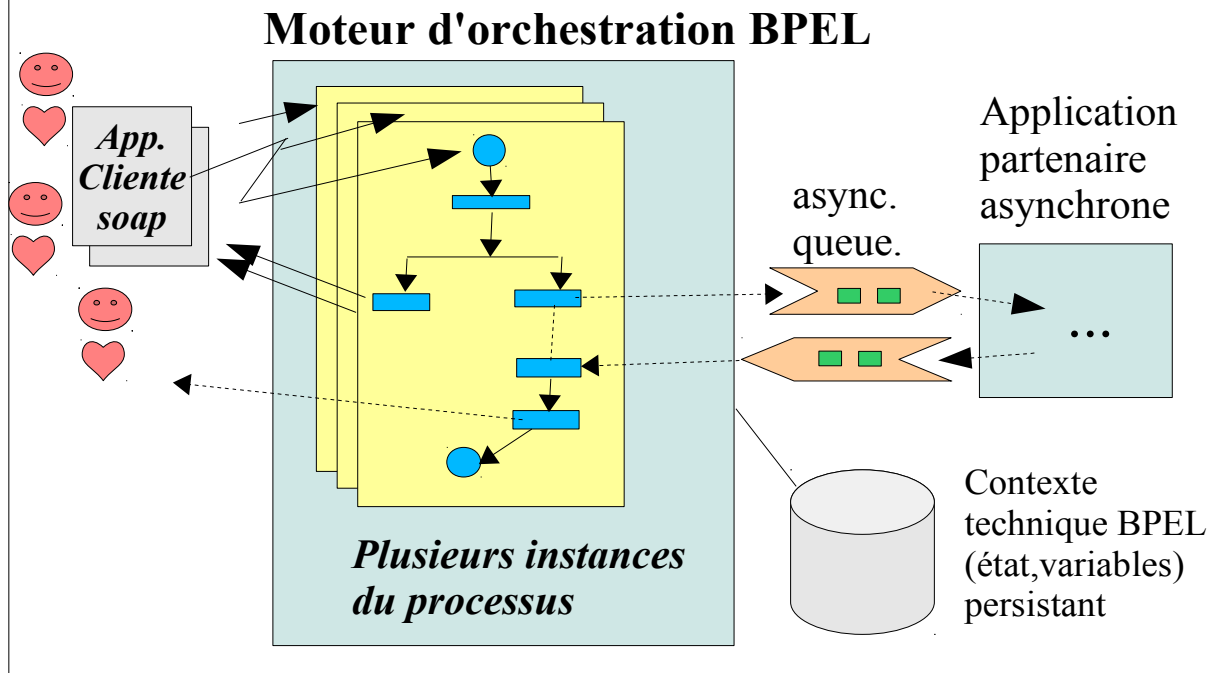
Mode opératoire global (l'ordre est important) :

- 1) préparer tous les fichiers WSDL (avec parties `plnk:partnerLinkType` et `xmlns:plnk`)
- 2) ajouter les `<bpel:import "....wsdl" />` dans le haut du fichier `bpel`
- 3) ajouter/paramétrer les "partnerLink" (avec `myRole="roleBpel"` ou `partnerRole="RoleExterne"`)
- 4) ajouter les **variables "bpel" basées sur des types de messages WSDL** (variables `xxxInput` et `xxxOutput` pour chaque opération à invoquer)
- 5) ajouter et paramétrer tous les "invoke ..." dans une séquence du BPEL
- 6) ajouter et paramétrer tous les "assign / copy/from" selon l'algorithme du processus.
(Bien penser à initialiser la variable "output" du reply du processus `bpel` et les variables "xxxInput" de chaque opération invoquée : To = variable/payload , From=fixed_value).

X - BPEL asynchrone

1. BPEL en mode asynchrone

BPEL (dans tout son potentiel) en mode asynchrone



BPEL - mode asynchrone et corrélations

```
<invoke operation="xxx" ...>
  <correlations>
    <correlation set="xxxCorSet"
      initiate="yes" />
  </correlations> ... </invoke>
```

one-way

request(s)

```
<pick> <onMessage
  operation="xxxRespCallBack">
    <correlations>
      <correlation set="xxxCorSet"
        initiate="no" />
    </correlations> ... </onMessage>
```

one-way

response(s)

BPEL – *pick* (receive (if correlation) ... or)

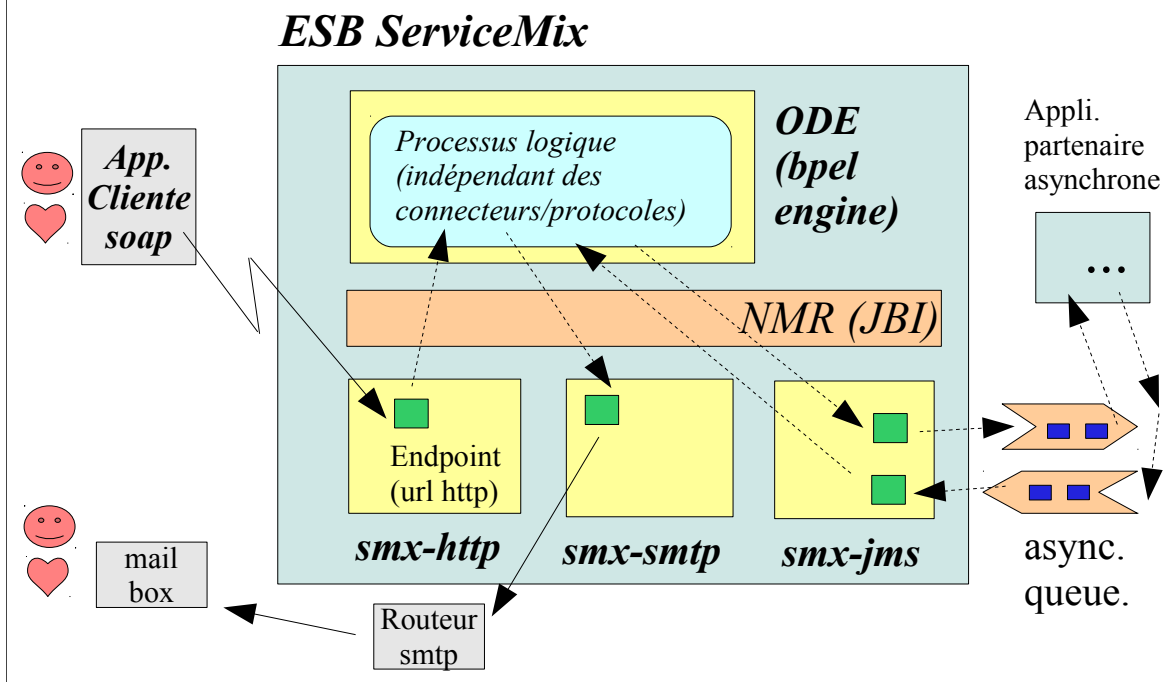
```

<pick>
  <onMessage operation="xxxResponse"
    ....> <correlations> ...</correlations>
    <!-- gérer la réponse positive (acceptation) -->
  </onMessage>
  <onMessage operation="xxxReject"
    ....> <correlations> ...</...>
    <!-- gérer le refus -->
  </onMessage>
  ...
</pick>

```

callback

Ex. d'intégration 2: BPEL asynchrone via ODE dans ServiceMix



1.1. Propriétés de corrélations pour BPEL asynchrone

BPEL – correlation property (partie WSDL)

abstract_properties.wsdl (imported)

```
<wsdl:definitions name="properties"
targetNamespace="http://xyz/supplyCorrelation" ...>
<vprop:property name="customerID" type="xsd:string"/>
<vprop:property name="orderNumber" type="xsd:int" >
</wsdl:definitions>
```

```
<wsdl:definitions ...
xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop" >
  <wsdl:import namespace=".../supplyCorrelation"
    location="abstract_properties.wsdl" /> ...
  <vprop:propertyAlias propertyName="cor:customerID"
    messageType="tns:POMessage" part="PO">
    <bpel:query>ClientID</bpel:query>
    <!-- element name or xpath expr -->
  </vprop:propertyAlias> ...</wsdl:definitions>
```

BPEL – correlationSet (partie bpel)

```
<process>
  <variables> ... </variables>
  <correlationSets
    xmlns:cor="http://xyz/supplyCorrelation">
      <correlationSet name="PurchaseOrder"
        properties="cor:customerID
          cor:orderNumber" />
    ... </correlationSets>
</process>
```

* Les valeurs (propriétés) *customerID* et *orderNumber* doivent rester les mêmes (entre requête et réponse asynchrones)

NB: 1 propriété--> n alias possibles --> n structures XML .

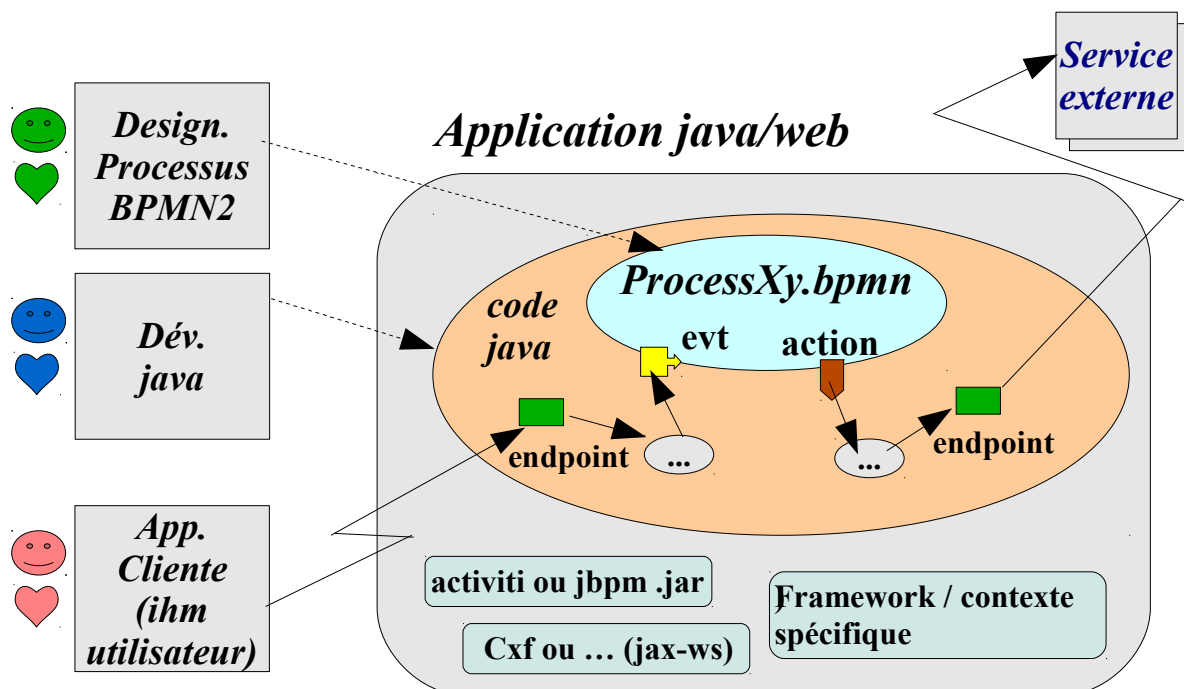
XI - Bpmn2 , activiti , jbpm (présentation)

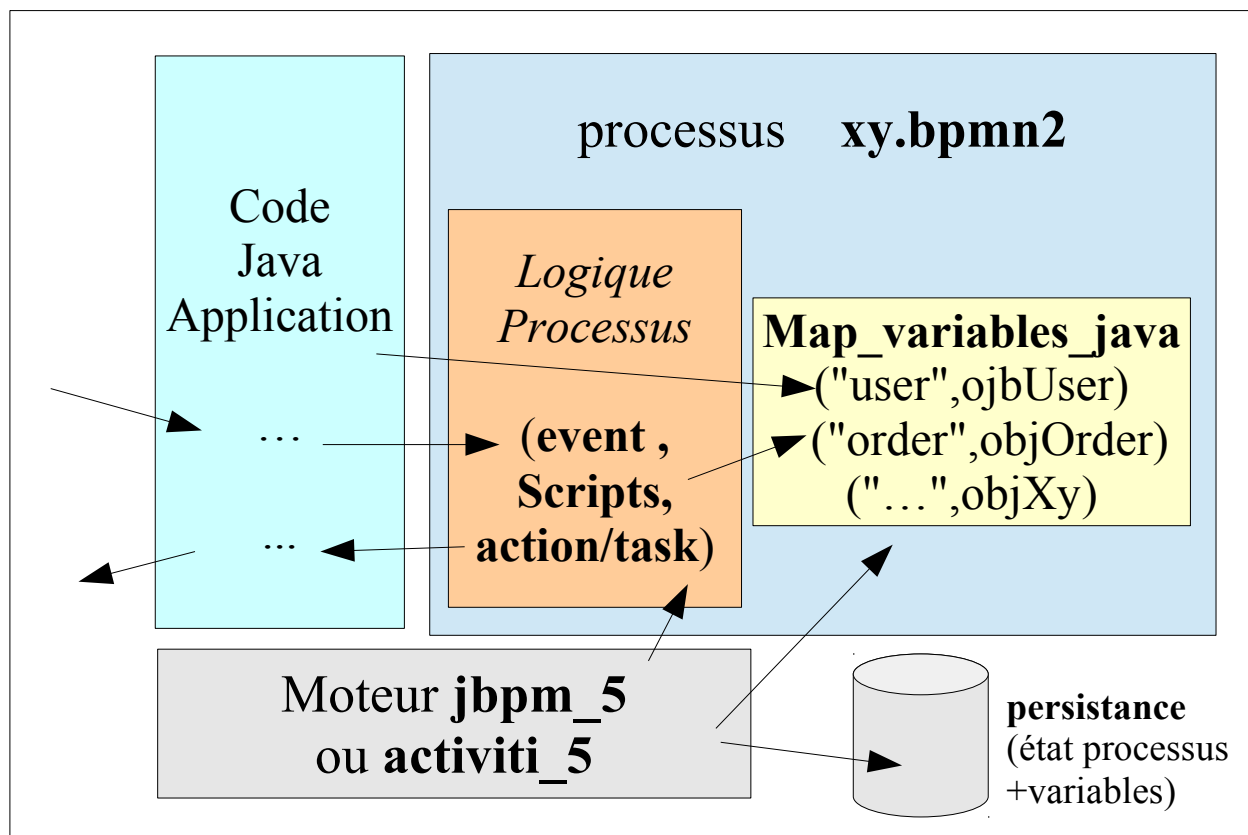
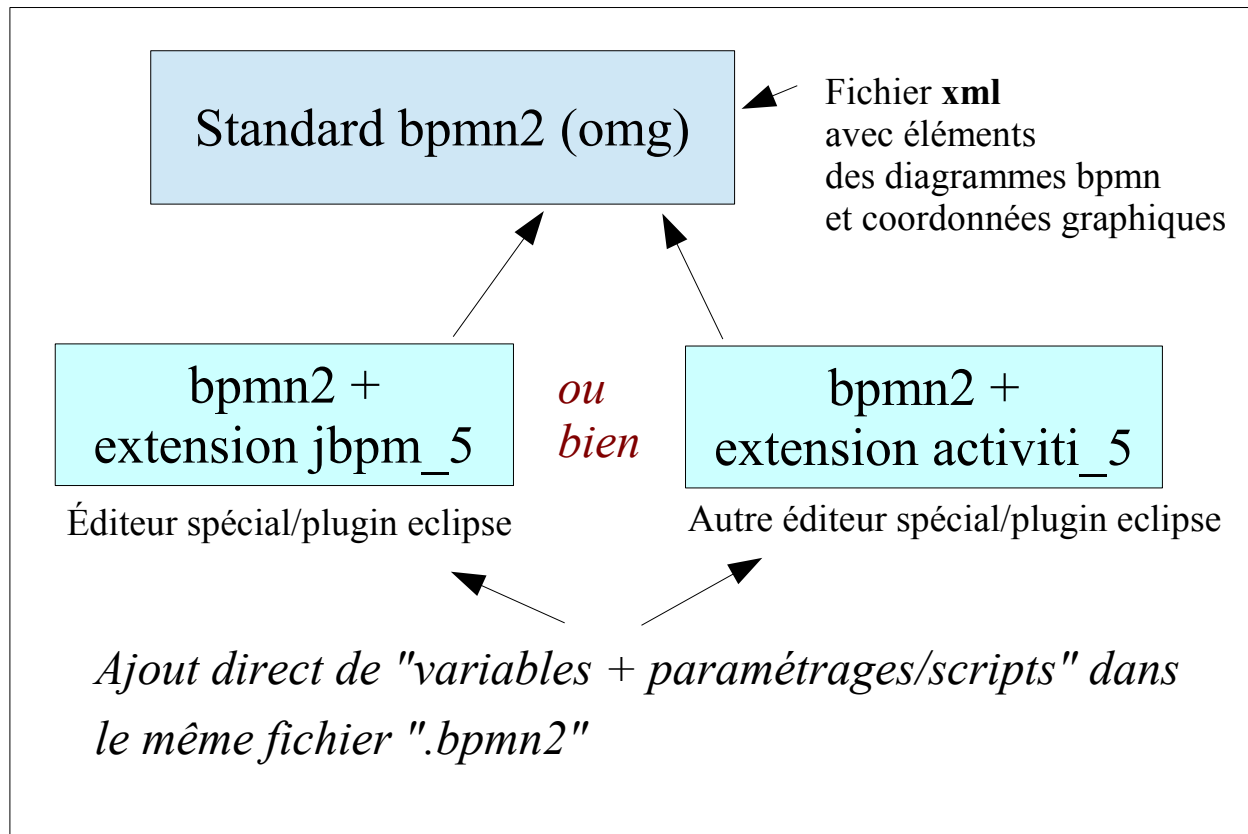
1. jBPM et activiti (présentation)

Eventuelle alternative (java/BPM) à BPEL (xml)

- * **jBPM_5** et **activiti_5** sont des **moteurs de workflow** à intégrer des applications **java** (sous forme de ".jar").
- * *Ces deux technologies (très proches) s'appuient directement sur le standard **BPMN2**.*
- * **jBPM_5** est en fait un des constituants de **Drools 5** et s'intègre facilement dans **Jboss 7.1** ou **Jboss EAP 6**
- * **activiti_5** est plus léger et peut facilement s'intégrer dans le framework **Spring** (et donc dans **tomcat 7**)
- * Bien que nécessitant pas mal de code java (variables, invocations,...), jBPM5 et activiti peuvent être utilisés pour effectuer de l'orchestration de services.

Principe de fonctionnement de (java/BPM)





XII - Activiti (bpmn2/java , intégration Spring)

1. Activiti (présentation détaillée)

Activiti est une technologie de type "**bpm**" (*business process management*) basée à **java** et **bpmn2** .

Activiti ressemble beaucoup à jbpm (de Jboss) mais est basé sur un **cœur complètement indépendant** (pas de dépendance vis à vis de drools ou autres).

Activiti peut éventuellement être utilisé seul mais l'**intégration au sein de spring** apporte beaucoup d'avantages. il serait dommage de les ignorer.

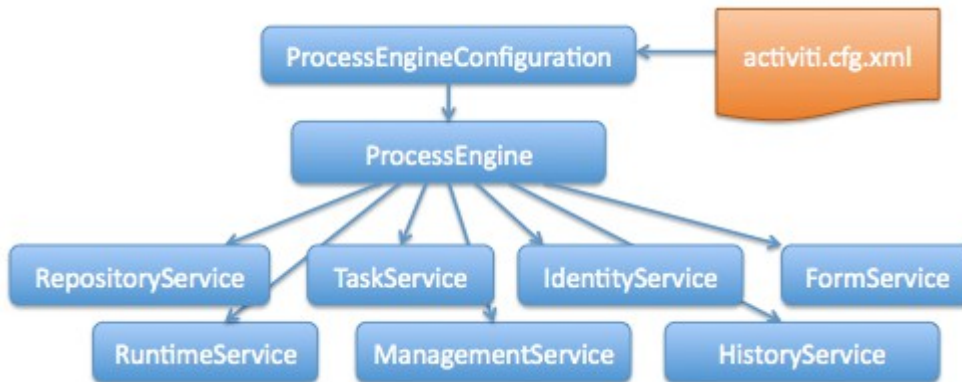
Etant constitué d'une simple **collection de ".jar"** , activiti peut être intégré un peu partout (dans une application java/web pour tomcat ou dans un ESB de type "serviceMix" ou "muleESB").

Une **documentation détaillé d'activiti** est accessible au bout de l'URL suivante

<http://www.activiti.org/userguide/>

1.1. Services techniques d'activiti

L'une des principales particularités de la technologie "activiti" est son **architecture à base de services techniques** :



L'accès à chacun de ces services techniques d'activiti s'effectue simplement comme ceci :

```

ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

RuntimeService runtimeService = processEngine.getRuntimeService();
RepositoryService repositoryService = processEngine.getRepositoryService();
TaskService taskService = processEngine.getTaskService();
ManagementService managementService = processEngine.getManagementService();
IdentityService identityService = processEngine.getIdentityService();
HistoryService historyService = processEngine.getHistoryService();
FormService formService = processEngine.getFormService();
  
```

Service technique d'activiti	utilité
------------------------------	---------

RepositoryService	Pour gérer la persistance (base de données) et pour charger le processus (fichier .bpmn2) en mémoire
RuntimeService	Démarrer , contrôler une instance du processus

1.2. Configuration et chargement/démarrage

Fichier de configuration "activiti.cfg.xml"

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="processEngineConfiguration"
class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">

        <property name="jdbcUrl"
value="jdbc:h2:mem:activiti;DB_CLOSE_DELAY=1000" />
        <property name="jdbcDriver" value="org.h2.Driver" />
        <property name="jdbcUsername" value="sa" />
        <property name="jdbcPassword" value="" />

        <property name="databaseSchemaUpdate" value="true" />

        <property name="jobExecutorActivate" value="false" />

    </bean>
</beans>
```

NB : ce fichier peut quelquefois être remplacé par une configuration "spring" équivalente.

Chargement (sans spring) d'un processus activiti et démarrage d'une instance :

```
public static void main(String[] args) {
    try{
        Map<String,Object> params = new HashMap<String,Object>();
        //NB: each param/variable (with all sub-objects) must be serializable !!!
        params.put("username","toto");    params.put("age",new Integer(12));
        params.put("pers", new Personne("toto",44L));

        // Create Activiti process engine
        ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

        // Get Activiti services
        RepositoryService repositoryService = processEngine.getRepositoryService();
        RuntimeService activitiRuntimeService = processEngine.getRuntimeService();

        // Deploy the process definition
        repositoryService.createDeployment()
        .addClasspathResource("test_activiti_with_props.bpmn").deploy();

        ProcessInstance activitiProcessInstance =
```

```

    activitiRuntimeService.startProcessInstanceByKey("myProcess", params);
    String processInstanceId=activitiProcessInstance.getId();
    System.out.println("started new processInstanceId="+processInstanceId);
  } catch (Exception e) {
    e.printStackTrace();
  }
}

```

1.3. Syntaxes au sein du processus ".bpmn2" pour activiti :

pour:

```

params.put("username","toto");
params.put("age",new Integer(12));

```

et pour "aCtx" = nom d'un composant (Spring ou ...) accessible

scriptTask (javascript):

```
aCtx.doAction(execution,"debut " + username + " fin");
```

flow condition : **\${ age < 18 }**

=====

pour :

```
params.put("pers", new Personne("toto",44));
```

scriptTask (javascript):

```
aCtx.doAction(execution,"debut " + pers.username + " fin");
```

flow condition : **\${ pers.age < 18 }**

scriptTask :

```
aCtx.doActionWithArgs(execution,"debut " + pers.username + " fin",new  
                        tp.test.activiti.Client(pers.username,pers.age));
```

ok en groovy (mais pas en javascript) pour la partie new tp.test.activiti.Client()

ok également en groovy pour un appel de méthode à nombre d'arguments variable

ex: public void doActionWithArgs(DelegateExecution execution,String actionName,Object ... objs)

exemple de condition:

```

<![CDATA[${order.price > 100 && order.price < 250}]]>
<![CDATA[${order.isStandardOrder()}]]>

```

NB : pour qu'un processus activiti ne se limite pas à des manipulations de données/variables on a

souvent besoin de :

1. configurer des composants "spring" accessibles pour déclencher des traitements .
2. préparer un paquet de classes utilitaires pour communiquer efficacement avec l'extérieur (envois de mail , appels de services , ...)

1.4. Déclenchement d'événements

Exemple de classe utilitaire pour déclencher des événements bpmn2/activiti :

```
public class ActivitiBpmEventManager {

    private RuntimeService activitiRuntimeService;

    public void setActivitiRuntimeService(RuntimeService activitiRuntimeService) {
        this.activitiRuntimeService = activitiRuntimeService;
    }

    public void signalEvent(String processName , String processInstanceId , String signalName ){
        try {
            Execution execution = activitiRuntimeService.createExecutionQuery()
                .processDefinitionKey(processName)
                .processInstanceId(processInstanceId)
                .signalEventSubscriptionName(signalName).singleResult();
            activitiRuntimeService.signalEventReceived(signalName,execution.getId());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void messageEvent(String processName , String processInstanceId , String eventName ,
        Map<String,Object> newProcessVariablesParams){
        try {
            Execution execution = activitiRuntimeService.createExecutionQuery()
                .processDefinitionKey(processName)
                .processInstanceId(processInstanceId)
                .messageEventSubscriptionName(eventName).singleResult();
            activitiRuntimeService.messageEventReceived(eventName,execution.getId(),
                newProcessVariablesParams);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
<definitions....>
<message id="reponseConges" name="reponseConges"></message>
<process id="myAsyncProcess" name="My async process" isExecutable="true">
    ...
<intermediateCatchEvent id="messageintermediatecatchevent1" name="MessageCatchEvent">
    <messageEventDefinition messageRef="reponseConges"></messageEventDefinition>
```

```

</intermediateCatchEvent>
....
</process>
</definition>

```

1.5. Délégation java (pour actions)

Exemple:

```

public class ActivitiBpmContextWithDataAccess {
...
public Object doAction(DelegateExecution execution,String actionName){
    Object result=null;
    logger.debug("activiti-do- actionName=" + actionName +
        " , instanceId : " + execution.getProcessInstanceId());
    // varXy = (ClasseXY) execution.getVariable("varXy");
    // execution.setVariable(varName, value);
    return result;
}
}

```

Cette classe de traitement peut être vue comme un composant spring de nom "*ctx*"

Et dans ce cas une Tâche de type java/javascript/groovy pourra simplement effectuer une délégation d'action java via une syntaxe du type

```
ctx.doAction(execution,"actionXY");
```

1.6. Principaux types de tâches/actions

ScriptTask	javascript (limité) ou groovy
UserTask	Tâche humaine interactive
...	

Exemples (pour la syntaxe dans le fichier .bpmn) :

```

<scriptTask id="scripttask6" name="delegate_increment_age" scriptFormat="javascript"
activiti:autoStoreVariables="true">
  <script>pCtx.doAction(execution,"increment_age");</script>
</scriptTask>

```


2. intégration de Activiti au sein de Spring

NB: le très gros intérêt de la configuration de activiti au sein de spring tient dans le fait que tout composant spring pourra être utilisé dans un processus activiti (via ServiceTask ou scriptTask) .

Autrement dit , un script (java, javascript , groovy, ...) du processus .bpmn2 pourra utiliser un composant Spring en tant qu'objet de traitement (ex : pour appeler un service web , pour envoyer un mail ou encore pour un message dans une file d'attente JMS) .

service-activiti-spring.xml (exemple) :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="activitiDataSource"
class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
    <property name="driverClass" value="org.h2.Driver" />
    <property name="url" value="jdbc:h2:mem:activiti;DB_CLOSE_DELAY=1000" />
    <property name="username" value="sa" />
    <property name="password" value="" />
  </bean>

  <bean id="activitiTransactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="activitiDataSource" />
  </bean>
  <!-- tout ce qui précède est un équivalent spring de activiti.cfg.xml -->

  <bean id="processEngineConfiguration"
class="org.activiti.spring.SpringProcessEngineConfiguration">
    <property name="dataSource" ref="activitiDataSource" />
    <property name="transactionManager" ref="activitiTransactionManager" />
    <property name="databaseSchemaUpdate" value="true" />
    <property name="jobExecutorActivate" value="false" />
  </bean>

  <bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">
    <property name="processEngineConfiguration" ref="processEngineConfiguration" />
  </bean>

  <bean id="repositoryService" factory-bean="processEngine"
    factory-method="getRepositoryService" />
  <bean id="runtimeService" factory-bean="processEngine"
    factory-method="getRuntimeService" />
  <bean id="taskService" factory-bean="processEngine" factory-method="getTaskService" />
  <bean id="formService" factory-bean="processEngine" factory-method="getFormService" />
  <bean id="historyService" factory-bean="processEngine" factory-method="getHistoryService" />
```

```

<bean id="managementService" factory-bean="processEngine"
      factory-method="getManagementService" />

<!-- autres beans accessibles depuis le processus (.bpmn): -->

<bean name="dynSoapClient" class="generic.ws.util.client.DynReflectSoapClient" />
<!-- utilisation: dynSoapClient.dynSoapCall(wsUrl, serviceInterfaceName,
                                           methodName, soapArgs); -->

<!-- et aussi
<bean id="javaJmsClient" class="generic.async.jms.util.GenericJavaJmsClient" >
  de jms-spring.xml -->
</beans>

```

Chargement d'un processus bpmn et démarrage via Spring :

```

public static void main(String[] args) {
    try{
        ClassPathXmlApplicationContext applicationContext = new
            ClassPathXmlApplicationContext("springContext.xml");
        RepositoryService repositoryService = (RepositoryService)
            applicationContext.getBean("repositoryService");
        RuntimeService activitiRuntimeService = (RuntimeService)
            applicationContext.getBean("runtimeService");
        String deploymentId = repositoryService
            .createDeployment()
            .addClasspathResource("test_activiti_with_props.bpmn")
            .deploy().getId();

        Map<String, Object> params = new HashMap<String, Object>();
        params.put("pers", new Personne("toto", 44L));
        ProcessInstance activitiProcessInstance =
            activitiRuntimeService.startProcessInstanceByKey("myProcess", params);
        // dès le démarrage --> initialisation dans eventuel ActivitiStartProcessListener
        // si défini dans .bpmn
        String processInstanceId=activitiProcessInstance.getId();
        System.out.println("started (with spring) new processInstanceId="+processInstanceId);

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Partie intéressante de la configuration maven pour spring et activiti :

pom.xml

```

...
<repositories>
  <repository>

```

```

    <id>Alfresco_Maven_Repository</id>
    <url>http://maven.alfresco.com/nexus/content/groups/public/</url>
  </repository>
</repositories>

<dependencies>

  <dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-engine</artifactId>
    <version>5.12</version>
  </dependency>

  <dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-spring</artifactId>
    <version>5.12</version>
  </dependency>

  <dependency>
    <groupId>org.codehaus.groovy</groupId>
    <artifactId>groovy-all</artifactId>
    <version>2.1.3</version>
  </dependency>

  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.3.170</version>
  </dependency>
  ....
</dependencies>
...

```

3. (interactive) UserTask – activiti

Une "UserTask" de activiti ressemble à une "HumanTask" de Jbpm/bpel et correspond à une tâche humaine (interactive et pas tout à fait immédiate).

Au sein d'Activiti , les "UserTask" gardent à peu près la même finalité que les "HumanTask" normalisées par Oasis / bpel for people mais se mettent en œuvre de façon simplifiée.

3.1. Sous processus logique d'une UserTask :

Une UserTask peut être associée à un groupe d'individus et dans ce cas :

- chaque membre du groupe aura l'occasion de "prendre en charge" cette tâche (claim: la réclamer) en choisissant celle ci parmi les tâches disponibles
- une fois réclamée , cette tâche sera affectée plus précisément à un certain utilisateur qui pourra alors :
 - * remplir un formulaire (avec des infos en lecture et des champs à renseigner)
 - * soumettre ce formulaire et ainsi compléter cette "UserTask"

Si dès le départ , une "UserTask" est affectée précisément à un individu , il n'y a pas besoin de réclamer la tâche.

3.2. Paramétrage d'une UserTask au sein du processus bpmn

```
<userTask id="usertask1" name="UserTaskSaisirAge" activiti:candidateUsers="kermit"
activiti:candidateGroups="management">
  <documentation>test userTask</documentation>
  <extensionElements>
    <activiti:formProperty id="age" name="age" type="long"
      expression="#{pers.age}"></activiti:formProperty>
    <activiti:formProperty id="username" name="username"
      expression="#{pers.username}" writable="false"></activiti:formProperty>
  </extensionElements>
</userTask>
```

La partie expression="#{objName.subPart}" permet de coder des associations avec certaines variables du processus bpmn et les noms/id seront exploités coté "formulaire de saisie" .

3.3. Contrôle via api "taskService" et "formService" d'activiti:

Activiti offre le moyen de personnaliser l'interface graphique du sous processus ci dessus.

On peut ainsi bâtir une IHM avec Struts , JSF ou GWT .

L'exemple de ManagedBean JSF suivant montre l'essentiel :

3.4. Exemple de classe Java associée à une UserTask

```
import org.activiti.engine.FormService;
import org.activiti.engine.RepositoryService;
import org.activiti.engine.TaskService;
import org.activiti.engine.form.FormProperty;
```

```

import org.activiti.engine.form.TaskFormData;
import org.activiti.engine.task.Task;

@Component
@Scope(value="session") //ici pour JSF
public class UserTaskMBean {

    @Autowired //@Inject
    private RepositoryService repositoryService;

    @Autowired //@Inject
    private CurrentUser currentUser;

    @Autowired //@Inject
    private FormService formService;

    private String taskIdToClaim;
    private String taskIdToComplete;

    private Task currentTask; //to perform/complete

    private List<FormProperty> formProperties;

    @Autowired //@Inject
    private TaskService taskService;

    private List<Task> availableTasks; //to claim

    private List<Task> assignedTasks; //to complete

    //liste des "UserTask" disponibles pour un groupe :
    public List<Task> getAvailableTasks() {
        availableTasks =
            taskService.createTaskQuery().taskCandidateGroup("management").list();
        return availableTasks; // "management" = exemple de group name
    }

    public String claimTask() {
        System.out.println("claimTask() with taskIdToClaim" + taskIdToClaim);
        taskService.claim(taskIdToClaim,currentUser.getUsername());
        return null;
    }

    //liste des "UserTask" affectées à un utilisateur précis :
    public List<Task> getAssignedTasks() {
        assignedTasks =
            taskService.createTaskQuery().taskAssignee(currentUser.getUsername()).list();
        return assignedTasks;
    }

    //préparation du formulaire pour la tâche choisie :
    public String performTask() {

```

```

String suite=null;
currentTask=taskService.createTaskQuery()
    .taskId(this.taskIdToComplete).singleResult(); //pour details;
System.out.println("performTask() with taskIdToComplete" + taskIdToComplete);
TaskFormData taskFormData = formService.getTaskFormData(taskIdToComplete);
formProperties = taskFormData.getFormProperties();
if(formProperties!=null && !formProperties.isEmpty())
{
    suite="performTask";//.xhtml //demander à afficher formulaire
    // later : formService.submitTaskFormData(taskId, properties);
}
else{
System.out.println("directly complete task with form properties");
taskService.complete(taskIdToComplete);
}
return suite;
}

```

//terminer la tâche en renvoyant les données saisies :

```

public String completeFormTask(){
String suite=null;
System.out.println("completeFormTask() with taskIdToComplete" + taskIdToComplete);
Map<String,String> idValuesPropertyMap = new HashMap<String,String>();
for(FormProperty prop:formProperties){
    //writable , required , type , name , id , ...
    System.out.println("formProperties[id="+prop.getId()+",value="
        + String.valueOf(prop.getValue())+"]");
    if(prop.isWritable()){
        idValuesPropertyMap.put(prop.getId(), String.valueOf(prop.getValue()));
    }
}
formService.submitTaskFormData(taskIdToComplete, idValuesPropertyMap);
//complete task with values
suite="userTask";//.xhtml //rediriger vers la liste des (autres) tâches
return suite;
}
//+get/set
}

```

userTask.xhtml (pour choisir une tâche parmi une liste)

```

...
<ui:define name="body">
    <h:form>
        <h5>assigned UserTask / to perfom/complete</h5>
        <h:dataTable border="2" var="assignedTask"
            value="#{userTaskMBean.assignedTasks}">
            <h:column> <f:facet name="header">Name</f:facet>
            <h:outputText value="#{assignedTask.name}" />
            </h:column>
            <h:column> <f:facet name="header">Description</f:facet>
            <h:outputText value="#{assignedTask.description}" />
            </h:column>
        </h:dataTable>
    </h:form>
</ui:define>

```

```

        <h:column>        <f:facet name="header">Action</f:facet>
            <b:h:commandButton action="#{userTaskMBean.performTask}"
                value="perform">
                <f:setPropertyActionListener value="#{assignedTask.id}"
                    target="#{userTaskMBean.taskIdToComplete}" />
            </h:commandButton >
        </h:column>
    </h:dataTable>

    <b><h5>available UserTask / to claim</h5>
    <h:dataTable border="2" var="availableTask" value="#{userTaskMBean.availableTasks}">
        <h:column>    <f:facet name="header">Name</f:facet>
            <h:outputText value="#{availableTask.name}" />
        </h:column>
        <h:column>    <f:facet name="header">Description</f:facet>
            <h:outputText value="#{availableTask.description}" />
        </h:column>
        <h:column>    <f:facet name="header">Action</f:facet>
            <b:h:commandButton action="#{userTaskMBean.claimTask}" value="claim">
                <f:setPropertyActionListener value="#{availableTask.id}"
                    target="#{userTaskMBean.taskIdToClaim}" />
            </h:commandButton >
        </h:column>
    </h:dataTable>
</h:form>
</ui:define>...

```

performTask.html (formulaire à remplir) :

```

...
<ui:define name="body">
    <h3><h:outputText value="#{userTaskMBean.currentTask.name}" /></h3>
    <hr/>
    <h:outputText value="#{userTaskMBean.currentTask.description}" />
    <hr/>
    <h:form>
    <h:messages/>
    <b><h:dataTable border="0" var="prop" value="#{userTaskMBean.formProperties}">
        <h:column>    <f:facet name="header">name</f:facet>
            <h:outputText value="#{prop.name}" />
        </h:column>
        <h:column>    <f:facet name="header">value</f:facet>
            <b><h:inputText value="#{prop.value}" rendered="#{prop.writable}"
required="#{prop.required}" />
            <h:outputText value="#{prop.value}" rendered="#{!prop.writable}" />
        </h:column>
    </h:dataTable>
    <b><h:commandButton action="#{userTaskMBean.completeFormTask}" value="complete">
        </h:commandButton >
    </h:form>
</ui:define>...

```

4. Framework "generic_bpm" de mycontrib.org

Rappel: pour qu'un processus activiti ne se limite pas à des manipulations de données/variables on a souvent besoin de :

1. configurer des composants "spring" accessibles pour déclencher des traitements .
2. préparer un paquet de classes utilitaires pour communiquer efficacement avec l'extérieur (envois de mail , appels de services , ...)

Didier Defrance (l'auteur de ce support de cours) a développé un mini-framework intitulé "**generic_bpm**" permettant de simplifier l'utilisation de "activiti5" et "jbpm5" .

Ce mini framework (pour l'instant en phase d'incubation) offre les fonctionnalités suivantes :

- apporter une couche d'abstraction pour masquer certaines différences entre "activiti" et "jbpm"
- simplifier le déclenchement d'événements au niveau d'un processus
- appeler simplement (depuis script) des traitements utilitaires (ex : envoi de mail , de message JMS) tout en ayant accès aux variables du processus
- appeler simplement des services WEB Soap
-

Ce code source de ce framework est accessible au bout de l'URL suivante (pour **git clone**) : [https://github.com/didier-mycontrib/utills\(.git\)](https://github.com/didier-mycontrib/utills(.git))

Une description détaillée du framework devrait courant 2014 est accessible sur le site www.mycontrib.org

XIII - jBPM et drools (moteur de règles)

1. Jbpm (présentation détaillée)

Jbpm (java or jboss Bpm) est une technologie "bpm" (*business process management*) inventée par Jboss pour **exécuter et contrôler des processus métiers modélisés en BPMN** .
C'est un ***moteur de workflow*** .



Avant toutes choses, il faut savoir qu'il y a d'énormes différences entre les versions de jBpm :

jbpm 3	Basé sur le propriétaire jPDL (jbpm Process Definition Language)
jbpm 4	Basé sur le propriétaire jPDL (jbpm Process Definition Language) avec ré-écriture complète api/framework .
jbpm 5	Basé sur BPMN 2 (standard OMG), n'utilise plus jpdL , intégré à drools , livré avec éditeur bpmn 2

L'ensemble de ce document est basé sur la version 5 de jBpm .

1.1. Installation (v5)

JBPM nécessite un assez gros paquet de librairies (.jar) :

jBPM Library

```

 jbpm-bam-5.2.0.Final.jar - C:\Prog\java\jbpm-installer\runtime
 jbpm-bpmn2-5.2.0.Final.jar - C:\Prog\java\jbpm-installer\runtime
 jbpm-flow-5.2.0.Final.jar - C:\Prog\java\jbpm-installer\runtime
 jbpm-flow-builder-5.2.0.Final.jar - C:\Prog\java\jbpm-installer\runtime
 jbpm-human-task-5.2.0.Final.jar - C:\Prog\java\jbpm-installer\runtime
 jbpm-persistence-jpa-5.2.0.Final.jar - C:\Prog\java\jbpm-installer\runtime
 jbpm-test-5.2.0.Final.jar - C:\Prog\java\jbpm-installer\runtime
 jbpm-workitems-5.2.0.Final.jar - C:\Prog\java\jbpm-installer\runtime
...
 drools-compiler-5.3.1.Final.jar - C:\Prog\java\jbpm-installer\runtime\lib
 drools-core-5.3.1.Final.jar - C:\Prog\java\jbpm-installer\runtime\lib
 drools-persistence-jpa-5.3.1.Final-tests.jar - C:\Prog\java\jbpm-installer\runtime\lib
 drools-persistence-jpa-5.3.1.Final.jar - C:\Prog\java\jbpm-installer\runtime\lib
...
 knowledge-api-5.3.1.Final.jar - C:\Prog\java\jbpm-installer\runtime\lib

```

Au secours !

Un plugin eclipse ou bien une configuration "maven" s'impose !

Les plugins eclipse "jbpm 5 eclipse plugin" et/ou "bpmn 2" , "...." sont accessibles au bout des URLs (des "update sites") suivantes :

<http://download.eclipse.org/graphiti/updates/0.8.1/> (dépendance du plugin bpmn2)

<http://codehoop.com/bpmn2> (quelques bugs pour l'instant)

Configuration maven pour jbpm5 :

pom.xml

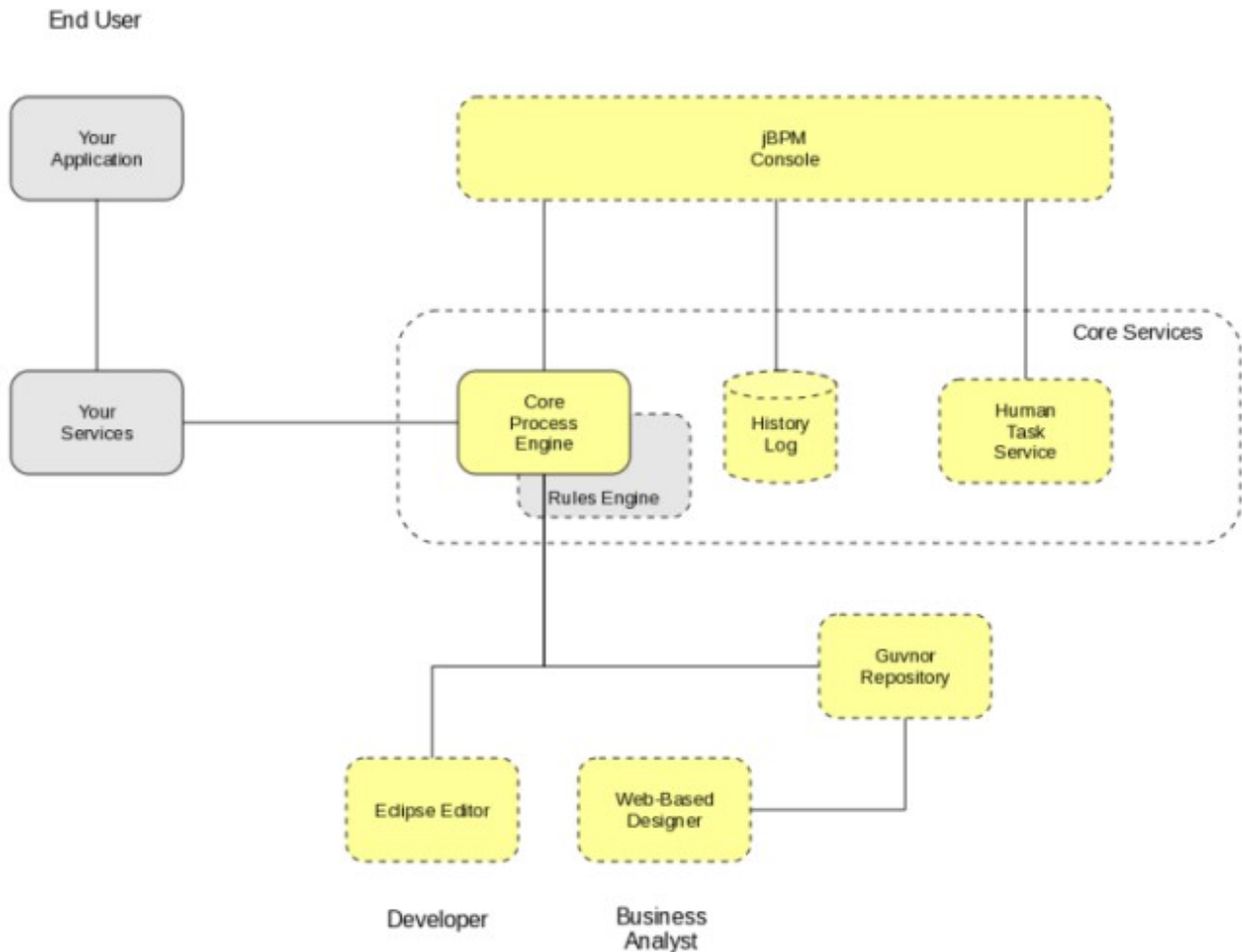
```
....
    <properties>
        <jbpm.version>5.2-Final</jbpm.version>  <!-- ou 5.1 , ... -->
    </properties>
....
<dependencies>
<dependency>
<groupId>org.jbpm</groupId>
<artifactId>jbpm-bpmn2</artifactId>
<version>${jbpm.version}</version>
</dependency>
<dependency>
<groupId>org.jbpm</groupId>
<artifactId>jbpm-human-task</artifactId>
<version>${jbpm.version}</version>
</dependency>
<dependency>
<groupId>org.jbpm</groupId>
<artifactId>jbpm-persistence-jpa</artifactId>
<version>${jbpm.version}</version>
</dependency>
<dependency>
<groupId>org.jbpm</groupId>
<artifactId>jbpm-bam</artifactId>
<version>${jbpm.version}</version>
</dependency>
...
</dependencies>
....
```

1.2. Editeur "jbpm process" ou bien "bpmn 2" ?

Fin 2011 , début 2012 l'éditeur "jbpm process" (sous forme de plugin eclipse) fonctionne bien mais ne prend pas en compte toutes les possibilités de BPMN2.

Un nouvel éditeur "bpmn2 editor" existe depuis jbpm 5.2 , il est à priori beaucoup plus complet mais il comporte aujourd'hui quelques bug (du côté .xsd , ...) - il faut attendre encore un peu .

1.3. Architecture (v5)



En gros :

- un analyste métier définit des "règles métiers" et des "processus métier" bpmn 2 depuis une console web
- un développeur java utilise eclipse pour retoucher si besoin un processus bpmn et l'entourer de code java (service , événements ,)
- l'utilisateur final utilise une IHM spécifique de l'application pour indirectement déclencher un service utilisant jbpm .
- un administrateur peut superviser le bon déroulement des processus via une console web

1.4. Utilisation depuis eclipse (v5)

Configuration préalable d'eclipse (3.6 ou 3.7) :

- installer le plugin jbpm

- 1) créer un nouveau projet de type "jbpm" (ou bien un projet java + ... ?)
- 2) créer un nouveau processus "bpmn" depuis l'éditeur intégré à eclipse
- 3) créer une classe java de test (avec un "main" ou avec Junit 4)

Le **code minimum** de cette classe de test doit être le suivant:

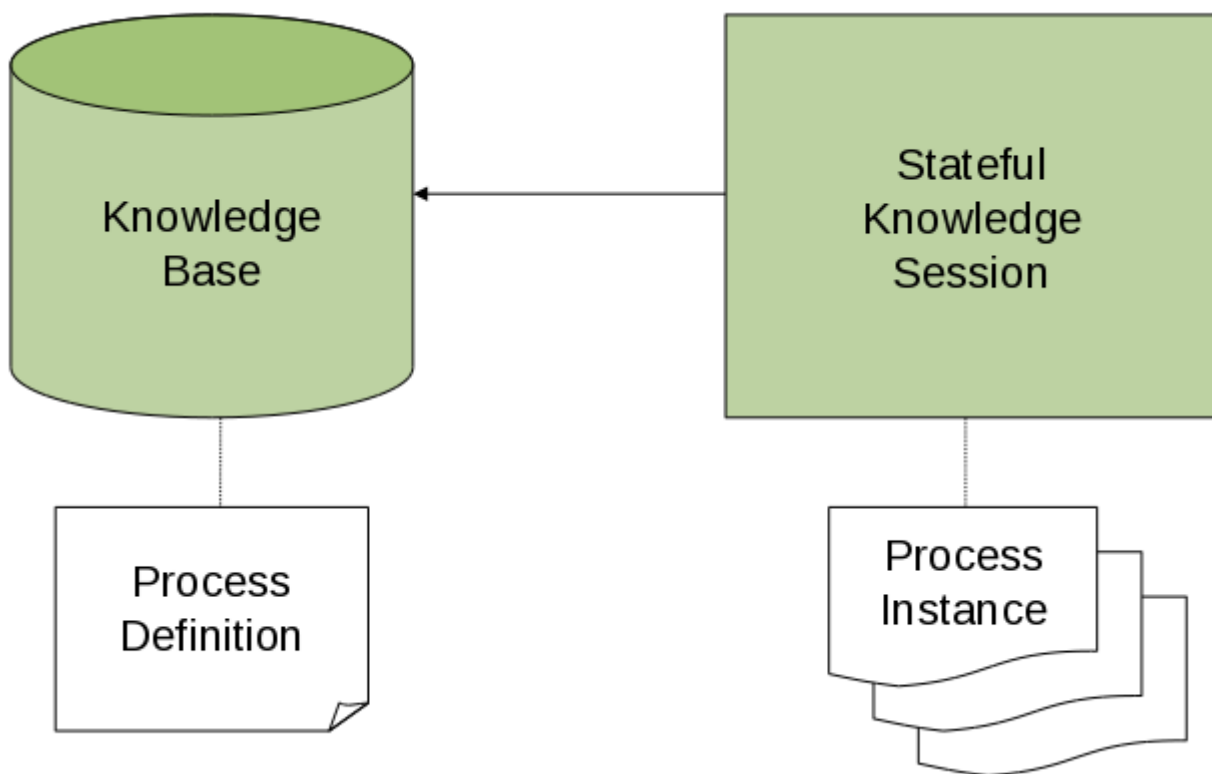
```
import org.drools.*;
...
```

```

public class ProcessMain {
public static final void main(String[] args) throws Exception {
    // load up the knowledge base
    KnowledgeBase kbase = readKnowledgeBase();
    StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();
    // start a new process instance
    ksession.startProcess("id_of_process");
}

private static KnowledgeBase readKnowledgeBase() throws Exception {
    KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
    kbuilder.add(ResourceFactory.newClassPathResource("myprocess.bpmn"),
        ResourceType.BPMN2);
    return kbuilder.newKnowledgeBase();
}
}

```



Passage de paramètres au processus à lancer :

Si le processus comporte des paramètres (variables), on peut préciser les valeurs initiales de celles-ci au lancement du processus :

```

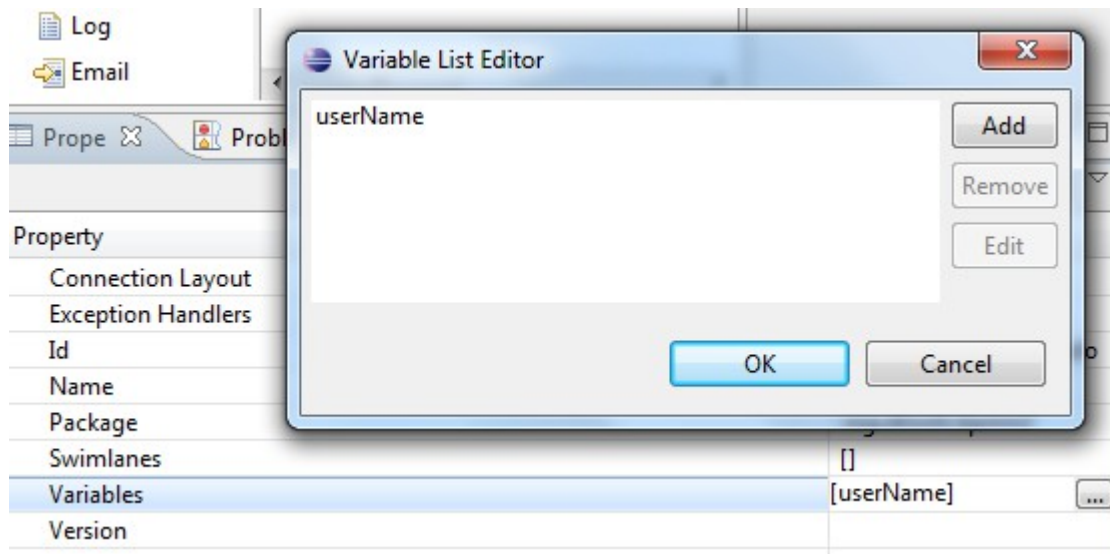
Map<String,Object> params = new HashMap<String,Object>();
params.put("userName","didier");
params.put("param2","valeur2");

ProcessInstance p = ksession.startProcess("id_of_process",params);

```

NB : au niveau du processus bpmn, le paramétrage des variables peut se faire via la fenêtre des

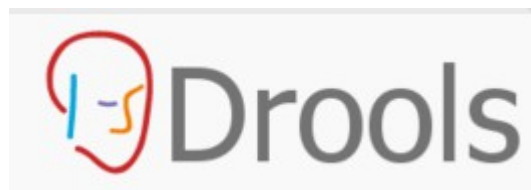
propriétés (en ayant préalablement sélectionner le fond du processus bpmn) :



et pour vérifier si la variable *userName* comporte la bonne valeur , on peut éventuellement paramétrer l'**action** d'une **tâche** via l'expression suivante : ***System.out.println(userName);***




2. Drools (présentation)

Drools est un système de gestion de règles métier utilisant un moteur d'inférence à chaînage avant (et éventuellement arrière) développé (*en open source*) par Jboss .



L'implémentation de "drools" est basé sur l'**algorithme de Rete**.

Composants de la version "community edition" (gratuite) de Jboss "drools 5" :

-  Drools Guvnor (Business Rules Manager)
-  Drools Expert (rule engine)
-  jBPM 5 (process/workflow)
-  Drools Fusion (event processing/temporal reasoning)
-  Drools Planner (automated planning)

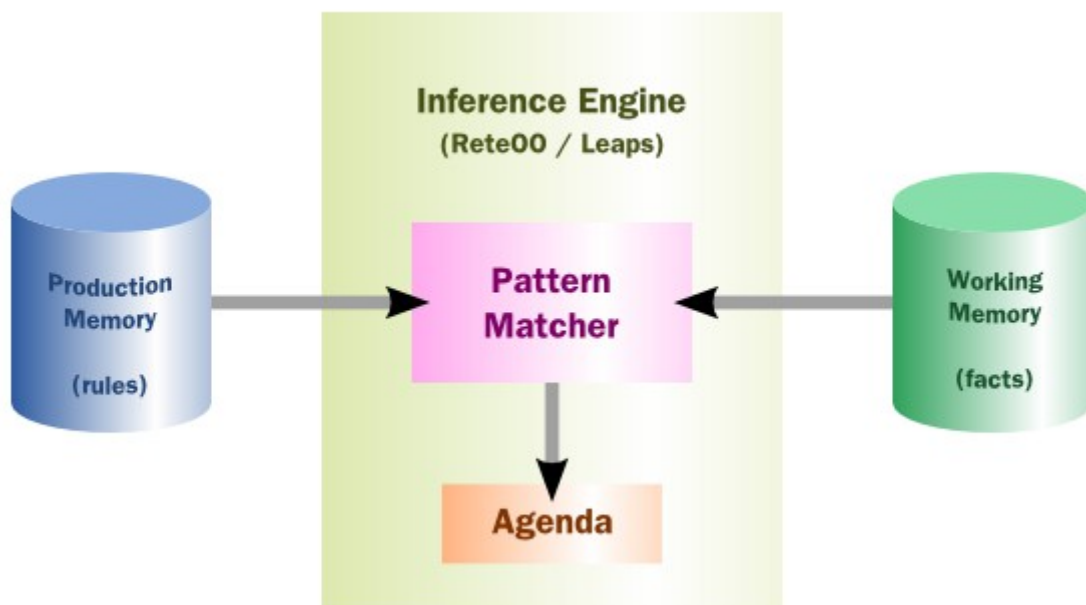
2.1. Principes de fonctionnement

Les **règles** (basée sur la logique du premier ordre) se présentent sous la forme

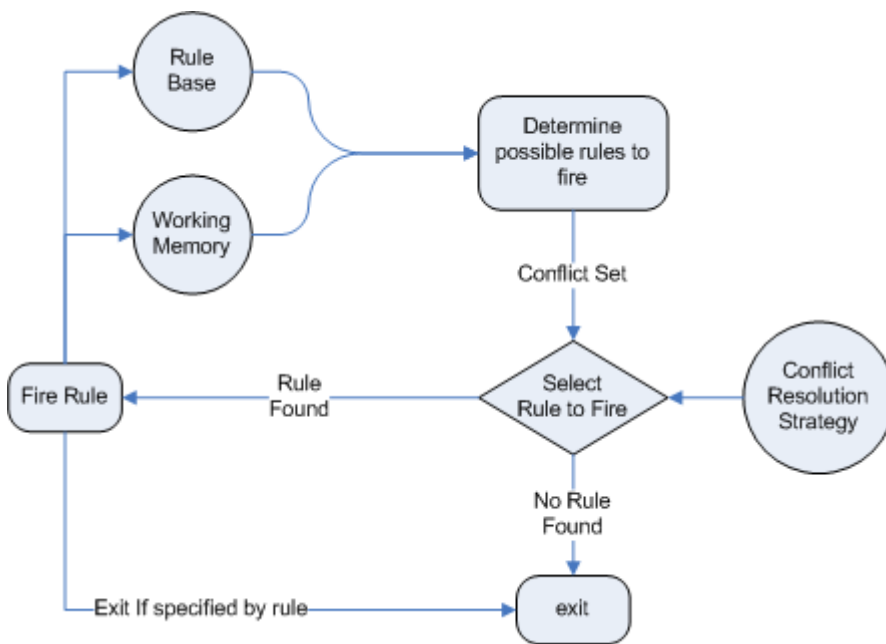
```
when
    <conditions>
then
    <actions>;
```

Elles sont **stockées dans une base de connaissances ("Knowledge base")** et sont utilisées par un système expert qui va essayer de les exploiter .

Une règle sera utilisée si l'expression de sa partie "conditions" peut être mise en correspondance avec un fait de l'application (valeur d'une instance java).



Fonctionnement du moteur de règles (en chaînage avant / déduction):



2.2. Installation

... selon version ...

2.3. lien entre expression des règles et classes java

```

public class Applicant {
    private String name;
    private int age;
    private boolean valid;
    // getter and setter methods here
}
  
```

<-->

```

package com.company.license

rule "Is of valid age"
when
    $a : Applicant( age < 18 )
then
    $a.setValid( false );
end
  
```

(dans fichier "licenseApplication.drl")

2.4. Utilisation basique de drools 5

Initialisation:

```

KnowledgeBuilder kbuilder= KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add ( ResourceFactory.newClassPathResource( "licenseApplication.drl",getClass()),
    ResourceType.DRL);
if(kbuilder.hasErrors()){
    System.err.println(kbuilder.getErrors().toString());
}
KnowledgeBase kbase=KnowledgeBaseFactory.newKnowledgeBase();
  
```

```
kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
```

Utilisation du moteur de règles:

```
StatelessKnowledgeSession ksession = kbase.newStatelessKnowledgeSession();
```

```
Applicant applicant=new Applicant("Mr John Smith",16);  
assertTrue(applicant.isValid());
```

```
ksession.execute(applicant); //l'exécution de la règle doit normalement basculer valid à false  
assertFalse(applicant.isValid());
```

NB: on peut déclencher le moteur de règles sur une collection (itérable) d'objets à traiter :

```
...  
List<Object> liste = new ArrayList<Object>();  
liste.add(o1);  
liste.add(o2);  
ksession.execute(liste);  
...
```


3. Événements et tâches de jbpms 5

3.1. Différents types d'événements et de tâches.

Principaux types d'événements :

TimerEvent	Déclenchement automatique après un certain délai (période de temps écoulée) , peut quelquefois être à déclenchement multiple/périodique.
SignalEvent	événement entrant déclenché par code java ou événement sortant

Principaux types de tâches :

ScriptTask	tâche bpmn associée à un script (java/mvel) qui sera interprété . (ex : <code>System.out.println("Hello " + userNameVariable)</code>) ;
ServiceTask	tâche abstraite bpmn associée à un du code java non directement interprété par le processus mais géré par l'environnement java hôte . Cette unité de traitement (ou service java) déclenchée depuis le processus bpmn peut communiquer des paramètres d'entrée et de retour (service result) .
UserTask	tâche faisant intervenir une interaction avec un utilisateur (humain) .
(business) RuleTask	tâche associée à l'évaluation d'un groupe de règles (drools)
(resusable) SubProcess	exécuter un sous processus complet identifié par un id .
Embedded Sub-process	Sous processus imbriqué (sous nœuds dans le nœud) .
Multi-instance sub-process	Idem "embedded sub-process" mais exécuté plusieurs fois (selon le nombre d'éléments d'une collection)

NB: Une *"mvel" expression* est de type `#{object.subobject.property}`

NB2 (global variable) :

- Est appelée "variable globale" , une variable de niveau "ksession" (qui pourrait être partagée par plusieurs "process instance").
- une variable globale peut être évaluée par `ksession.setGlobal(name, value)`;

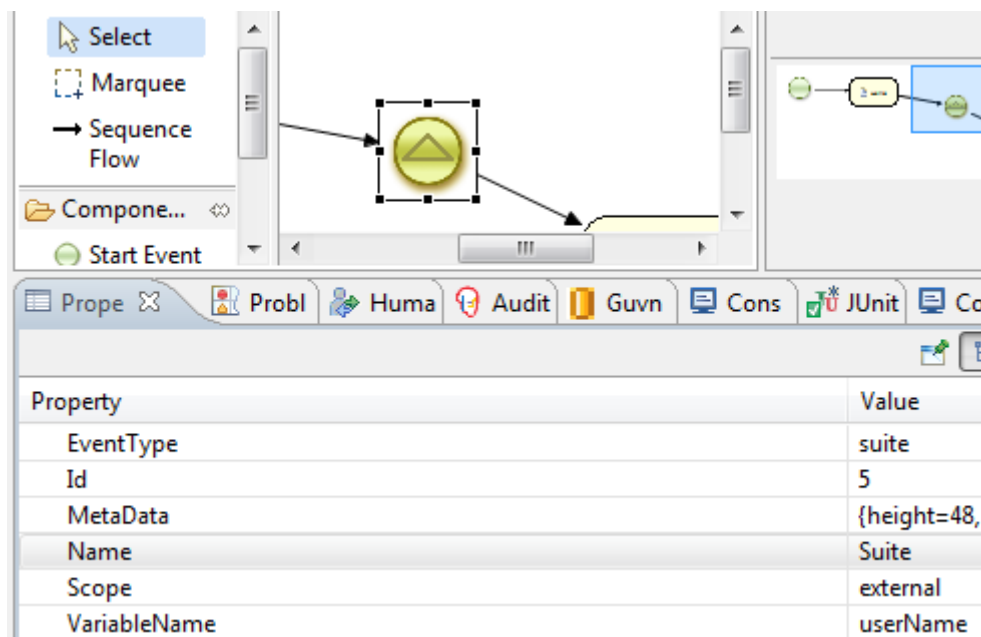
3.2. TimerEvent

Paramétrage du "Timer **Delay**" : 2s (ou 2000ms) ou 1s999ms ou ...

format général : `[#d] [#h] [#m] [#s] [#ms]`

3.3. Activation/Signalisation (en java) d'un événement

Soit "*suite*" un événement intermédiaire de type "*signal event*" du processus bpmn paramétré de la façon suivante :



NB : id (unique) , name (s'affiche quelquefois dans le diagramme)

eventType : type fonctionnel (expression libre) de l'événement qui servira à faire le lien avec le code java de déclenchement.

variableName (facultatif) : nom d'une éventuelle variable qui pourra être évaluée lors du déclenchement de l'événement.

Code java de déclenchement :

```
processInstance.signalEvent("eventType", dataObject);
ou bien
ksession.signalEvent("eventType", dataObject, processInstance);
```

où *dataObject* est une valeur (éventuellement nulle) qui sera affectée dans une éventuelle variable associée à l'événement dans le processus bpmn.

Exemples :

```
p.signalEvent("suite", null);
p.signalEvent("suite", "Didier");
```

3.4. Script Task

Action = toute série d'instructions java valides

Exemples d'expressions de l'action :

```
System.out.println("Hello " + userNameVariable );
```

```
x=5 ;
```

```
System.out.println(x) ;
```

si **p** est une variable déclarée dans le processus de type *data.Personne*

alors on peut écrire *p.getAge()*; ou bien *p.setAge(n)*; dans l'action d'un "scriptTask" .

NB1 : une affectation de variable effectuée au sein d'un premier script (via x=5) n'est pas vue depuis un autre script (d'un autre ScriptTask) .

Pour qu'une affectation de variable soit vue des autres scripts il faut utiliser l'expression suivante :

kcontext.setVariable(variableName, value);

exemple : *kcontext.setVariable("x",5);*

NB2 : pour modifier (depuis un script) une variable globale (de niveau ksession) , il faut utiliser l'expression suivante :

```
kcontext.getKnowledgeRuntime().setGlobal(variableName, value) ;
```

3.5. ServiceTask

De nouvelles tâches spécifiques au domaine de l'application se configurent via :

- une configuration déclarative xml (nom , paramètres , resultat)
- une classe java devant implémenter l'interface "WorkItemHandler" et à enregistrer

La déclaration xml des "service task" s'effectue dans un fichier du type suivant :

META-INF/MyWorkDefinitions.conf

```
import org.drools.process.core.datatype.impl.type.StringDataType;
[
  [
    "name" : "Notification",
    "parameters" : [
      "Message" : new StringDataType(),
      "From" : new StringDataType(),
      "To" : new StringDataType(),
      "Priority" : new StringDataType(),
    ],
    "displayName" : "Notification",
    "icon" : "icons/notification.gif",
  ],
]
```

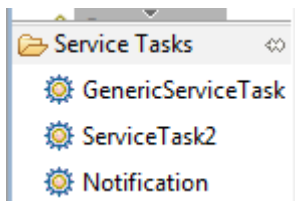
```
[
  "name" : "ServiceTask2",
  "parameters" : [
    "p1" : new StringDataType(),
  ],
  "displayName" : "ServiceTask2",
]
]
```

Ce fichier (de nom quelconque) doit être renseigné dans le fichier de nom imposé :

META-INF/drools-rulebase.conf

drools.workDefinitions = MyWorkDefinitions.conf [MyWorkDefinitions2.conf]

Suite à un "refresh" du projet (dans eclipse), cette configuration sera analysée et la palette de l'éditeur jbpm contiendra les "service task" spécifiques :



La classe java qui implémente un "serviceTask" spécifique doit être de ce type :

```
package com.sample;

import org.drools.runtime.process.WorkItem;
import org.drools.runtime.process.WorkItemHandler;
import org.drools.runtime.process.WorkItemManager;

public class MyNotificationWorkItemHandler implements WorkItemHandler {

    @Override
    public void abortWorkItem(WorkItem workItem, WorkItemManager manager) {
        // Do nothing, notifications cannot be aborted
    }

    @Override
    public void executeWorkItem(WorkItem workItem, WorkItemManager manager) {
        // extract parameters
        String from = (String) workItem.getParameter("From");
        String to = (String) workItem.getParameter("To");
        String message = (String) workItem.getParameter("Message");
        String priority = (String) workItem.getParameter("Priority");

        // do work item :
        System.out.println("message:" + message);

        // notify manager that work item has been completed
        manager.completeWorkItem(workItem.getId(), null);
    }
}
```

```
}  
}
```

Ce "gestionnaire de serviceTask" doit être enregistré avant le démarrage du processus :

```
ksession.getWorkItemManager().registerWorkItemHandler("ServiceTaskName",  
                                                    new SpecificServiceTaskWorkItemHandler());
```

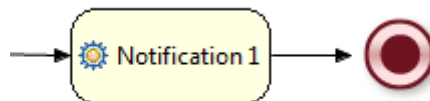
exemple :

```
ksession.getWorkItemManager().registerWorkItemHandler(  
    "Notification", new MyNotificationWorkItemHandler());
```

NB : le nom du (type de) "ServiceTask" doit être ici le nom déclaré dans le fichier *MyWorkDefinitions.conf* et qui se retrouve *préfixé par tns:taskName* dans le fichier *.bpmn* .

```
<task id="_9" name="Notification 1" tns:taskName="Notification" >  
...</task>
```

Autrement dit, le nom visible dans le diagramme (name=Notification 1") peut être ajusté.

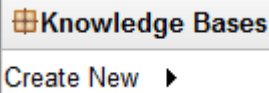


4. Bpmn2 webDesigner (of drools)

La console web de "drools" comporte un éditeur BPMN2 très complet .

4.1. Sélection ou création d'un processus bpmn

- 1) lancer la console web de drools via l'URL suivante : <http://localhost:8080/drools-guvnor>
- 2) dans l'arborescence se placer dans "*Knowledge Bases*" puis sélectionner un processus existant ou bien en créer un nouveau .



(*Create New / new BPMN2 Process*).

4.2. Edition d'un processus bpmn 2 via le "web designer"

...

NB: ne pas oublier de sauvegarder les modifications effectuées via le menu "*File/Save changes*" !

ANNEXES

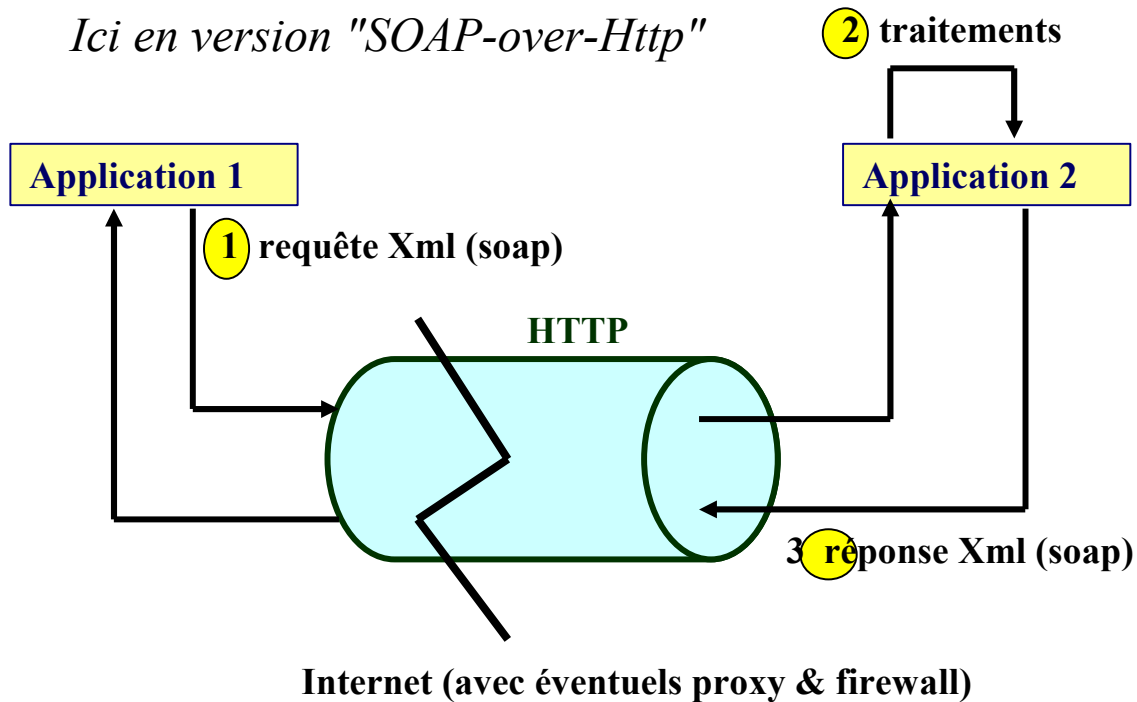
XIV - Annexe – SOAP et WSDL

1. Protocole SOAP

1.1. SOAP on HTTP en mode synchrone (RPC)

SOAP (*Simple Object Access Protocol*)

Ici en version "SOAP-over-Http"



1.2. SOAP on JMS (ou SMTP) en mode asynchrone

L'enveloppe SOAP véhiculée est la même qu'avec SOAP over HTTP.
Seul le protocole de transfert est variable ==> SMTP ,

Un mode asynchrone avec de véritable "files d'attentes" ou ("mail box") peut quelquefois est très pratique.

2. Éléments du protocole SOAP

Bien que le sigle **S.O.A.P.** sous entende principalement l'accès distant à un objet de façon à y invoquer des fonctions, le protocole **SOAP** est avant tout basé sur le concept d'**envoi de messages** (message de requête, message de réponse).

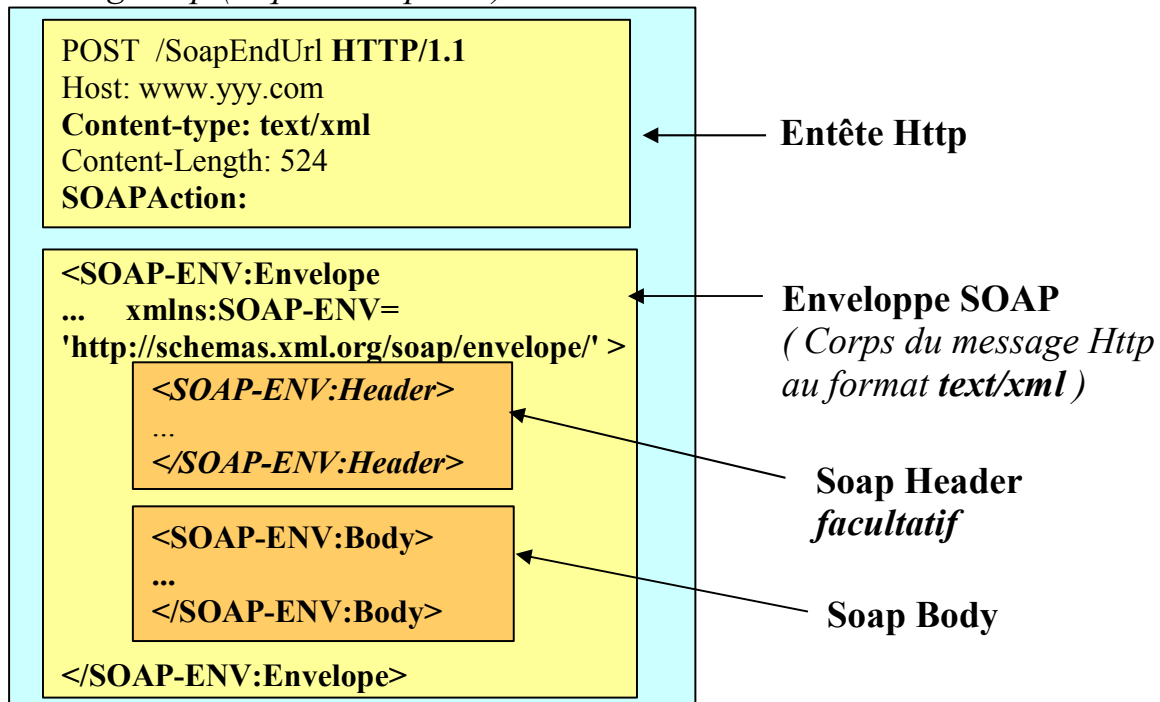
Ces messages peuvent tout aussi bien comporter des appels de fonctions (avec des paramètres valués) que comporter des messages textuels (éventuellement balisés en xml) et eux-même éventuellement accompagnés de pièces jointes.

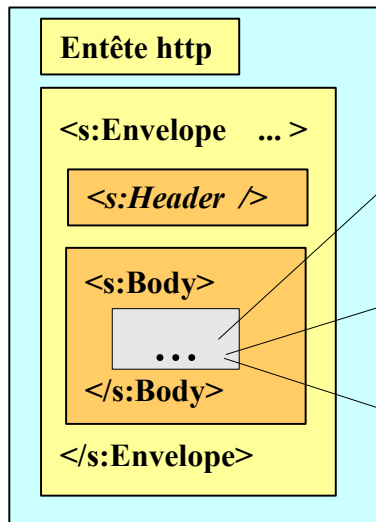
Le **protocole SOAP** est constitué de **3 grandes parties (aspects complémentaires)**:

- L' **enveloppe SOAP** précisant le **contenu** du message avec ses parties facultative (header) et obligatoire (body) .
- L' **ENCodage SOAP** (sérialisation et dé-sérialisation des types de données)
NB: l'encodage SOAP normalisé au sein de la version 1.1 et correspondant au mode "rpc/encoded" est petit à petit supplanté par l'encodage "literal" .
- La représentation des appels de fonctions (**SOAP-RPC**) et des réponses (ou erreurs).

Enveloppe SOAP véhiculée via HTTP

message http (requête / réponse)



Contenu du corps de l'enveloppe "SOAP"*message http**(avec contenu SOAP)*requête

```
<methodeXy>
  <p1>val1</p1>
  <p2>val2</p2>
</methodeXy>
```

réponse

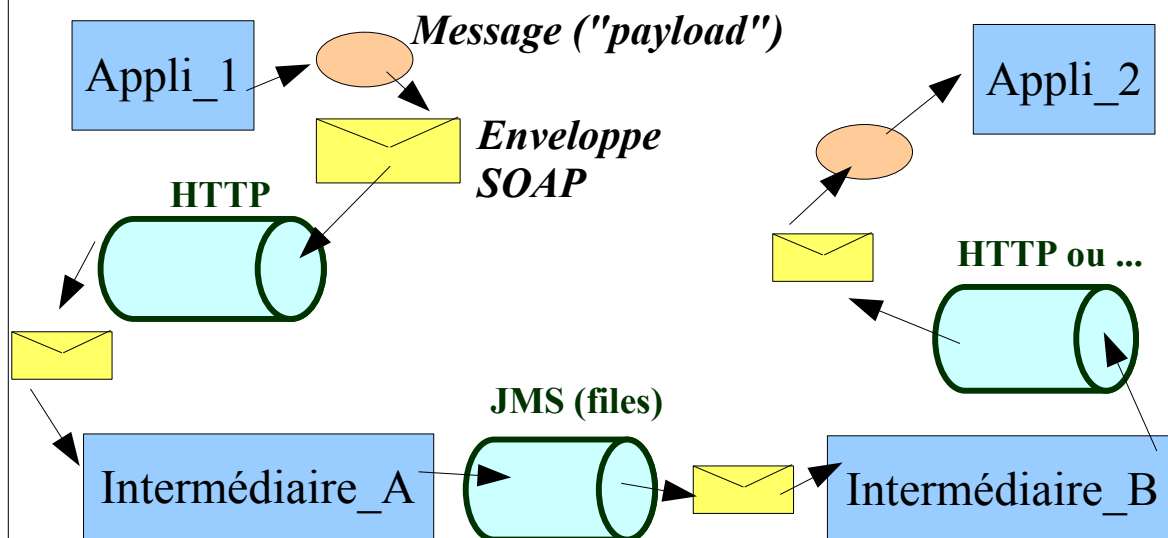
```
<methodeXyResponse>
  <return>val_resultat</return>
</methodeXyResponse>
```

Ou bien

Ou bien

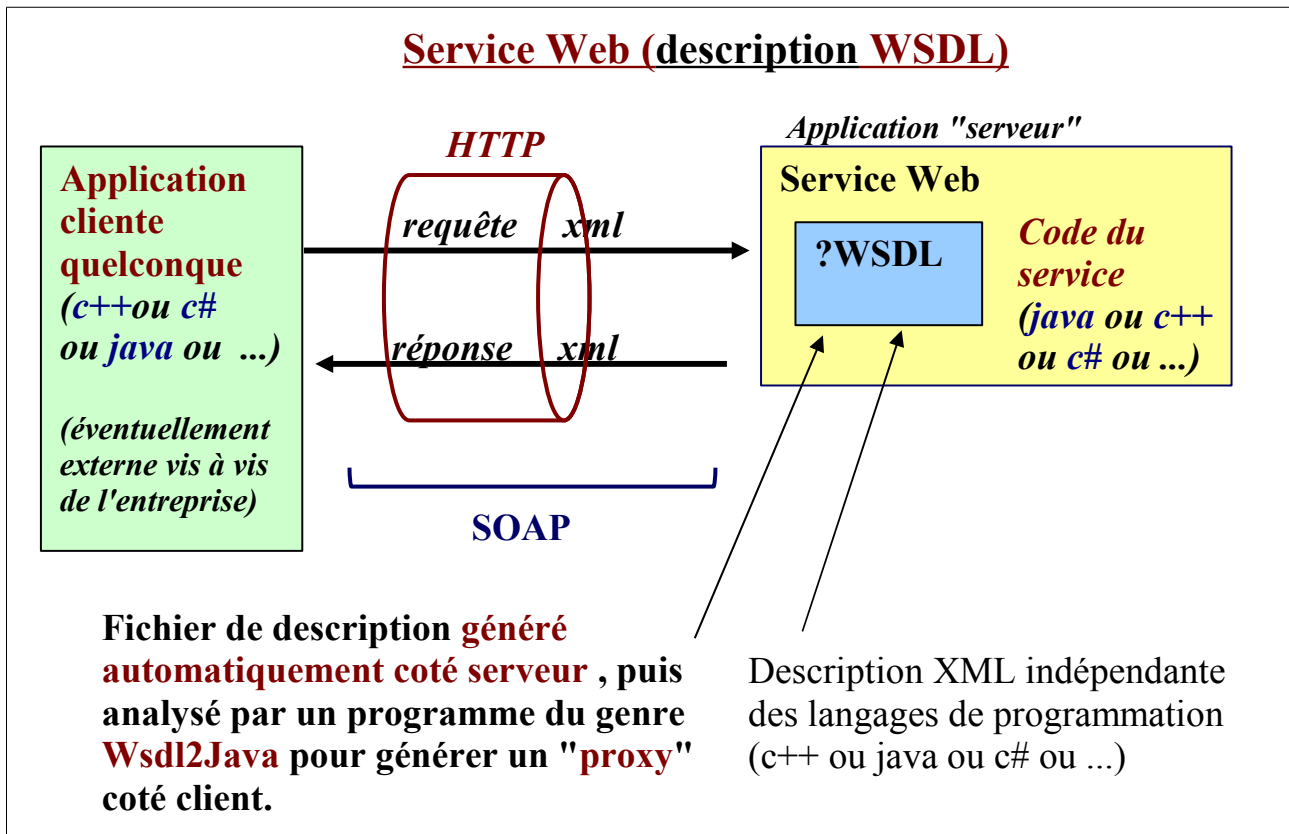
Exception (fault)

```
<s:fault>
  <faultcode>soap:Server</faultcode>
  <faultstring>msg_error</faultstring>
</s:fault>
```

SOAP: "Payload", enveloppe et transportsAnalogies: Lettre , enveloppe , acheminement (voiture + train + avion + ...)

Charge utile , conteneur , transport (train + bateau + camion + ...)

3. WSDL (Web Service Description Language)



WSDL est un langage standard basé sur Xml qui a pour objectif de **décrire précisément les prototypes des fonctions activables au niveau d'un certain service web.**

Code coté serveur du *service web* en C++ (gSoap)

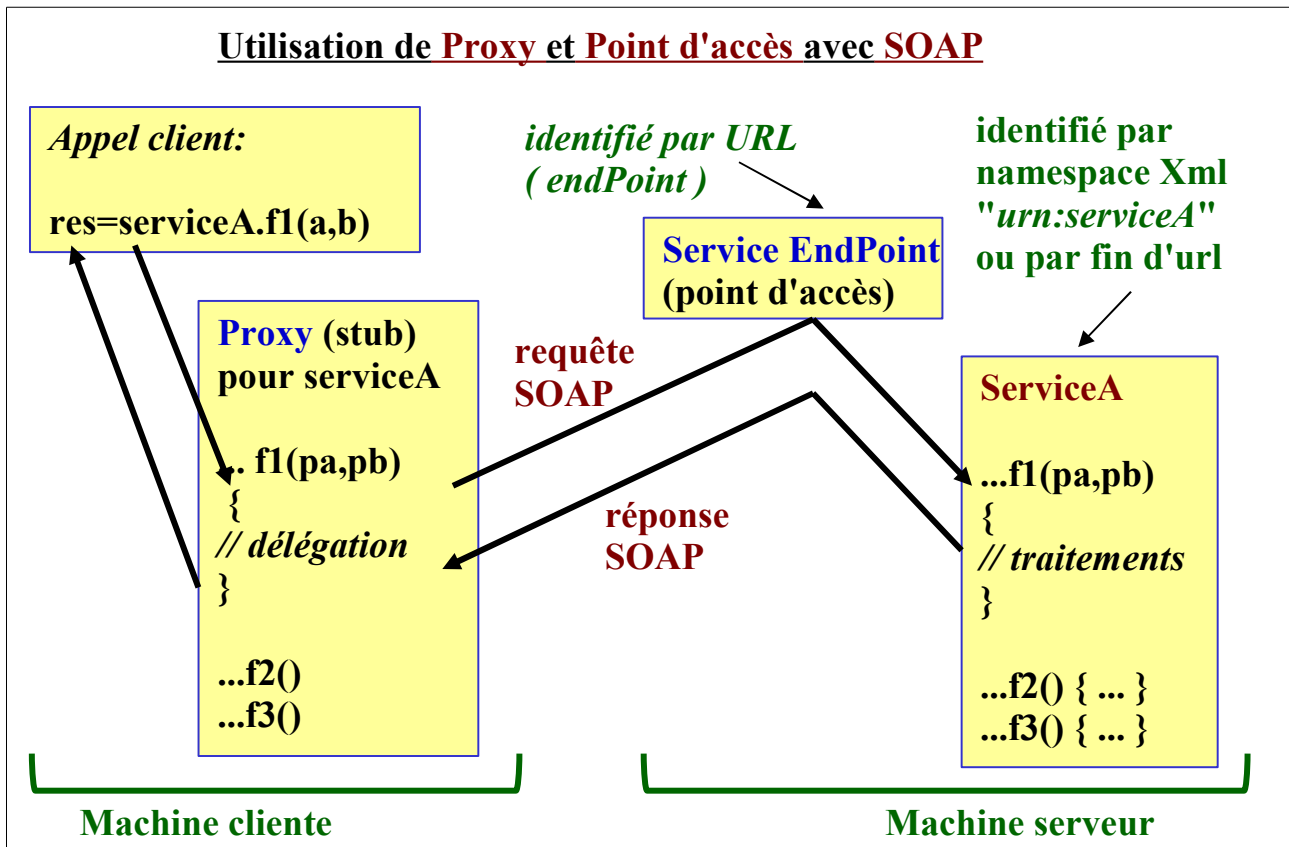
- génération du fichier WSDL via soapcpp2

WSDL

- génération via **WSDL2Java** (ou **wsimport**)
d'un *proxy* pour client **Java**

Quelques définitions (WSDL):

message	paramètre(s) en entrée ou bien en sortie d'une opération
portType	Interface (ensemble d'opérations abstraites)
operation	couple (message d'entrée, message de retour) = méthode
binding	associer un portType à un protocole (SOAP, COBBA, DCOM) et à un manespace (un même URI qualifiant généralement tout un service web)
service	service Web (avec url) comportant un ensemble de port(s) = endpoint(s)



4. "Basic profile" pour services "web"

De façon à garantir une bonne interopérabilité entre différentes implémentations des services "Web" (java, c++, .net , ...) qui risqueraient de diverger sur certains détails, l'organisme suivant est censé publier quelques règles assez strictes:

<http://www.ws-i.org/> (**Web Services – Interoperability Organization**)

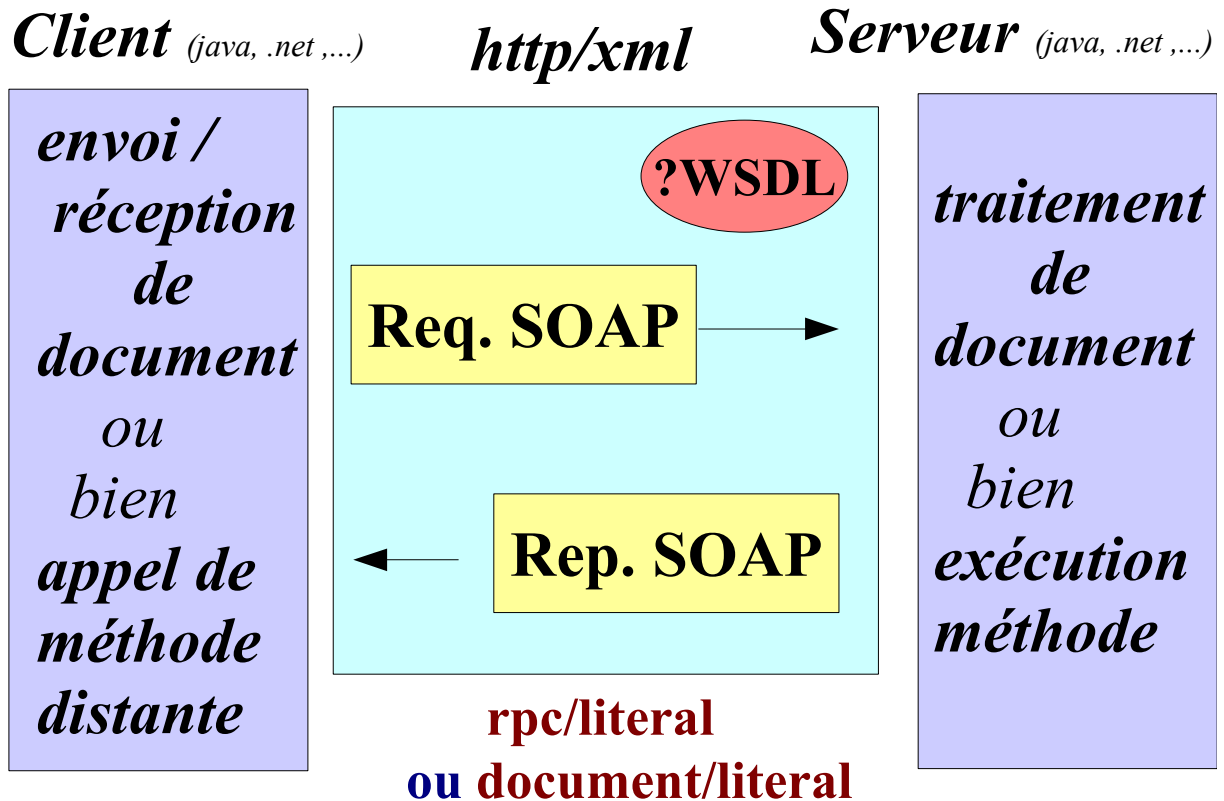
==> basic profile (v1 , v1.1 , ...) .
confirmations des choix de SOAP et précisions sur quelques points ambigus .

NB: Ceci est essentiellement utile pour IBM, SUN, Microsoft , ... et les autres éditeurs qui doivent mettre en place des outils et des API permettant de gérer simplement SOAP.

Le programmeur de services Web n'a généralement pas à se pré-occuper de ces détails car l'encodage XML est indirectement effectué par des API de hauts niveaux .

NB: Lors du choix d'une technologie liée aux services WEB, la compatibilité WS-I est un critère fondamental .

5. "Style/use" SOAP (rpc/literal , document/literal)

**Remarque importante:**

Lors du paramétrage d'un service WEB, il faut généralement choisir une combinaison "style/use" qui peut officiellement prendre l'une des valeurs suivantes:

- *rpc/encoded*
- *rpc/literal*
- *document/encoded*
- *document/literal*

En pratique "*document/encoded*" n'est **jamais utilisé**.

Bien que le style "document" soit plutôt approprié aux échanges de documents et que le style "rpc" soit plutôt approprié aux appels de méthodes à distance, les 2 styles SOAP "rpc" et "document" doivent normalement être replacés dans leurs contextes exacts:

- Les styles SOAP ("rpc" ou "document") n'ont (en toute rigueur) qu'un impact sur la partie xml (messages SOAP et description WSDL).
- Rien n'empêche une application d'utiliser le style "document" pour appeler une méthode à distance.

5.1. "rpc/encoded" et "rpc/literal"

Détails :

Partie d'une description WSDL généré en mode **"rpc/encoded"** ou **"rpc/literal"** :

```
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>

<binding .../>
```

==> pas de bloc de type "ComplexType" , ...

Partie (un peu simplifiée) d'une requête SOAP en mode **"rpc/encoded"** :

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <x xsi:type="xsd:int">5</x>
      <y xsi:type="xsd:float">5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

Partie (un peu simplifiée) d'une requête SOAP en mode **"rpc/literal"** :

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
      <y>5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

5.2. "document/literal" et variantes ("wrapped" , "bared" , ...)

Détails:

Partie (un peu simplifiée) d'une requête SOAP en mode **"document/literal"** :

```
<soap:envelope>
  <soap:body>
    <myDoc ou myMethod>
      <x>5</x>
      <y>5.0</y>
    </myDoc ou myMethod>
  </soap:body>
</soap:envelope>
```

(==> pas de différence notable coté "message SOAP" avec le mode "rpc/literal").

La principale nouveauté du mode "**document/literal**" tient dans le fichier **WSDL** décrivant le service WEB :

```
<types>
  <schema>
    <element name="myDoc ou myMethod">
      <complexType>
        <sequence>
          <element name="x" type="xsd:int"/>
          <element name="y" type="xsd:float"/>
        </sequence>
      </complexType>
    </element>
    <element name="myMethodResponse">
      <complexType/>
    </element>
  </schema>
</types>
<message name="myMethodRequest">
  <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
  <part name="parameters" element="myMethodResponse"/>
</message>

<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>

<binding .../>
```

Le mode "*document/literal*" constitue le meilleur choix dans la majorité des cas.
Ce mode est pris en charge par les technologies récentes (coté ".net" et coté "java") .

Variantes "wrapped" et "bared":

A partir d'une même expression "SOAP/WSDL" lié au mode "document/literal" , on peut envisager 2 variantes au niveau du code (java ou ...) dénommées "**wrapped**" et "**bared**":

- une variante où la méthode distante comporte (et enveloppe) tout un tas de paramètres unitaires (x,y , ...)
- une variante où la méthode distante comporte un seul gros objet de type "message_Requete" comportant lui même les champs élémentaires "x" , "y" , ...

Quelques URL pour approfondir le sujet:

<http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/>

<http://www.awprofessional.com/articles/article.asp?p=169106&seqNum=5&rl=1>

<http://java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns/index.html>

5.3. Versions et évolutions de SOAP/WSDL

Différentes versions (et évolution)

... , **SOAP 1.1** (xmlns="<http://schemas.xml.org/soap/envelope/>")
SOAP 1.2 (xmlns="<http://www.w3.org/2003/05/soap-envelope>")

Attention: SOAP 1.2 n'est pas pris en charge par toutes les technologies "WebService" et la version 1.1 suffit en général.

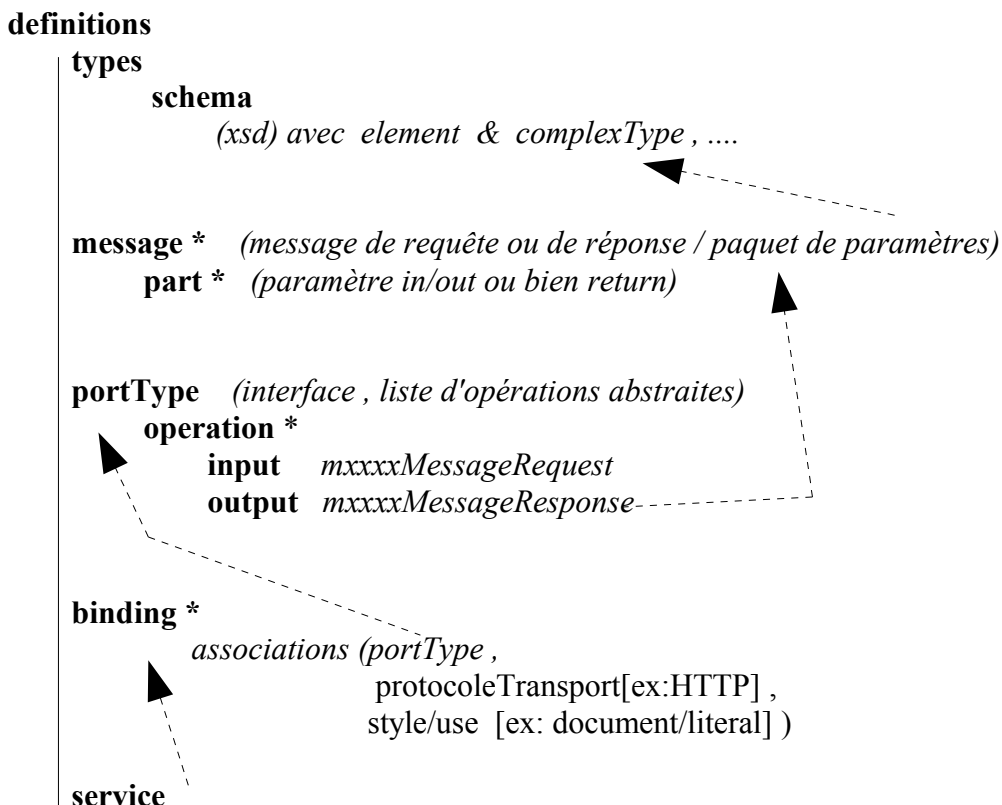
... , **WSDL 1.1** (xmlns="<http://schemas.xmlsoap.org/wsdl/>")
WSDL 2 (xmlns="<http://www.w3.org/ns/wsdl>")

Attention: WSDL 2 n'est pas pris en charge par toutes les technologies "WebService" et la version 1.1 suffit en général. Cependant, certaines extensions WSDL2 (MEP) sont utilisées par certains produits (ex: ESB)

Pour SOAP, **style/use** = *rpc/encoded* (complexe et "has been")
 puis *rpc/literal* (plus performant, plus simple)
 puis *document/literal* (avec schéma "xsd" pour valider)

6. Détails sur WSDL

6.1. Structure générale d'un fichier WSDL




```
port * (point d'accès)
  <address location="url_du_service" />
```

NB: Le contenu de la partie <xsd:schema> (sous la branche "wsdl:types") peut éventuellement correspondre à un sous fichier annexe "*xxx.xsd*" relié au fichier "*xxx.wsdl*" via un "*xsd:import*".

6.2. Exemple de fichier WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://calcul"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://calcul"
xmlns:intf="http://calcul" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48 PDT)-->

  <wsdl:types>
    <schema elementFormDefault="qualified" targetNamespace="http://calcul"
xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="addition">
        <complexType>
          <sequence>
            <element name="a" type="xsd:int"/>
            <element name="b" type="xsd:int"/>
          </sequence>
        </complexType>
      </element>
      <element name="additionResponse">
        <complexType>
          <sequence>
            <element name="additionReturn" type="xsd:int"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </wsdl:types>

  <wsdl:message name="additionResponse">
    <wsdl:part element="impl:additionResponse" name="parameters"/>
  </wsdl:message>

  <wsdl:message name="additionRequest">
    <wsdl:part element="impl:addition" name="parameters"/>
  </wsdl:message>

  <wsdl:portType name="Calculator">
    <wsdl:operation name="addition">
      <wsdl:input message="impl:additionRequest" name="additionRequest"/>
      <wsdl:output message="impl:additionResponse" name="additionResponse"/>
    </wsdl:operation>
  </wsdl:portType>
```

```

<wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">
  <wsdlsoap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="addition">
    <wsdlsoap:operation soapAction=""/>

    <wsdl:input name="additionRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>

    <wsdl:output name="additionResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="CalculatorService">
  <wsdl:port binding="impl:CalculatorSoapBinding" name="Calculator">
    <wsdlsoap:address
      location="http://localhost:8080/axis2-webapp/services/Calculator"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

6.3. WSDL abstraits et concrets

Structure **WSDL** (*abstrait ou concret*)

WSDL abstrait (*interface*)

```

<definitions ...>
  <types>
    <schema>
      <complexType .../> ...
      <element .../> ...
    </schema>
  </types>
  <message>...</message>
  <message>...</message>
  <portType>
    <operation ...>
      <input .../> <output .../>
    </operation>
    <operation>...</operation>
  </portType>
</definitions>

```

WSDL concret/complet (*impl*)

```

<definitions>
  <types>
    <schema ... />
  </types>
  <message>...</message> ...
  <portType>
    <operation .../> ...
  </portType>
  <binding>
    ... HTTP/SOAP style="document"
    .... use="literal" ....
  </binding>
  <service>
    <port><address location="...url..." />
  </port> </service>
</definitions>

```

Partie abstraite

Structure et sémantique WSDL (partie abstraite)

- Un *même "WSDL abstrait"* (idéalement issu d'une norme) pourra être physiquement implémenté par *plusieurs services distincts* (avec des *points d'accès (et URL) différents*).
- Un *portType* représente (en XML) l'*interface fonctionnelle* qui sera accessible depuis un futur "port" (*alias "endPoint"*)
- Chaque opération d'un "portType" est associée à des *messages* (structures des *requêtes[input]* et *réponses[output]*).
- En mode "*document/literal*", ces *messages* sont définis comme des *références* vers des "*element*"s (*xml/xsd*) qui ont la structure XML précise définie dans des "*complexType*".

Structure et sémantique WSDL (partie concrète)

- Un fichier "*WSDL concret*" comporte toute la partie abstraite précédente plus tous les paramétrages nécessaires pour pouvoir invoquer *un point d'accès précis sur le réseau*.
- Un *port* (*alias "endPoint"*) permet d'atteindre une *implémentation* physique d'un service publié sur un serveur. La principale information rattachée est l'**URL** permettant d'invoquer le service via SOAP.
- La partie "**binding**" (référéncée par un "port") permet d'*associer/relier* entre eux les différents éléments suivants:
 - * *interface abstraite (portType)* et ses opérations
 - * *protocole de transport (HTTP + détails ou)*
 - * *point d'accès (port)*

XV - Annexe – Services REST et JAX-RS

1. Web Services "R.E.S.T."

1.1. REST

Un style d'architecture

REST est l'acronyme de **Representational State Transfert**. C'est un **style d'architecture** qui a été décrit par **Roy Thomas Fielding** dans sa thèse «*Architectural Styles and the Design of Network-based Software Architectures*».

Architectures orientées ressource

L'information de base, dans une architecture REST, est appelée **ressource**. Toute information qui peut être nommée est une ressource: un article d'un journal, une photo, un service ou n'importe quel concept.

Si la valeur d'une ressource peut changer au cours du temps, il est important de noter que la sémantique de la ressource, elle, ne devra jamais changer.

Une ressource est identifiée par un **identificateur de ressource**. Il permet aux composants de l'architecture d'identifier les ressources qu'ils manipulent. Sur le web ces identificateurs sont les **URI** (Uniform Resource Identifier).

Les composants de l'architecture REST manipulent ces ressources en transférant des **représentations de ces ressources**. Sur le web, on trouve aujourd'hui le plus souvent des représentations au format HTML, XML ou JSON.

`http://exemple.org/maMethode?p1=v1&p2=v2`

REST s'appuie à fond sur HTTP et n'impose pas un encodage XML (contrairement à SOAP).

Les méthodes HTTP sont utilisées pour indiquer la sémantique des actions demandées :

- **GET** : lecture d'information
- **POST** : envoi d'information
- **PUT** : mise à jour d'information
- **DELETE** : suppression d'information

Par exemple, pour récupérer la liste des adhérents de mon club, j'effectue une requête de type **GET** vers la ressource **http://monsite.com/adherents**. Pour obtenir que les adhérents ayant plus de 20 ans, la requête devient **http://monsite.com/adherents?ageMinimum=20**.

Pour supprimer les adhérents 3 et 4, on peut employer une requête de type **DELETE** telle que **http://monsite.com/adherents/3/4**.

Pour envoyer des informations, on utilise **POST** ou **PUT** en passant les informations dans le corps du message.

Pour décrire les méthodes d'un service REST --> WADL (pas normalisé) ou bien extension de WSDL 2.

2. API java pour REST (JAX-RS)

JAX-RS est une API java très récente (depuis les spécifications JEE 6) qui permet de mettre en œuvre des services REST en JAVA :

Une classe java avec annotations JAX-RS sera exposée comme web service REST.

JAX-RS se limite à l'implémentation serveur, la spécification ne propose rien du côté client

Pour implémenter les services REST, on utilise principalement les annotations JAX-RS suivantes:

- **@Path** : définit le chemin de la ressource. Cette annotation se place sur la classe et/ou sur la méthode implémentant le service.
- **@GET, @PUT, @POST, @DELETE** : définit le verbe implémenté par le service
- **@Produces** spécifie le ou les Types MIME de la réponse du service
- **@Consumes** : spécifie le ou les Types MIME acceptés en entrée du service

Les principales implémentations de JAX-RS sont :

- [Jersey](#), implémentation de référence de SUN
- [Resteasy](#), l'implémentation jboss
- [CXF](#)
- [Restlet](#)

2.1. Mise en oeuvre de JAX-RS avec CXF

NB: les versions récentes de CXF (>=2.2) implémentent maintenant JAX-RS (en plus de JAX-WS)

WEB-INF/web.xml (pour spring + cxf):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
...
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/classes/deploy-context.xml</param-value>
</context-param>

<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<servlet>
  <servlet-name>CXFServlet</servlet-name>
  <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>CXFServlet</servlet-name>
  <url-pattern>/services/*</url-pattern>
</servlet-mapping>
</web-app>
```

Configuration "Spring/CXF" pour JAX-RS:

deploy-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxrs="http://cxf.apache.org/jaxrs"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://cxf.apache.org/jaxrs
                           http://cxf.apache.org/schemas/jaxrs.xsd"
       default-lazy-init="false">

    <import resource="classpath:META-INF/cxf/cxf.xml" />
    <!-- <import resource="classpath:META-INF/cxf/cxf-extension-jaxrs-binding.xml" /> for old cxf version -->
    <import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

    <!-- url complete de type "http://localhost:8080/mywebapp/services/rest/myService/xxx"
         avec "services" associe à l'url-pattern de CxfServlet dans web.xml
         et myservices/xxx associe aux valeurs de @Path() de la classe java et des méthodes
    -->

    <jaxrs:server id="myRestServices" address="/rest">
        <jaxrs:serviceBeans>
            <ref bean="serviceImpl" />
            <!-- <ref bean="service2Impl" /> -->
        </jaxrs:serviceBeans>
        <jaxrs:extensionMappings>
            <entry key="xml" value="application/xml" />          <entry key="json" value="application/json" />
        </jaxrs:extensionMappings>
    </jaxrs:server>

    <bean id="serviceImpl" class="service.ServiceImpl" />
    <bean id="service2Impl" class="service.Service2Impl" />
</beans>
```

NB : La version **2.5.5** de CXF génère bien (automatiquement) du JSON.

Les versions 2.6 et 2.7 sont censés le faire (différemment) mais configuration ????

User pojo:

```
package pojo;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "user") /* pour préciser balise xml englobante et pour jaxb2 */
public class User {
    private Integer id; // +get/set
    private String name; // +get/set

    @Override
    public String toString() {
        return String.format("{id=%s,name=%s}", id, name);
    }
}
```

classe d'implémentation du service:

```

package service;

import java.util.HashMap; import java.util.Map;

import javax.ws.rs.GET;          import javax.ws.rs.Path;
import javax.ws.rs.PathParam;    import javax.ws.rs.QueryParam;
import javax.ws.rs.Produces;     import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

import pojo.User;

@Path("/myservice/")
@Produces("application/xml") //par défaut pour toutes les méthodes de la classe
public class ServiceImpl /* implements ServiceDefn */{ //pas d'interface obligatoire

    private static Map<Integer,User> users = new HashMap<Integer,User>();
    static {
        users.put(1, new User(1, "foo"));    users.put(2, new User(2, "bar"));
    }
    public ServiceImpl() {
    }

    @GET
    @Path("/users")
    public Collection<User> getUsers() {
        return users.values();
    }

    @GET
    @Path("/user/{id}")
    // pour URL = http://localhost:8080/mywebapp/services/rest/myservice/user/2
    public User getUser(@PathParam("id") Integer id) {
        return users.get(id);
    }

    @GET
    @Path("/utilisateur")
    // pour URL = http://localhost:8080/mywebapp/services/rest/myservice/utilisateur?id=2
    // et quelquefois ...?p1=val1&p2=val2&p3=val3
    public User getUserV2(@QueryParam("id") Integer id) {
        return users.get(id);
    }

    @GET
    @Path("/users/bad")
    public Response getBadRequest() {
        return Response.status(Status.BAD_REQUEST).build();
    }

    @POST // (REST conseille POST pour des ajouts )
    @Path("/users/add")
    // pour action URL = http://localhost:8080/mywebapp/services/rest/myservice/users/add
    // dans form avec <input name="id" /> et <input name="name" /> et method="POST"
    public Response addNewUser(@FormParam("id") Integer id,@FormParam("name") String name) {
        users.put(id,new User(id,name));
        return Response.status(Status.OK).build();
    }
}

```

http://localhost:8080/mywebapp/services/rest/myservice/users/

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<userCollection>
  <users>
    <user><id>1</id><name>foo</name></user>
    <user><id>2</id><name>bar</name></user>
  </users>
</userCollection>
```

http://localhost:8080/mywebapp/services/rest/myservice/user/1

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user>
  <id>1</id>
  <name>foo</name>
</user>
```

Quelques autres exemples :

```
@GET
@Path("/euroToFranc/{s}")
@Produces("text/plain")
public double euroToFranc(@PathParam("s")double s){
    return s*6.55957 ;
}
```

renvoie directement la valeur en mode texte (sans aucune balise)

....

```
@PUT
@Path("/update")
public Response updateDeviseChange(@FormParam("name")String name,
                                   @FormParam("change")double newChange){
    Devise devise = mapDevises.get(name);
    if(devise!=null){
        devise.setChange(newChange);
        return Response.status(Status.OK).build();
    }
    else return Response.status(Status.NOT_FOUND).build();
}
```

//même principe pour @DELETE

Configuration maven type pour cxf :

```
<properties>
  <org.springframework.version>3.1.2.RELEASE</org.springframework.version>
  <org.apache.cxf.version>2.5.5</org.apache.cxf.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-api</artifactId>
    <version>${org.apache.cxf.version}</version>
    <scope>compile</scope>
  </dependency>

  <dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-frontend-jaxrs</artifactId>
    <version>${org.apache.cxf.version}</version>
```



```

</dependency>

<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-frontend-jaxws</artifactId>
  <version>${org.apache.cxf.version}</version>
  <exclusions>
    <exclusion>
      <groupId>asm</groupId>
      <artifactId>asm</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.apache.geronimo.specs</groupId>
      <artifactId>geronimo-javamail_1.4_spec</artifactId>
    </exclusion>
  </exclusions>
</dependency>

<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-ws-security</artifactId>
  <version>${org.apache.cxf.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-transports-http</artifactId>
  <version>${org.apache.cxf.version}</version>
</dependency>
</dependencies>

```

2.2. Appel de webServices REST en java via l'API "httpclient"

L'api java JAX-RS n'est utile que du coté serveur . Pour le coté client des services web REST en java on peut utiliser une **API** open-source du monde *apache* spécialisée dans les appels http (en modes GET, POST,PUT, DELETE) : **httpclient** (de httpcomponents)

```

<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId> <!-- and indirectly httpcore -->
  <version>4.2.1</version>
</dependency>

```

Quelques exemples :

```

package client;

import java.net.URI;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

public class CalculateurClientRestApp {

    public static void main(String[] args) {
        // test du web service rest via org.apache.httpcomponents/httpclient
        try {
            String restAppPart = "/wsCalculateur/services/rest";

```

```

        URIBuilder builder = new URIBuilder();
        builder.setScheme("http").setHost("localhost").setPort(8080)
            .setPath(restAppPart + "/calculateur/addition")
            .setParameter("a", "5")
            .setParameter("b", "6");
        URI uri = builder.build();
        String url5plus6=uri.toString();
        System.out.println("REST GET URL="+url5plus6);

        HttpClient httpclient = new DefaultHttpClient();
        HttpGet httpget = new HttpGet(url5plus6);
        HttpResponse response = httpclient.execute(httpget);

        String res5plus6=EntityUtils.toString(response.getEntity());

        System.out.println("5+6="+res5plus6);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

public static void testCreateNewDevise(String name,String change) {
    try {
        ...
        HttpPost httppost = new HttpPost(urlCreateDevises);

        List<NameValuePair> formparams = new ArrayList<NameValuePair>();
        formparams.add(new BasicNameValuePair("name", name));
        formparams.add(new BasicNameValuePair("change", change));
        UrlEncodedFormEntity entity= new UrlEncodedFormEntity(formparams, "UTF-8");
        httppost.setEntity(entity);
        HttpResponse response = httpclient.execute(httppost);
        System.out.println("response:"+response.toString());
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void testUpdateDevise(String name,String change) {
    ...
    HttpPut httpput = new HttpPut(urlUpdateDevises);
    ...
    UrlEncodedFormEntity entity =new UrlEncodedFormEntity(formparams, "UTF-8");
    httpput.setEntity(entity);
    HttpResponse response = httpclient.execute(httpput);
    System.out.println("response:"+response.toString());
    ...
}

public static void testDeleteDevise(String name) {
    String restAppPart = "/wsCalculateur/services/rest";
    URIBuilder builder = new URIBuilder();
    builder.setScheme("http").setHost("localhost").setPort(8080)
        .setPath(restAppPart + "/devises/delete/" + name);
    URI uri = builder.build();
    String urlDeleteDevises=uri.toString();
    HttpClient httpclient = new DefaultHttpClient();
    HttpDelete httpdelete = new HttpDelete(urlDeleteDevises);
    HttpResponse response = httpclient.execute(httpdelete);
    System.out.println("response:"+response.toString());
    ...
}

```

XVI - Annexe – Sécurité sur les services WEB

1. Gestion de la sécurité / Services Web

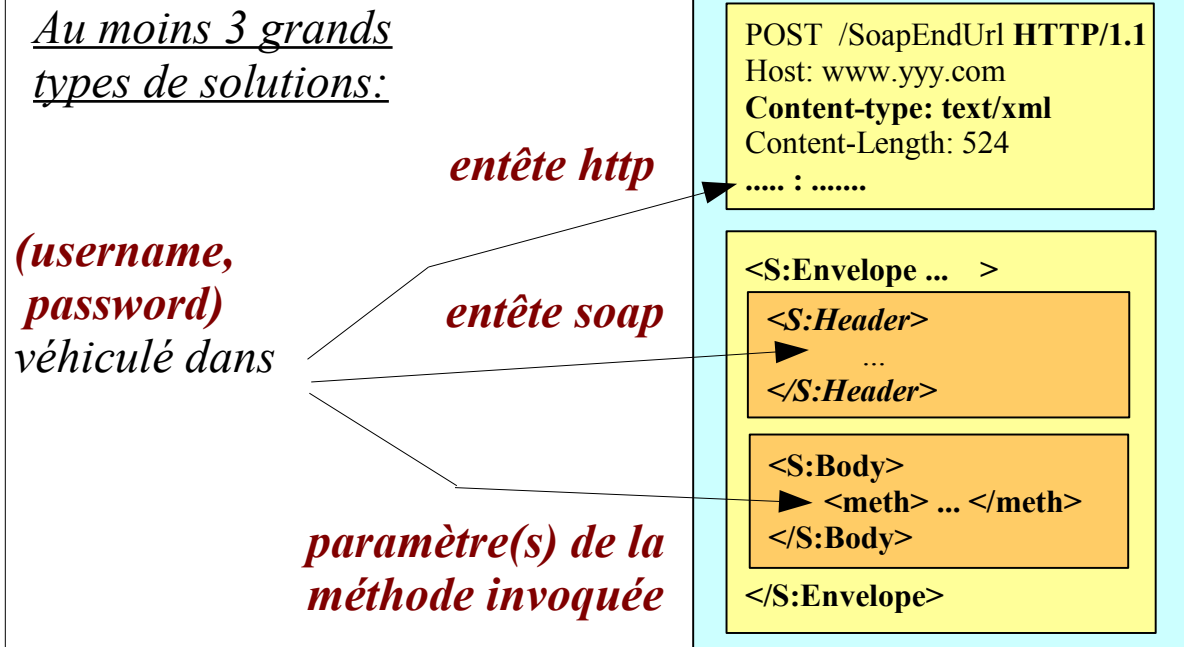
1.1. Éléments de sécurité nécessaires sur les services WEB

Sécurité liée aux services "WEB"

- ***Authentification du client*** via (*username,password*) ou autre pour contrôler les accès aux services
--> "*infos auth*" dans *entête Http* ou *entête SOAP* ou
- ***Cryptage / confidentialité***
--> via *HTTPS/SSL* et/ou cryptage XML/SOAP
- ***Signature électronique des messages*** (certificats)
- ***Intégrité des messages (vérif. non interception/modification)***
--> via *signature d'une empreinte d'un message*
- ...

1.2. différents types d'authentications (WS)

Authentification du client invoquant un service WEB



1.3. authentification élémentaire / fonctionnelle via "jeton"

La technique élémentaire d'authentification associée aux services WEB correspond à celle qui est utilisée au niveau de UDDI à savoir:

- 1) appel d'une méthode de type
 String sessToken = getSessionToken(String userName,String password)
 qui renvoie un jeton de sécurité lié à une session après avoir vérifié la validité du couple (username,password) d'une façon ou d'une autre (ex: via ldap).
- 2) passage du jeton de sécurité "sessToken" en tant qu'argument supplémentaire de toutes les autres méthodes du service Web : **methodeXy**(String sessToken, String param2,...) ;
 Une simple vérification de la validité du jeton servira à décider si le service accepte ou pas de répondre à la requête formulée.

Variantes classiques:

- repasser le jeton de sécurité/session au travers d'un argument plus large généralement appelé "contexte" (au sens "contexte technique" et non métier). Ce contexte pourra ainsi véhiculer d'autres informations techniques jugées pertinentes (ex: transactions , référence xy, ...).
- utiliser en plus SOAP over HTTPS pour crypter si besoin certaines informations échangées (ex: [username, password] ou plus).

---> gros inconvénient de la méthode "élémentaire/fonctionnelle" : chaque méthode appelée et

sécurisée comporte au moins un paramètre supplémentaire .

Autrement dit , la sécurité se voit dans les signatures des méthodes à invoquer.

1.4. Authentification véhiculée via l'entête (header) "HTTP"

En passant les informations d'authentification (username,password) ou ... dans l'entête HTTP on bénéficie des avantages suivants:

- authentification bien séparée des aspects fonctionnels
- réutilisation d'une fonctionnalité standard d'HTTP "Authentification basique HTTP" (username/password) véhiculés de façon un peu crypté dans un champ de l'entête HTTP.
- paramétrage/codage assez simple à faire via des intercepteurs ou API des technologies "Web Services" (ex: *intercepteur "cxf"* , *interface "BindingProvider" de JAX-WS*)

Pour encore plus de sécurité , on peut utiliser conjointement SSL/HTTPS .

Cette méthode est très bien dans la plupart des cas "ordinaires" (non pointus).

Limitation:

Dans certains cas pointus , l'enveloppe SOAP est relayée par différents intermédiaires (ex: ESB ,) . Certains maillons du transport peuvent être effectués par d'autres protocoles que HTTP et les informations de l'entête HTTP risquent alors de ne pas bien être retransmises jusqu'à la destination prévue.

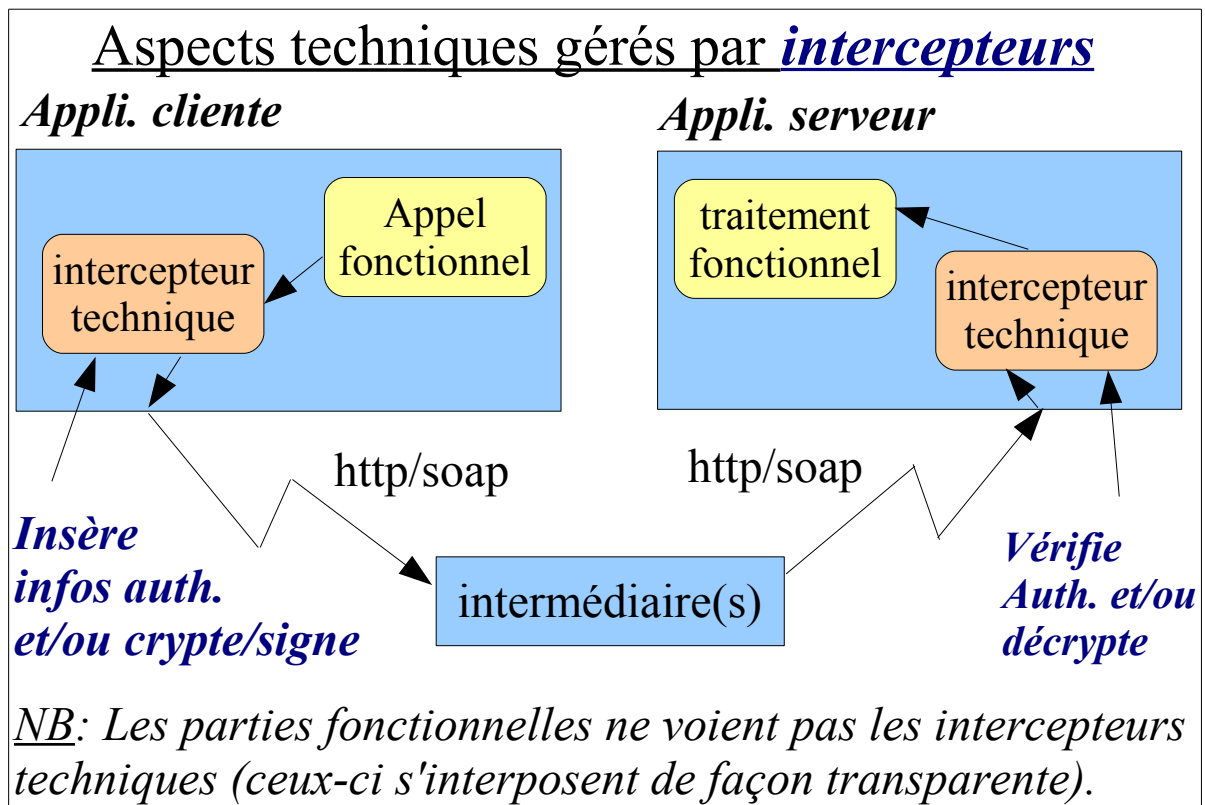
1.5. Authentification véhiculée via l'entête (header) "soap"

Avantage: fonctionne même si plusieurs protocoles de transport (HTTP puis JMS puis ...) plein d'options permettant une authentification forte (multicritères , très sécurisée)

Inconvénients :

- API et/ou intercepteur(s) un peu plus complexe(s)
- Les technologies "client" et "serveur" doivent toutes les deux être compatibles sur la gestion des entêtes SOAP .

1.6. Aspects techniques gérés par intercepteurs



1.7. Problématique (sécurité avancée)

Bien que permettant globalement une interaction entre 2 parties (client et serveur), la technologie des services WEB (XML sur HTTP) fait intervenir de multiples intermédiaires à travers le réseau internet.

Le contexte de sécurité SOAP (idéalement géré de bout en bout mais pas toujours pour des raisons de coûts) doit être suffisamment perfectionné pour supporter des éventuelles attaques ou menaces (interception des messages, altération de ceux-ci, ...).

D'autre part, tout service n'est pas forcément gratuit et un accord préalable (abonnement, ...) peut déboucher sur la génération d'un compte utilisateur (username, password) permettant d'accorder finement des droits d'accès au service Web.

Recommandation du ws-i en terme de sécurité:

==> utiliser **HTTPS** (avec SSL 3.0 ou TLS 1.0) et non pas HTTP pour véhiculer l'enveloppe SOAP. Cette recommandation est toutefois nuancée et insiste sur le compromis **sécurité/sur-coûts** qui doit être évalué au cas par cas.

Dans certains cas SSL (prévu pour un mode point à point) ne suffit pas. Il faut alors utiliser quelques unes des technologies présentées ci-après.

1.8. Web Service Security (WS-S)

Développé par l'organisme OASIS, **WS-S** est une **extension de SOAP** permettant de contrôler de bout en bout les éléments suivants sur certains messages:

- intégrité (via *Xml-Signature* et *Security token* (ex: *certificat X-509* ou *ticket kerberos* encapsulés dans des éléments XML))
- confidentialité (via *Xml-Encryption* + *Security token*)
- authentication

NB:

WS-S indique comment représenter en XML des jetons de sécurité mais n'impose pas un type précis de certificat .

Ces jetons (certificats) servent essentiellement à s'assurer que les messages SOAP n'ont pas été interceptés ni modifiés.

Un peu à la façon de SSL , WS-S agit de façon transparente en ajoutant automatiquement des éléments de sécurité sur les messages "SOAP" .

Exemple (partiel):

```
<SOAP:Envelope xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP:Header>
    <wsse:Security
      xmlns:wsse='http://schemas.xmlsoap.org/ws/2002/07/secext'>
      <Signature xmlns='http://www.w3.org/2000/09/xmldsig#'>
        ...defined below...
      </Signature>
    </wsse:Security>
  </SOAP:Header>
  <SOAP:Body id='Body'>
    ...message body...
  </SOAP:Body>
</SOAP:Envelope>
```

1.9. Aspects avancés liés à la sécurité

==> Approfondir si besoin WS-Security (Xml-Signature, Xml-encryption, ...) pour les cas "pointus" .

Les API sophistiquées (java , .net) telles que CXF prennent généralement en sorte WS-Security.

Il existe également un lien avec **Security assertion markup language (SAML)** (sorte de "SSO / Single Sign On" en version Web) .

2. Authentification basique http avec JAX-WS

L'authentification basique Http (la plus simple/classique/interopérable) consiste à véhiculer les informations d'authentification (basiquement cryptées) dans l'entête Http des requêtes Http véhiculant l'enveloppe SOAP .

2.1. Authentification basique Http coté client

L'interface cachée "**BindingProvider**" peut servir à renseigner les paramètres d'authentification coté client:

```
Calculateur calculateur = serviceCalculateur.getCalculateurServicePort();
// ou calculateur = (Calculateur) service.getPort(PORT_NAME, Calculateur.class);
javax.xml.ws.BindingProvider bp = (javax.xml.ws.BindingProvider) calculateur;
```

```

Map<String,Object> context = bp.getRequestContext();
/*context.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
               "http://localhost:1234/myWebApp/services/calculateur");*/
context.put(BindingProvider.USERNAME_PROPERTY,"userNameSiBasicHttpAuth");
context.put(BindingProvider.PASSWORD_PROPERTY, "pwdSiBasicHttpAuth");
System.out.println(calculateur.getTva(200, 19.6));

```

ou bien avec intercepteur coté client.

2.2. Authentification basique Http coté serveur (avec CXF)

Récupérer le code de *basicHttpSecurityInterceptor* et *UserMapAuthManager*.

Ajouter cette configuration spring:

```

<bean id="userMapAuthManager" class="generic.util.auth.local.UserMapAuthManager">
    <property name="users"><map>
        <entry key="user1" value="pwd1"/>
        <entry key="user2" value="pwd2"/>
    </map></property>
</bean>
...
<bean id="basicHttpSecurityInterceptor"
    class="generic.ws.util.interceptor.cxf.CxfBasicAuthInterceptor">
    <property name="authManager" ref="userMapAuthManager"/>
    <!-- test avec mode = "preemptive" dans soap-ui recent -->
</bean>

<jaxws:endpoint id="calculateurEndPoint"
    implementor="#calculateurService_impl" address="/calculateur" >
    <jaxws:inInterceptors>
        <ref bean="basicHttpSecurityInterceptor"/>
    </jaxws:inInterceptors>
</jaxws:endpoint>

```

code java de la classe utilitaire **CxfBasicAuthInterceptor**

```

package generic.ws.util.interceptor.cxf;

import generic.util.auth.AuthManager;

import java.io.IOException;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.util.Arrays;
import java.util.List;
import java.util.Map;

import org.apache.cxf.binding.soap.interceptor.SoapHeaderInterceptor;
import org.apache.cxf.configuration.security.AuthorizationPolicy;
import org.apache.cxf.endpoint.Endpoint;
import org.apache.cxf.interceptor.Fault;
import org.apache.cxf.message.Exchange;
import org.apache.cxf.message.Message;

```



```

import org.apache.cxf.transport.Conduit;
import org.apache.cxf.ws.addressing.EndpointReferenceType;
import org.apache.log4j.Logger;

/**
 * CXF Interceptor that provides HTTP Basic Authentication validation. Based on
 * the concepts outline here:
 * http://chrisdail.com/2008/03/31/apache-cxf-with-http-basic-authentication
 *
 * @author CDail & D. Defrance
 */
public class CxfBasicAuthInterceptor extends SoapHeaderInterceptor {
    protected Logger log = Logger.getLogger(getClass());

    private AuthManager authManager;

    //@Required (spring injection)
    public void setAuthManager(AuthManager authManager) {
        this.authManager = authManager;
    }

    @Override
    public void handleMessage(Message message) throws Fault {
        // This is set by CXF
        AuthorizationPolicy policy = message.get(AuthorizationPolicy.class);
        // If the policy is not set, the user did not specify credentials
        // A 401 is sent to the client to indicate that authentication is required
        if (policy == null) {
            if (log.isDebugEnabled()) {
                log.debug("User attempted to log in with no credentials");
            }
            sendErrorResponse(message, HttpURLConnection.HTTP_UNAUTHORIZED);
            return;
        }
        if (log.isDebugEnabled()) {
            log.debug("Logging in use: " + policy.getUserName());
        }
        // Verify the password:
        if (! authManager.isValidPasswordForUser(policy.getUserName()).equals(policy.getPassword())) {
            log.warn("Invalid username or password for user: "
                + policy.getUserName());
            sendErrorResponse(message, HttpURLConnection.HTTP_FORBIDDEN);
        }
    }

    private void sendErrorResponse(Message message, int responseCode) {
        Message outMessage = getOutMessage(message);
        outMessage.put(Message.RESPONSE_CODE, responseCode);
        // Set the response headers
        Map<String, List<String>> responseHeaders = (Map<String, List<String>>) message
            .get(Message.PROTOCOL_HEADERS);
        if (responseHeaders != null) {
            responseHeaders.put("WWW-Authenticate",
                Arrays.asList(new String[] { "Basic realm=realm" }));
            responseHeaders.put("Content-Length",
                Arrays.asList(new String[] { "0" }));
        }
        message.getInterceptorChain().abort();
        try {
            getConduit(message).prepare(outMessage);
            close(outMessage);
        } catch (IOException e) {
            log.warn(e.getMessage(), e);
        }
    }
}

```

```

    }
}

private Message getOutMessage(Message inMessage) {
    Exchange exchange = inMessage.getExchange();
    Message outMessage = exchange.getOutMessage();
    if (outMessage == null) {
        Endpoint endpoint = exchange.get(Endpoint.class);
        outMessage = endpoint.getBinding().createMessage();
        exchange.setOutMessage(outMessage);
    }
    outMessage.putAll(inMessage);
    return outMessage;
}

private Conduit getConduit(Message inMessage) throws IOException {
    Exchange exchange = inMessage.getExchange();
    EndpointReferenceType target = exchange
        .get(EndpointReferenceType.class);
    Conduit conduit = exchange.getDestination().getBackChannel(inMessage,
        null, target);
    exchange.setConduit(conduit);
    return conduit;
}

private void close(Message outMessage) throws IOException {
    OutputStream os = outMessage.getContent(OutputStream.class);
    os.flush();
    os.close();
}
}

```

interface abstraite (utilitaire) **AuthManager**

```

package generic.util.auth;

public interface AuthManager {

    //public Boolean verifyAuth(String username,String password);

    /**
     * @param username
     * @return valid password for user if user exist
     *         null if user doesn't exist or if not implemented
     *
     * NB: une callback ws-security a absolument besoin de cette méthode
     *      qui doit donc être implémentée
     */
    public String getValidPasswordForUser(String username);
}

```

Implémentation simple via une map de (username,password) en mémoire

```

package generic.util.auth.local;

import generic.util.auth.AuthManager;

import java.util.Map;

import org.apache.log4j.Logger;

public class UserMapAuthManager implements AuthManager {

    protected Logger log = Logger.getLogger(getClass());
}

```

```

/** Map of allowed users to this system with their corresponding passwords. */
private Map<String, String> users;

public void setUsers(Map<String, String> users) {
    this.users = users;
}

@Override
public String getValidPasswordForUser(String username) {
    String validPwd=null;
    try {
        validPwd = users.get(username);
    } catch (Exception e) {
        log.error(e);
        e.printStackTrace();
    }
    return validPwd;
}
}

```

autre implémentation basée sur l'extension "Spring-security"

```

package generic.util.auth.local;

import generic.util.auth.AuthManager;

import org.apache.log4j.Logger;
import org.springframework.security.core.userdetails.UserDetailsService;

public class SpringSecurityUserSerciceAuthManagerAdapter implements AuthManager
{

    private UserDetailsService userDetailsService; //of spring-security

    //for spring injection
    public void setUserDetailsService(UserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    protected Logger log = Logger.getLogger(getClass());

    @Override
    public String getValidPasswordForUser(String username) {
        String validPwd=null;
        try {
            validPwd =
                userDetailsService.loadUserByUsername(username).getPassword();
        } catch (Exception e) {
            log.error(e);
            e.printStackTrace();
        }
        return validPwd;
    }
}

```

Configuration alternative avec spring-security :

security-config.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:security="http://www.springframework.org/schema/security"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

```

```

http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.1.xsd"

<security:authentication-manager id="springSecurityAuthenticationManager">
  <security:authentication-provider>

    <security:user-service id="simpleUserDetailsService">
      <security:user name="user1" password="pwd1" authorities="user" />
      <security:user name="user2" password="pwd2" authorities="user" />
    </security:user-service>
  </security:authentication-provider>
</security:authentication-manager>
</beans>

```

et

```

<import resource="security-config.xml" />
...
<bean id="userMapAuthManagerViaSpringSecurity"
class="generic.util.auth.local.SpringSecurityUserServiceAuthManagerAdapter">
  <property name="userService" ref="simpleUserDetailsService" />
</bean>

<bean id="basicHttpSecurityInterceptor"
class="generic.ws.util.interceptor.cxf.CxfBasicAuthInterceptor">
  <!-- <property name="authManager" ref="userMapAuthManager"/> -->
  <property name="authManager" ref="userMapAuthManagerViaSpringSecurity"/>
</bean>
...

```

et avec

```

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-core</artifactId>
  <version>${org.springframework.version}</version>
</dependency>
et <artifactId>spring-security-web</artifactId> et <artifactId>spring-security-config</artifactId>
dans pom.xml

```

3. Authentification soap via WS-Security et jax-ws/cxf

Cette configuration alternative de la sécurité utilise l'entête de l'enveloppe soap.

3.1. Configuration WS-Security coté serveur avec Spring+CXF:

```

<bean id="myPasswordCallback" class="generic.ws.util.auth.wss.PasswordCallback" >
  <!-- <property name="authManager" ref="userMapAuthManager"/> -->
  <property name="authManager" ref="userMapAuthManagerViaSpringSecurity"/> <!--
test avec mode ="preemptive" et outgoing=config de niveau projet dans soap-ui recent -->
</bean>

<bean id="wssSecurityInterceptor"
class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor" >

```

```

<constructor-arg>
  <map>
    <entry key="action" value="UsernameToken"/>
    <!-- <entry key="passwordType" value="PasswordDigest"/> --> <!-- text by default -->
    <!-- <entry key="signaturePropFile" value="..." -->
    <entry key="passwordCallbackRef">
      <ref bean="myPasswordCallback"/>
    </entry>
  </map>
</constructor-arg>
</bean>

<jaxws:endpoint id="wss_serviceCalculateur_soap_endPoint"
  implementor="#calculateurImpl" address="/wssSecureCalculateur">
  <jaxws:inInterceptors>
    <ref bean="wssSecurityInterceptor"/>
  </jaxws:inInterceptors>
</jaxws:endpoint>

```

code de la classe **PasswordCallback** :

```

package generic.ws.util.auth.wss;

import generic.util.auth.AuthManager;

import java.io.IOException;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.apache.ws.security.WSPasswordCallback;

public class PasswordCallback implements CallbackHandler
{
    private AuthManager authManager;

    //@Required (spring injection)
    public void setAuthManager(AuthManager authManager) {
        this.authManager = authManager;
    }

    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException
    {
        WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];
        // WSPasswordCallback is an indirect dependency of cxf-rt-ws-security

        String username = pc.getIdentifier();
        String validPwdToCompare = authManager.getValidPasswordForUser(username);
        pc.setPassword(validPwdToCompare);
    }
}

```

```

    if (pc.getIdentifiant().equals("user1"))
    {
        pc.setPassword("pwd1");
    }
    else if (pc.getIdentifiant().equals("user2"))
    {
        pc.setPassword("pwd2");
    }*/
}
}

```

et avec dépendance maven

```

<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxfrtwssecurity</artifactId>
    <version>${org.apache.cxf.version}</version>
</dependency>

```

3.2. Configuration WS-Security coté client avec cxf

wsXxx.properties

```

wsdl.url=http://localhost:8080/wsCalculateur/services/calculateur?wsdl
soap.url=http://localhost:8080/wsCalculateur/services/wssSecureCalculateur
username=user1
password=pwd1

```

```

CalculateurImplService calculateurSoapService = new CalculateurImplService();
Calculateur calculateur = calculateurSoapService.getCalculateurImplPort();

CxfJaxWsClientPropertiesUtil jaxWsClientPropertiesUtil
    = new CxfJaxWsClientPropertiesUtil("wsXxx.properties");
jaxWsClientPropertiesUtil.setWssAuthFromProperties(calculateur);
jaxWsClientPropertiesUtil.setEndpointUrlFromProperties(calculateur);
System.out.println("3+5="+calculateur.addition(3, 5));

```

code des classes utilitaires **CxfJaxWsClientPropertiesUtil** et **CxfWSSUsernameTokenSecurityUtil**, **CxfWSSUsernameTokenSecurityUtil** et **WsBindingProviderUtil**:

```

package generic.ws.util.client.cxf;

import generic.ws.util.client.WsBindingProviderUtil;
import generic.ws.util.client.wss.SimpleWssHandlerResolverSettings;
import generic.ws.util.client.wss.cxf.CxfWSSUsernameTokenSecurityUtil;

import java.io.IOException;
import java.util.Properties;

import javax.xml.ws.Service;

/**
 * @author Didier DEFRANCE
 *
 * Classe utilitaire pour paramétrer un appel de web service SOAP
 * version simplifiée avec cxf mais nécessitant quelques librairies de cxf
 * en plus du jdk >= 1.6 .
 */

public class CxfJaxWsClientPropertiesUtil {

```

```

private String propertiesFileName;
Properties props = new Properties(); //java.util

public CxfJaxWsClientPropertiesUtil() {
    super();
}

public CxfJaxWsClientPropertiesUtil(String propertiesFileName) {
    super();
    this.propertiesFileName = propertiesFileName;
    initProperties();
}

public void setBasicHttpAuthFromProperties(Object wsProxy) {
    WsBindingProviderUtil.setBasicHttpAuth(wsProxy,
        props.getProperty("username"),
        props.getProperty("password"));
}

public void setWssAuthFromProperties(Object wsProxy) {
    CxfWSSUsernameTokenSecurityUtil cxfWSSUsernameTokenSecurityUtil
        = new CxfWSSUsernameTokenSecurityUtil(props.getProperty("username"),
            props.getProperty("password"));

    cxfWSSUsernameTokenSecurityUtil.setWssSecurityViaCxf(wsProxy);
}

public void setEndpointUrlFromProperties(Object wsProxy) {
    WsBindingProviderUtil.setSoapEndpointUrl(wsProxy,
        props.getProperty("soap.url"));
}

private void initProperties() {
    try {
        props.load(Thread.currentThread().getContextClassLoader()
            .getResourceAsStream(propertiesFileName));
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public String getPropertiesFileName() {
    return propertiesFileName;
}

//injection
public void setPropertyFileName(String propertiesFileName) {
    this.propertiesFileName = propertiesFileName;
    initProperties();
}
}

```

```

package generic.ws.util.client.wss.cxf;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

```

```

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.apache.cxf.frontend.ClientProxy;
import org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor;
import org.apache.ws.security.WSConstants;
import org.apache.ws.security.WSPasswordCallback;
import org.apache.ws.security.handler.WSHandlerConstants;

public class CxfWSSUsernameTokenSecurityUtil {

    private String login=null;
    private String pwd=null;

    /*classe imbriquée:*/
    public class ClientPasswordCallback implements CallbackHandler {

        public void handle(Callback[] callbacks) throws IOException,
            UnsupportedCallbackException {

            WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];

            // set the password for our message.
            pc.setPassword(pwd);
        }
    }
    /*fin classe imbriquée*/

    public void setLogin(String login) {
        this.login = login;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public CxfWSSUsernameTokenSecurityUtil() {
        super();
    }

    public CxfWSSUsernameTokenSecurityUtil(String login, String pwd) {
        super();
        this.login = login;
        this.pwd = pwd;
    }

    public void setWssSecurityViaCxf(Object wsProxy){

        Map<String, Object> outProps = new HashMap<String, Object>();
        outProps.put(WSHandlerConstants.ACTION, WSHandlerConstants.USERNAME_TOKEN);
        // Specify our username
        outProps.put(WSHandlerConstants.USER, login);
        // Password type : plain text
        outProps.put(WSHandlerConstants.PASSWORD_TYPE, WSConstants.PW_TEXT);
        // for hashed password use:
        //properties.put(WSHandlerConstants.PASSWORD_TYPE, WSConstants.PW_DIGEST);
        // Callback used to retrieve password for given user.
        outProps.put(WSHandlerConstants.PW_CALLBACK_REF,
            new ClientPasswordCallback()); //ok on inner class

        WSS4JOutInterceptor wssOut = new WSS4JOutInterceptor(outProps);

        org.apache.cxf.endpoint.Client client = ClientProxy.getClient(wsProxy);
        org.apache.cxf.endpoint.Endpoint cxfEndpoint = client.getEndpoint();
    }
}

```



```

        cxfEndpoint.getOutInterceptors().add(wssOut);
    }
}

```

```

package generic.ws.util.client;

import java.util.Map;

import javax.xml.ws.BindingProvider;

public class WsBindingProviderUtil {

    public static void setSoapEndpointUrl(Object wsProxy,String soapUrl){
        javax.xml.ws.BindingProvider bp = (javax.xml.ws.BindingProvider) wsProxy /*port*/;
        Map<String,Object> context = bp.getRequestContext();
        context.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, soapUrl);
    }

    //pour authentification basic Http , pas utile pour auth Soap / WS-security
    public static void setBasicHttpAuth(Object wsProxy,String username,String password){
        javax.xml.ws.BindingProvider bp = (javax.xml.ws.BindingProvider) wsProxy /*port*/;
        Map<String,Object> context = bp.getRequestContext();
        context.put(BindingProvider.USERNAME_PROPERTY,username);
        context.put(BindingProvider.PASSWORD_PROPERTY,password);
    }
}

```

Variante (ws-security coté client avec JAX-WS du jdk>=1.6 sans cxf) :

```

package generic.ws.util.client.wss;

import java.util.ArrayList; import java.util.List;

import javax.xml.ws.Service; import javax.xml.ws.handler.Handler;
import javax.xml.ws.handler.HandlerResolver;
import javax.xml.ws.handler.PortInfo;

public class SimpleWssHandlerResolverSettings {
    public static void setWssUsernameTokenHandlerResolver(Service s /* not Object wsProxy */,
                                                         String username,String password){
        final WSSUsernameTokenSecurityHandler wSSUsernameTokenSecurityHandler
            = new WSSUsernameTokenSecurityHandler(username,password);

        s.setHandlerResolver( new HandlerResolver() {

```

```

@Override
public List<Handler> getHandlerChain(PortInfo portInfo) {
    List<Handler> handlerList = new ArrayList<Handler>();
    handlerList.add(wSSUsernameTokenSecurityHandler);
    return handlerList;
}
});
}
}

```

et

```

package generic.ws.util.client.wss;

import java.util.Set; import java.util.TreeSet;
import javax.annotation.Resource; import javax.xml.namespace.QName;
import javax.xml.soap.SOAPElement; import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPFactory; import javax.xml.soap.SOAPHeader;
import javax.xml.ws.handler.MessageContext; import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

public class WSSUsernameTokenSecurityHandler implements SOAPHandler<SOAPMessageContext> {

    private String login=null;
    private String pwd=null;

    public void setLogin(String login) {
        this.login = login;
    }

    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    public WSSUsernameTokenSecurityHandler() {
        super();
    }

    public WSSUsernameTokenSecurityHandler(String login, String pwd) {
        super();
        this.login = login;
        this.pwd = pwd;
    }

    @Override
    public boolean handleMessage(SOAPMessageContext context) {

        Boolean outboundProperty =
            (Boolean) context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
        if (outboundProperty.booleanValue()) {

            try {
                SOAPEnvelope envelope = context.getMessage().getSOAPPart().getEnvelope();
                SOAPFactory factory = SOAPFactory.newInstance();
                String prefix = "wsse";
                String uri = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd";
                SOAPElement securityElem =
                    factory.createElement("Security", prefix, uri);
                SOAPElement tokenElem =
                    factory.createElement("UsernameToken", prefix, uri);
                tokenElem.addAttribute(QName.valueOf("wsu:Id"), "UsernameToken-2");
            }

```

```

        tokenElem.addAttribute(QName.valueOf("xmlns:wsu"),
            "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd");
        SOAPElement userElem =
            factory.createElement("Username", prefix, uri);
        userElem.addTextNode(login);
        SOAPElement pwdElem =
            factory.createElement("Password", prefix, uri);
        pwdElem.addTextNode(pwd);
        pwdElem.addAttribute(QName.valueOf("Type"),
            "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText");
        tokenElem.addChildElement(userElem);
        tokenElem.addChildElement(pwdElem);
        securityElem.addChildElement(tokenElem);
        SOAPHeader header = envelope.addHeader();
        header.addChildElement(securityElem);
    } catch (Exception e) {
        e.printStackTrace();
    }
    } else {
        // inbound
    }
    return true;
}

@Override
public Set<QName> getHeaders() {
    return new TreeSet();
}

@Override
public boolean handleFault(SOAPMessageContext context) {
    return false;
}

@Override
public void close(MessageContext context) {
    //
}
}

```

XVII - Normes JBI et SCA (présentation)

1. SCA (Service Component Architecture)

SCA (Service Component Architecture)

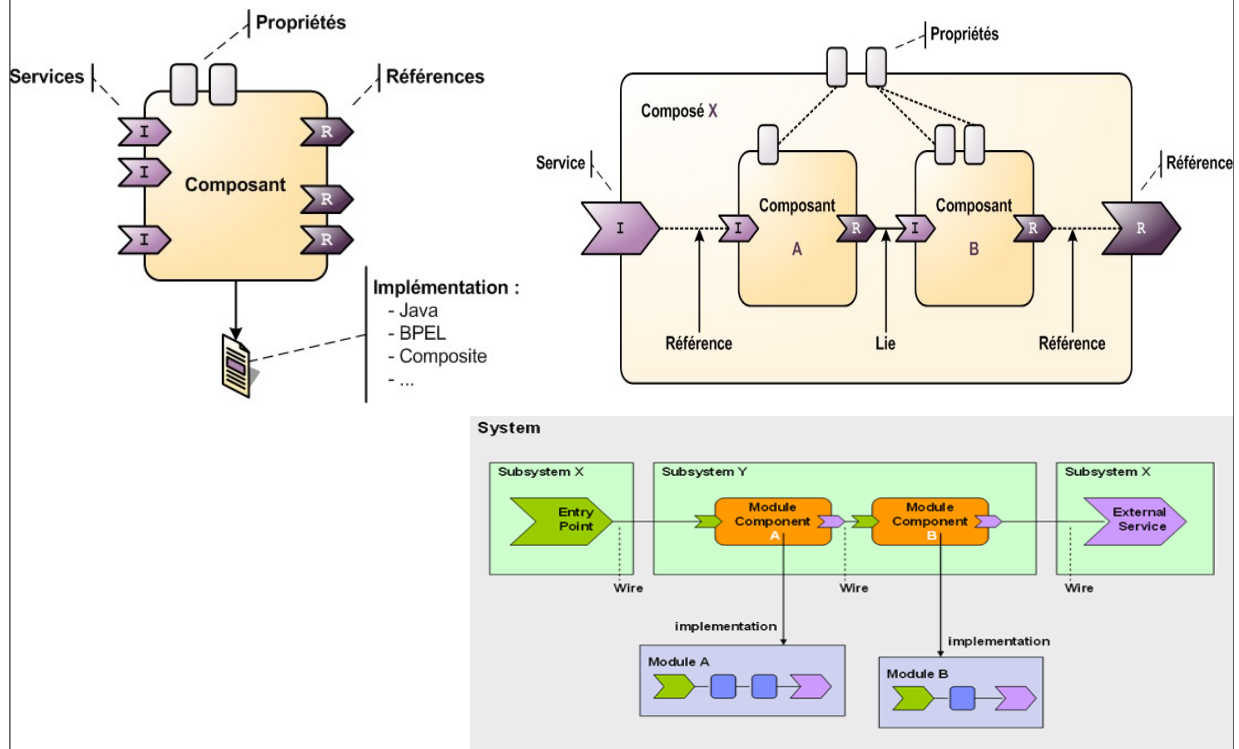
Les spécifications **S.C.A.** (d'origine *IBM*) visent à *structurer des combinaisons/assemblages de services* sur le mode "*consommateurs/fournisseurs*".

S.C.A. se veut être *indépendant des langages de programmation* (java , c++ , c# , bpel ,) .

Principales implémentations : *WebSphere ESB* (IBM)
et *Tuscany* (Apache)

Contrairement à JBI , SCA n'impose pas un environnement d'exécution (type de serveur hôte) précis . SCA est plutôt à considérer comme un modèle de structuration d'une unité de services (paramétrages/programmation/...) .

Illustrations sur l'architecture SCA



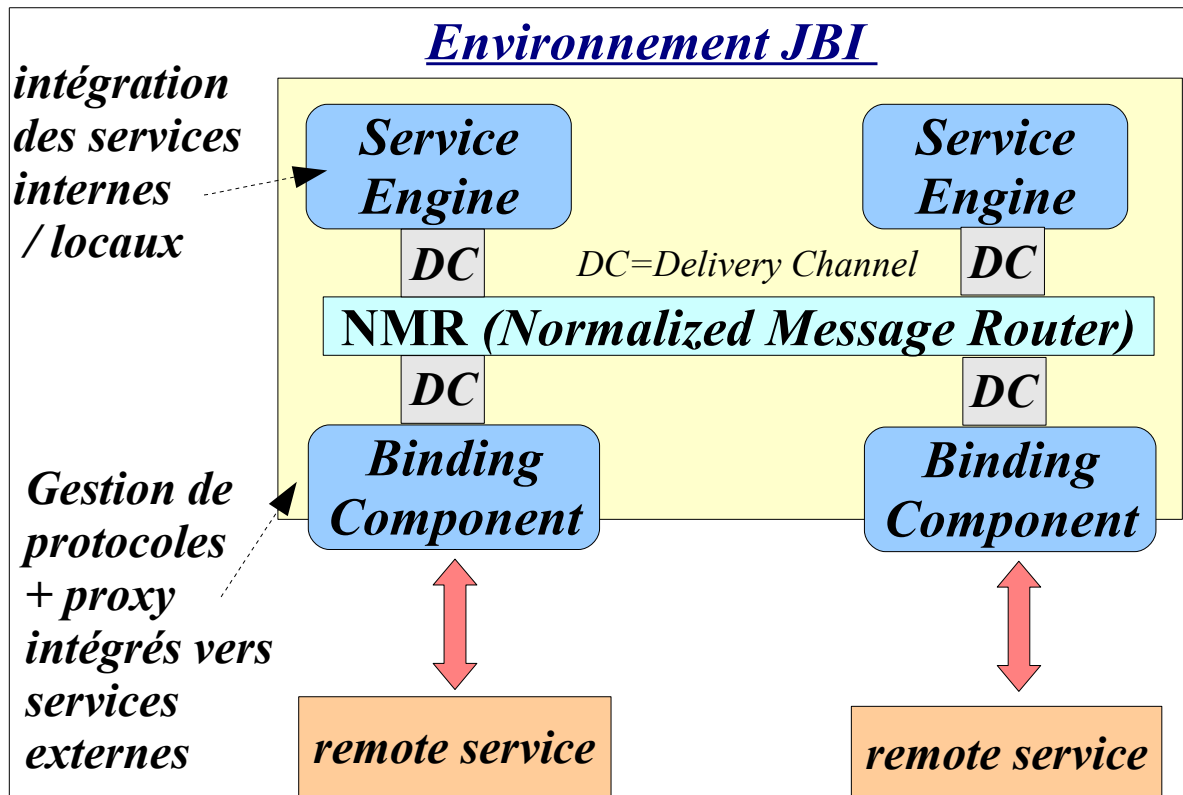
2. JBI (Java Business Integration)

JBI (Java Business Integration)

- * Les spécifications **JBI** précisent comment **intégrer des associations/combinaisons de services** (*distants ou locaux*) **dans un environnement d'exécution Java** (*ex : serveur ServiceMix*).
- * JBI propose de **standardiser** quelques éléments de configuration (à grande échelle) et de **déploiement** dans le monde Java.

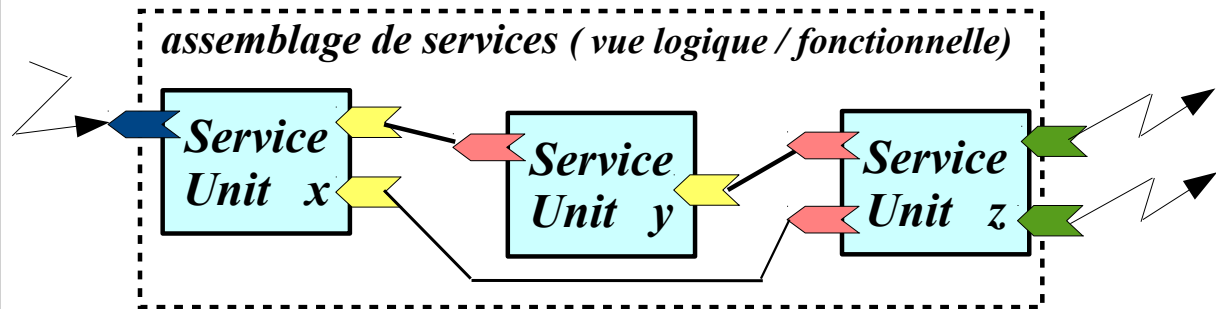
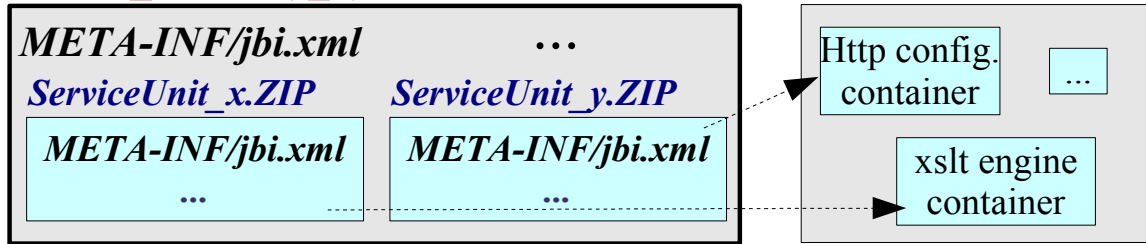
Concrètement, JBI peut être utilisé pour intégrer dans un environnement d'exécution java (Serveur):

- * certaines fonctionnalités des **ESB** (routage, proxy, ...) et les paramétrages associés (protocole, url , ...).
- * des **moteurs de services** (*ex: Tx [transformations xslt ou java], orchestration bpel ou ...*) avec des éléments à interpréter.



JBI Service Assembly (et déploiement jbi)

service_assembly_xy.ZIP à déployer dans **serveur Jbi** (ex: *serviceMix*)



Analogie: *appliXy.ear* contenant *webXy.war* et *ejbXy.jar*

3. Détails de l'architecture JBI

JBI : Détails internes (NMR , MEP, ...)

Au sein d'un environnement JBI, les échanges (communications internes entre services producteurs et services consommateurs) sont effectués selon les MEP de WSDL2 et contrôlés par le **NMR** (*Normalized Message Router*).

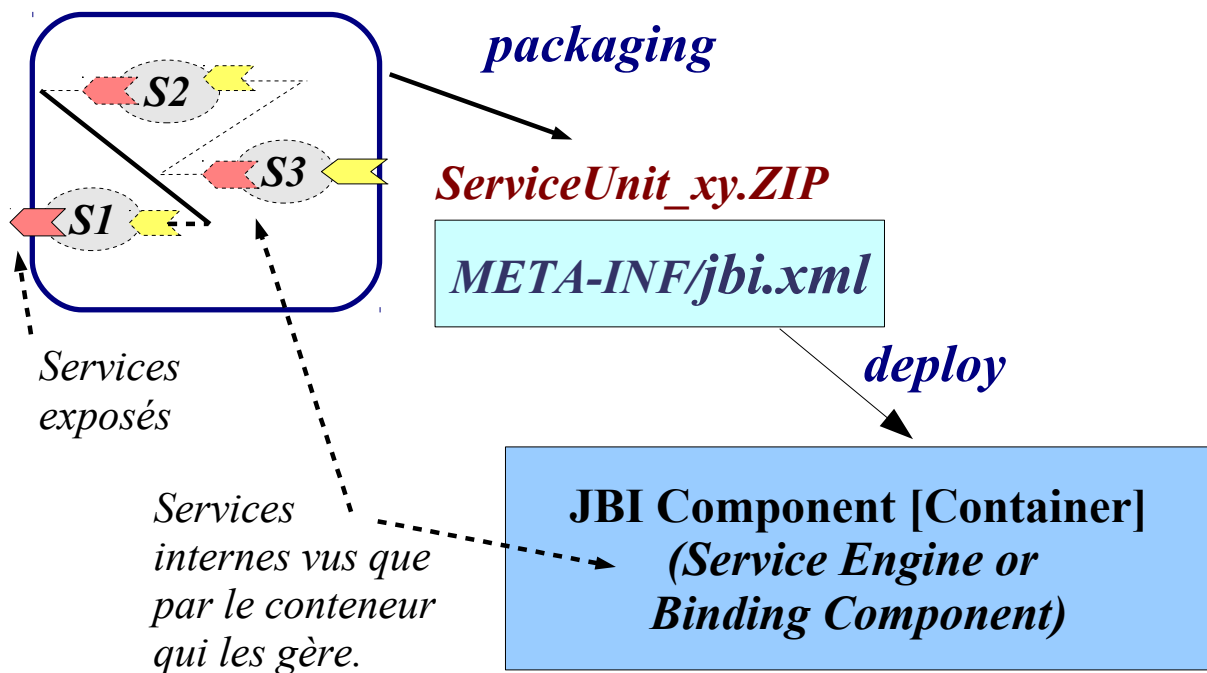
MEP = Message Exchange Patterns

MEP:

- * **in_only** (one way) , **robust in_only** (with fault)
- * **in_out** (with or without fault) , **in_optional-out**

NB: Le **NMR** joue toujours le rôle d'*intermédiaire dans les échanges* et *gère les acquittements* internes (send/accept "done" status).

Au moins une sous archive serviceUnitXY.zip par type de conteneur technique JBI (Http , Xslt , Bpel , JMS , SMTP , EIP , ...)



META-INF/jbi.xml (in the assembly MyCompositeApp.zip)

```

<jbi xmlns="http://java.sun.com/xml/ns/jbi" ... xsi:schemaLocation="... jbi.xsd">
  <service-assembly> <identification> <name>MyCompositeApp</name>
    <description>....</description> </identification>
    <service-unit> <identification> <name>MyBpelSU</name>
      <description>...</description> </identification>
      <target> <artifacts-zip>MyBpelSu.jar</artifacts-zip>
        <component-name>xxx-bpel-engine</component-name> </target>
    </service-unit>
    <service-unit> <identification> <name>CompositeApp1-xxx-http-binding</name>
      <description>.... </description> </identification>
      <target> <artifacts-zip>xxx-http-binding.jar</artifacts-zip>
        <component-name>xxx-http-binding</component-name> </target>
    </service-unit>
    <connections>
      <connection> <consumer endpoint-name="zzPort" service-name="ns1:aaService"/>
        <provider endpoint-name="xxPort" service-name="ns2:bbService"/> </connection>
      <connection> ... </connection> </connections>
    </service-assembly>
</jbi>

```

+ autres éléments spécifiques (Env. Jbi xxx)

META-INF/jbi.xml (in xxx-http-binding.jar)

```

<jbi xmlns="http://java.sun.com/xml/ns/jbi"
xmlns:ns1="http://...." .... version="1.0">
  <services binding-component="true">
    <provides endpoint-name="xxPort"
      interface-name="ns1:XxxPortType"
      service-name="ns2:bbService"/>
    <consumes endpoint-name="yyPort"
      interface-name="ns3:YyyPortType"
      service-name="ns4:ccService"/>
  </services>
</jbi>

```

*jbi.xml
for each
service-unit*

META-INF/jbi.xml (in MyBpelSu.jar)

+ ...wsdl, ...xsd,
....bpel

```

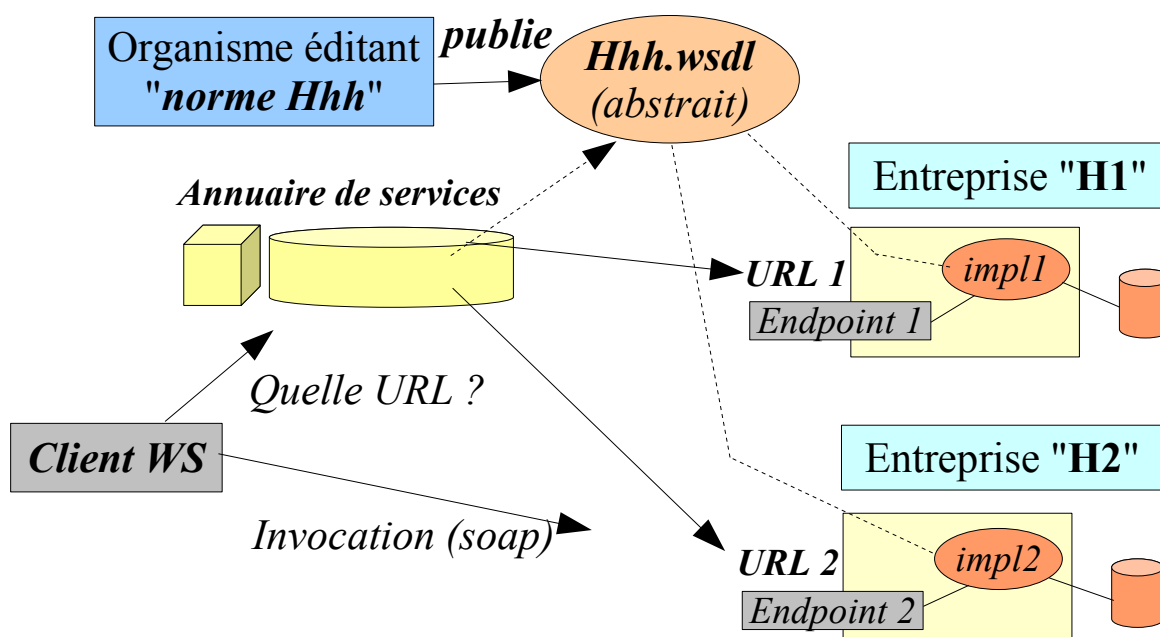
<jbi xmlns="http://java.sun.com/xml/ns/jbi" ...>
  <services binding-component="false">
    <provides ... />
    <consumes .... />
  </services>
</jbi>

```


XVIII - Annexe – Annuaire de services et UDDI

1. Utilité d'un annuaire de "service WEB"

Un **service fonctionnel** (avec *WSDL abstrait*) peut avoir **plusieurs implémentations** avec **URL(s) à découvrir** via un **annuaire**.



Un **annuaire de services** est une sorte de **base de données d'URL** avec des **critères de recherches**.

En général, dans une logique SOA, l'interrogation et la mise à jour de l'annuaire s'effectuent via des services WEB techniques.

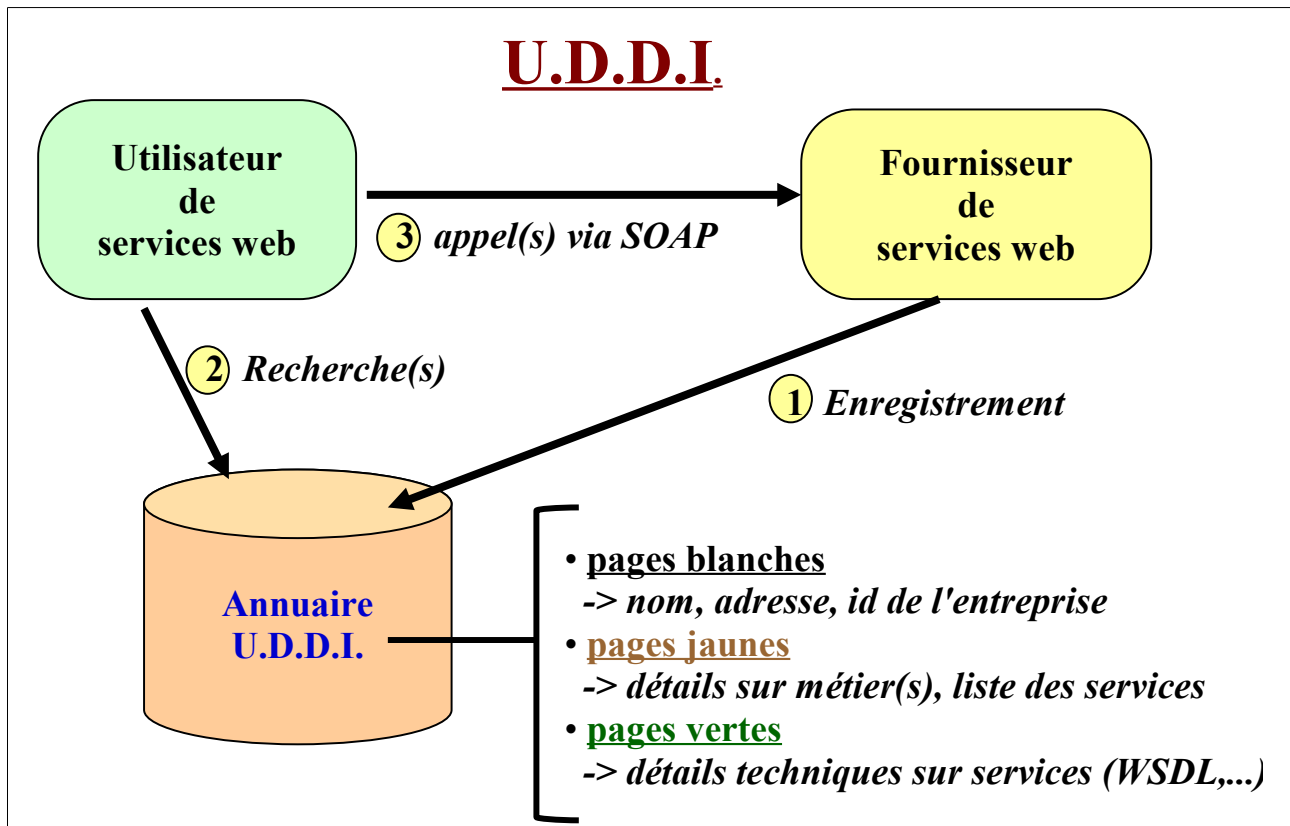
La norme **UDDI** vise à structurer la structure des annuaires de services et définit des services techniques pour la publication et l'interrogation.

Cependant, cette norme (déjà assez ancienne) est très complexe et n'est souvent utilisée qu'au sein de grandes entreprises multi-nationales (là où il faut tenir compte de plusieurs langues humaines au niveau des critères de recherche).

Une PME aura tout intérêt à utiliser un annuaire plus simple que UDDI.

2. Annuaire UDDI (présentation)

UDDI signifie *Universal Description, Discovery & Integration*



Universal Description, Discovery & Integration (<http://www.uddi.org>)

UDDI est un **service d'annuaire basé sur Xml** essentiellement prévu pour **référencer des services "web"**. Le noyau du projet UDDI est l'**UDDI Business Registry**, annuaire contenant des informations de trois types, décrites au format XML :

- **Pages blanches** : noms, adresses, contacts, identifiants,... des entreprises enregistrées. Ces informations sont décrites dans des entités de type **Business Entity**. Cette description inclut des informations de catégorisation permettant de faire des recherches spécifiques dépendant du métier de l'entreprise ;
- **Pages jaunes** : *détails sur le métier de l'entreprise, les services qu'elle propose*. Ces informations sont décrites dans des entités de type **Business Service** ;
- **Pages vertes** : *informations techniques sur les services proposés*. Les pages vertes incluent des références vers les spécifications des services Web, et les détails nécessaires à l'utilisation de ces services : interfaces implémentées, information sur les contacts pour un processus particulier, description du processus en plusieurs langages, catégorisation des processus, pointeurs vers les spécifications décrivant chaque API. Ces informations sont décrites dans deux documents : un *Binding Template*, et un *Technology Model (tModel)*.

3. Annuaire publics et annuaire privés

3.1. Principaux annuaires publics

NB: La plupart des annuaires UDDI publics (mise en place par IBM et Microsoft) ne sont plus maintenus.

Raisons prétextées:

- Technologie au point , tests terminés
- Ce n'est pas aux éditeurs de logiciels (IBM ou Microsoft , ...) de maintenir des annuaires publics (ce n'est pas leurs métiers)

Conséquences:

La technologie UDDI à un niveau privé perd un peu de son intérêt mais il faut accepter cet état de fait (tant qu'un organisme international indépendant ne prendra pas à sa charge un annuaire UDDI véritablement public).

3.2. Quelques implémentations d'annuaires UDDI

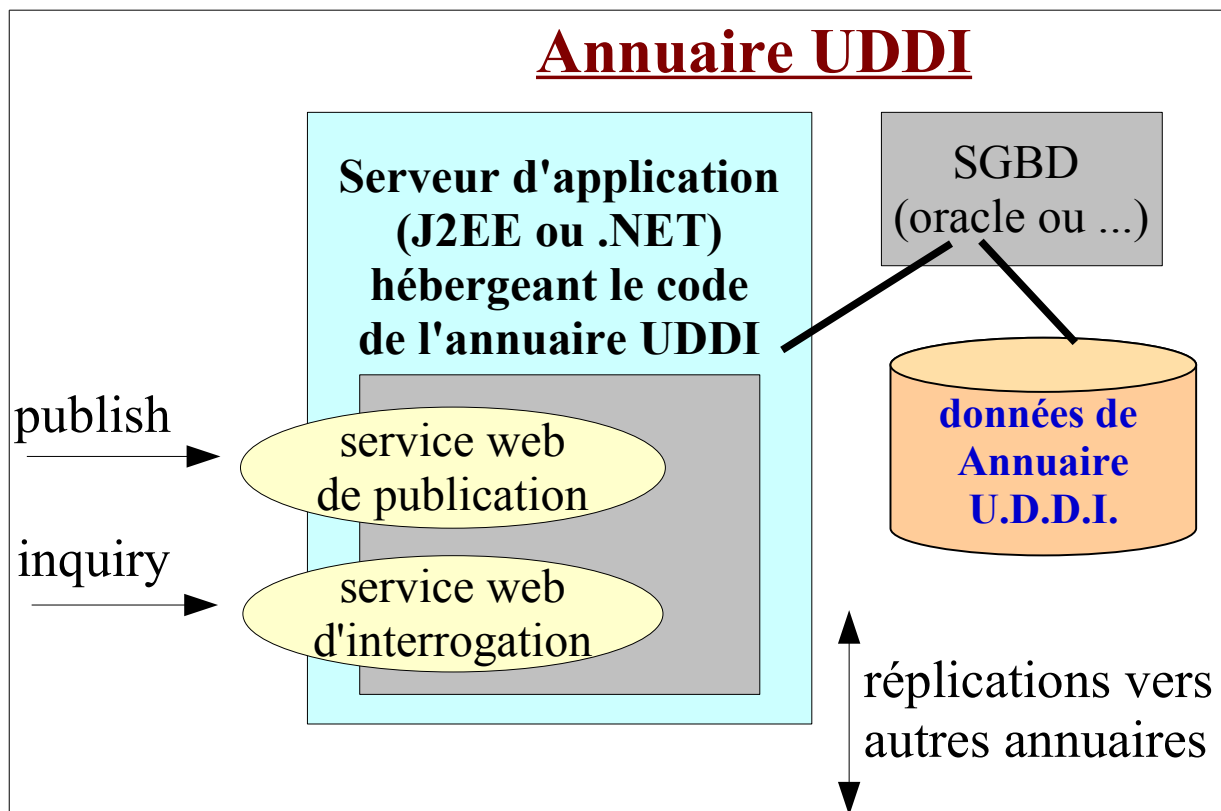
IBM/WebSphere/AppServer/InstallableApps/**uddi.ear** à installer dans **WebSphere 6**

JUDDI (<http://ws.apache.org/juddi>) ==> **juddi.ear** (ou **juddi.war**) = produit open source à installer dans n'importe quel conteneur Web récent (ex: Tomcat 5) ou serveur J2EE (ex: JBoss) .

NB: L'application J2EE correspondant à un annuaire UDDI nécessite généralement la mise en place d'une base de données relationnelle ainsi qu'un pool de connexions JDBC. ==> Lire la documentation de JUDDI ou autre pour connaître la procédure d'installation.

4. Structure d'un annuaire UDDI

4.1. Infrastructure physique d'un annuaire UDDI



Connexion à l'annuaire avec authentification :

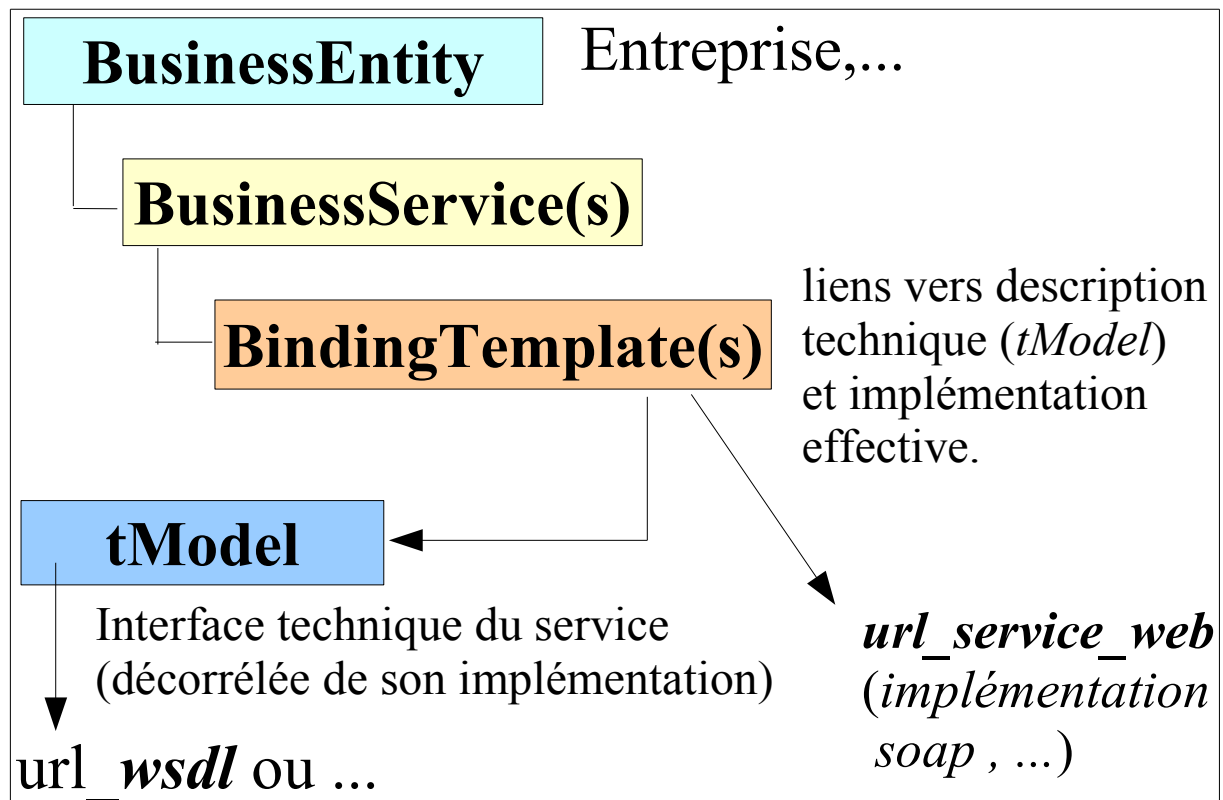
====> connexion UDDI (username , password)

<=== authenticationToken (authTokenString)

====> requêtes_ultérieures_uddi (authTokenString , autresParamètres)

NB: Pour publier des informations dans un annuaire UDDI , il faut préalablement s'inscrire de façon à disposer d'un compte (username,password).

4.2. Structure logique d'un annuaire UDDI



NB:

- La branche "**BusinessEntity** / **BusinessService** / **BindingTemplate**" correspond à des informations qui décrivent le fournisseur du service (sa mise en oeuvre concrète).
- La branche "**tModel**" correspond à une description technologique mais abstraite du service. Un enregistrement "tModel" (et le WSDL associé) est censé décrire une interface d'appel, une éventuelle partie d'une norme B2B ou d'un protocole inter-entreprise.

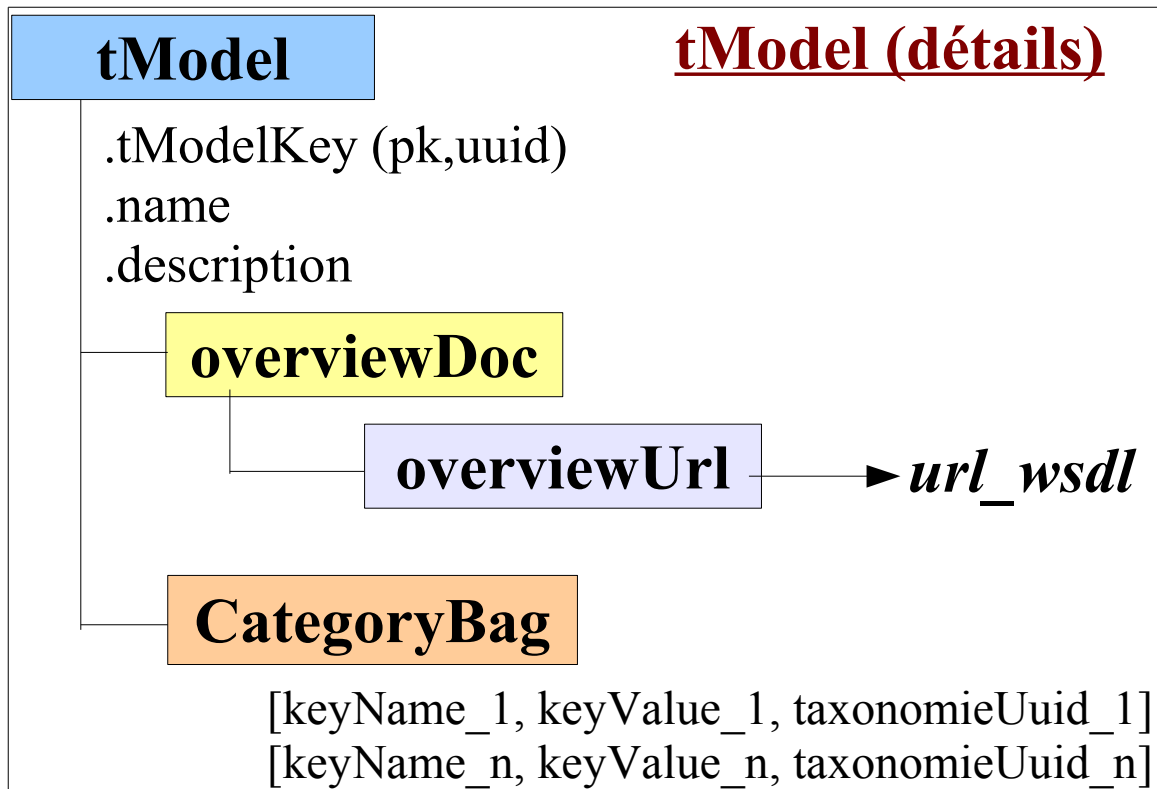
Ces deux branches sont à bien dissocier.

Remarque: le fait qu'un fichier WSDL comporte souvent (mais pas obligatoirement) l'url concrète d'un service WEB apporte quelquefois un peu de confusion.

Remarque générale:

La plupart des données qui sont enregistrées dans un annuaire UDDI (businessEntity, tModel, ...) sont identifiées par des **clefs primaires** au format **uuid** (*universal unique id*) à l'image des *uuid* qui sont utilisés dans la base de registre de Windows (chaîne hexadécimal très longue et unique de par l'algorithme de génération tenant compte de la date, de l'heure, de l'ordinateur, ...).

Il est fondamental de récupérer/retenir ou bien rechercher la valeur d'un *uuid* généré lors d'un enregistrement de façon à ne pas enregistrer plusieurs fois les mêmes informations (lors des mises à jour ultérieures au premier enregistrement).



- Un **tModel** devrait idéalement avoir un **nom** de type URI (ex: *"MonEntreprise_com_ou_org:MyWebServiceInterface.v1"*) et devra pointer sur l'emplacement du fichier WSDL.
- **L'url du fichier WSDL** doit être renseignée comme valeur de la propriété **"overviewUrl"** de la sous partie **"overviewDoc"**.
- Si le tModel fait référence à un fichier wsdl (ce qui est souvent le cas) , on doit normalement faire apparaître l'utilisation de la norme WSDL dans l'une des entrées de CategoryBag :

```
KeyedReference kr = new KeyedReference("uddi-org:types", "wsdlSpec");
kr.setTModelKey("UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4");
krList.add(kr);
```
- Les autres éléments de **"CategoryBag"** correspondent à **éléments de catégorisation** (appelés *"taxonomies"*). Ces taxonomies (qui seront ultérieurement approfondies en fin de chapitre) permettent d'associer des **critères géographiques ou professionnels (ex: branche métier)** qui **seront utiles lors des recherches**.

NB:

- il est important de récupérer le tModelKey (clef primaire de type "UUID:.....") qui a été affecté au tModel lors de son enregistrement. Cet identifiant de tModel devra être ultérieurement renseigné au sein d'un "BindingTemplate".
- Un tModel générique (associé à différentes implémentations d'un service abstrait) peut quelquefois être enregistré par un organisme de normalisation (consortium , ...).

BindingTemplate(s)**BindingTemplate
(détails)**

.description

.accessPoint --> *url (soap) du service***tModelInstanceDetails****tModelInstanceInfo****.tModelKey** --> *tModel dans l'annuaire***instanceDetails****overviewDoc****overviewURL****url_wsdl**

Bien qu'un peu technique (sur le plan de sa composition en sous éléments), un "**BindingTemplate**" peut être simplement considéré comme un **double lien** (un lien externe "**accessPoint**" vers une implémentation concrète de Service WEB – url soap , et un autre lien vers un "**tModel**" référençant lui même un fichier externe WSDL décrivant l'interface abstraite du service).

BusinessEntity & Service (détails)**BusinessEntity**

.businessKey (pk,uuid)

.defaultName , ...

BusinessService(s)

.defaultName

.defaultDescription

BindingTemplate(s)

==> Description de l'entreprise et ses services. Attention au businessKey : pas de doublon

5. taxonomies

Il existe plusieurs sortes de taxonomies (UNSPSC ,) et de nouvelles peuvent éventuellement apparaître. Une entrée de taxonomie comporte les 3 éléments suivants:

- l'**identifiant (au format UUID) de la taxonomie** (lié à un organisme et une version)
- le **nom** de l'entrée = *valeur en clair* (ex: "France" , "Accounting Software" ,)
- la **valeur** de l'entrée = *valeur codée* (ex: "Fr" , "43.23.16.01")

Le nom est pratique pour un affichage et une compréhension humaine.

La valeur est pratique et bien souvent nécessaire pour effectuer une recherche précise.

NB: Certaines taxonomies ont un attribut "Checked" à "true". Ceci signifie que la valeur sera vérifiée .

5.1. UNSPSC

The **United Nations Standard Products and Services Code** is a hierarchical convention that is used to classify all products and services.

Each level in the hierarchy has its own unique number:

XX Segment

[The logical aggregation of families for analytical purposes]

XX Family

[A commonly recognized group of inter-related commodity categories]

XX Class

[A group of commodities sharing common characteristics]

XX Commodity

[A group of substitutable products or services]

XX Business Function (optional)

[The function performed by an organization in support of the commodity]

All UNSPSC entities are further identified with an 8-digit structured numeric code which both indicates its location in the taxonomy and uniquely classifies it. An additional 2-digit suffix indicates the business function identifier.

A structural view of the code set would look as follows:

Hierarchy Category Number and NameSegment:

43 *Information Technology Broadcasting and Telecommunications Communications Devices and Accessories*

Family:

20 *Components for information technology or broadcasting or telecommunications Computer Equipment and Accessories*

Class:

15 *Computers Computer accessories*

Commodity:

01 *Computer switch boxes Docking stations*

Business Function:

14 *Retail*

=====> 43.20.15.01.14

Autre exemple: 43.23.16.01 , Accounting Software

Name: unspsc-org:unspsc

Description: Product and Services Taxonomy: UNSPSC (Version 7.3 remplaçant 3.1)

tModel UUID: uuid:CD153257-086A-4237-B336-6BDCBDCC6634

Categorization: categorization

Checked: Yes

ex:

```
<categoryBag>
  <keyedReference keyName="Domestic Air Cargo Transport"
    keyValue="78.10.15.01.00"
    tModelKey = "uuid:CD153257-086A-4237-B336-6BDCBDCC6634"/>
</categoryBag>
```

5.2. ISO 3166 Geographic Taxonomy

tModels:

Name: uddi-org:iso-ch:3166-1999

Description: ISO 3166-1:1997 and 3166-2:1998. Codes for names of countries and their subdivisions. Part 1: Country codes. Part 2: Country subdivision codes. Update newsletters include ISO 3166-1 V-1 (1998-02-05), V-2 (1999-10-01), ISO 3166-2 I-1 (1998).

tModel UUID: **uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88**

Categorization: categorization

Checked: Yes

<http://www.uddi.org/taxonomies/iso3166-1999-utf8.txt>

exemple : valeur (codée) , nom (en clair)

ex1 (région): **FR-J** , Ile-De-France

ex2 (département): **FR-75** , Paris

ex3 (pays): **FR** , France

```
<categoryBag>
  <!-- Categorize this businessEntity as serving California using the ISO 3166
        taxonomy -->
  <keyedReference keyName="California, USA"
    keyValue="US-CA"
    tModelKey="uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88"/>
</categoryBag>
```

XIX - Annexe – Intalio Bpms Designer (->bpel)

1. Intalio Bpms (BPMN → BPEL)

L'entreprise "Intalio" fut l'une des toutes premières à s'intéresser à BPMN et BPEL. Cette entreprise commercialise entre autre un assez bon produit appelé "Intalio Designer / Bpms" permettant de gérer du code BPEL à partir d'une modélisation BPMN .

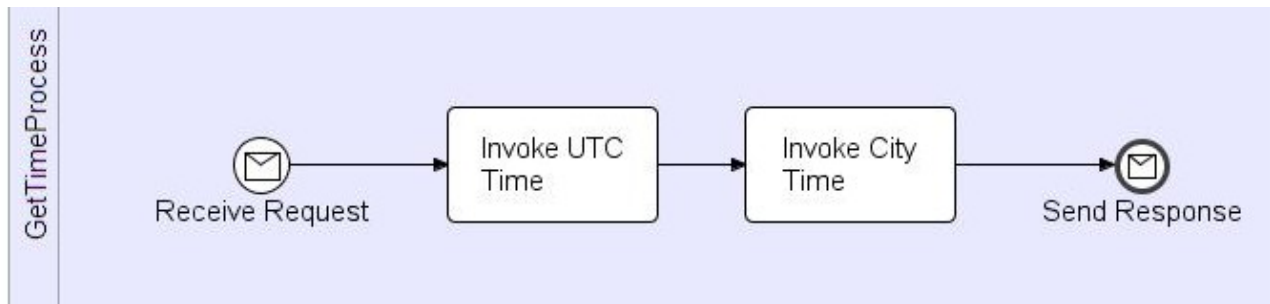
NB : Intalio BPMs Designer existe en plusieurs versions (CE/gratuite/basique et payante/élaborée) .

1.1. Projet "Intalio Business Process Project" et structure

Au sein d'un projet "Intalio Business Process Project" , les fichiers à éditer ne doivent pas être placés dans le répertoire "**build**" . Le contenu du répertoire "**build**" (BPEL , WSDL, ...) sera régulièrement généré automatiquement .

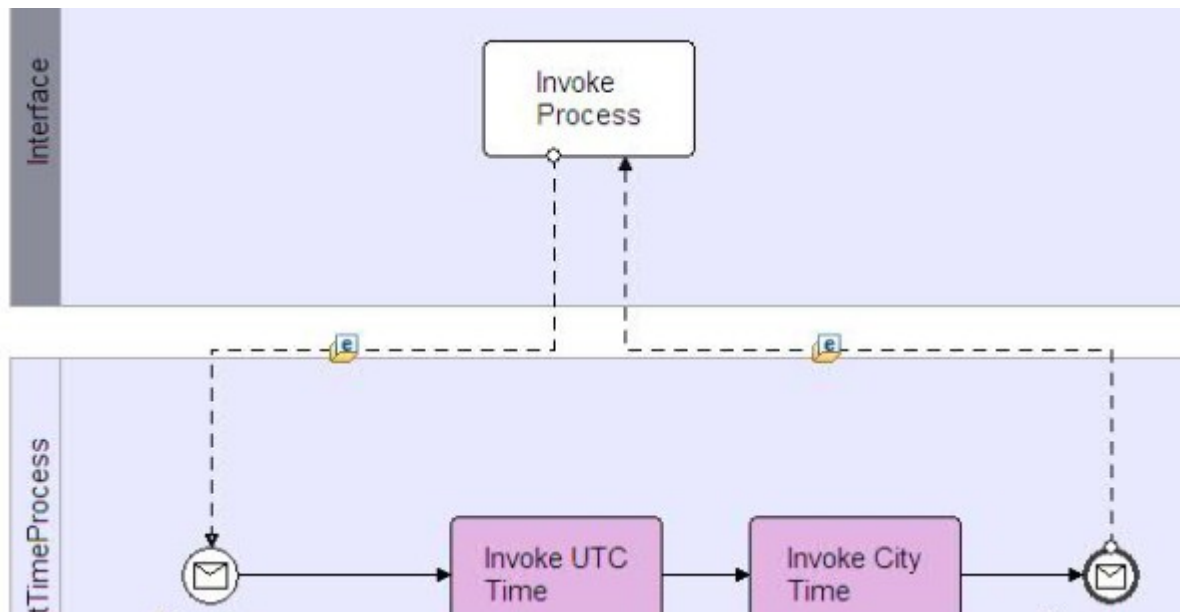
1.2. Processus métier BPMN

- renommer éventuellement le "pool" principal en "process" ou " ..."
- déposer les événements et tâches du processus ("Start Message" , "End Message" , "Task") nommer ces éléments avec des noms simples, courts et parlants .
- relier via des flèches pleines



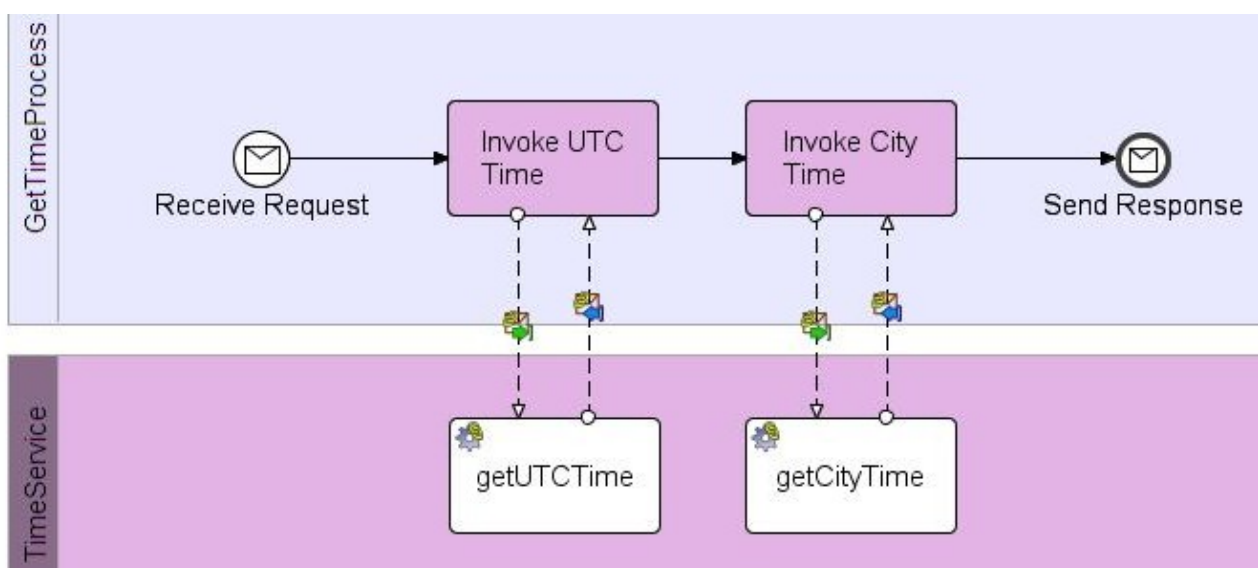
1.3. Interface du processus (avec structure XSD)

- Déposer (plutôt en haut) un second "Pool" que l'on peut nommer interface . Ceci représentera le format des appels effectués par les futurs clients (soap-ui , ...)
- Rendre ce pool "non exécutable"
- Y déposer une tâche "invoke process"
- Relier cette tâche vers l'entrée et la sortie du processus (du pool plus bas)
- créer un nouveau fichier "...xsd" (nouveau XML Schema)
- y placer des "complexType" et "element" pour structurer les formats des requêtes et des réponses (ex : *Axb* (*a:double* , *x:double*, *b:double*) , *AxbResponse* (*return:double*) , *axb* , *axbResponse*) .
- Développer la structure du fichier XSD dans l'arborescence (project explorer).
- Effectuer un "drag & drop" des éléments XSD vers les flèches en pointillées qui relient "invoke process" avec les événements "start" et "end" . Valider "add" . Un icône "e " devrait normalement apparaître .



1.4. Partenaire à invoquer (avec WSDL)

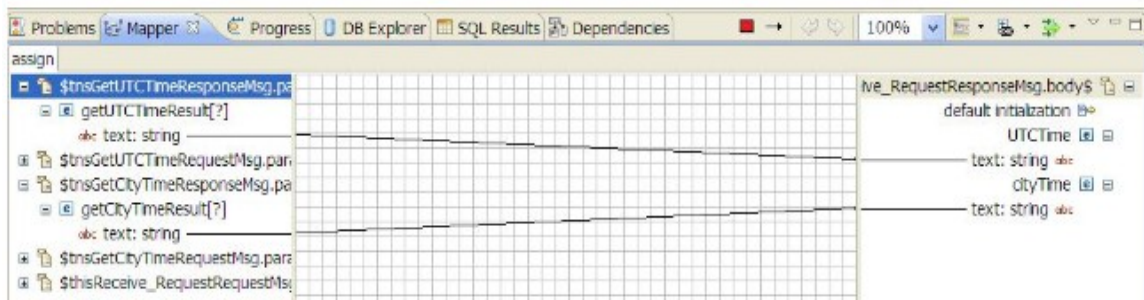
- Déposer (plutôt en bas) un troisième "Pool" avec un nom symbolisant un partenaire (service à invoquer). Rendre ce pool "non exécutable"
- Créer un fichier vide "xyz.wsdl" et y recopier le contenu de la description WSDL d'un service à invoquer depuis le processus . [Navigateur avec URL en "...?wsdl" , afficher code source , tout sélectionner , copier/coller]
- Développer la structure du fichier WSDL dans l'arborescence .
- Repérer les parties "service ... / xxxx / operation _zzz" (avec icône engrenage) mais ne pas choisir en développant l'interface/portType.
- Effectuer des "drag & drop" d'une opération vers le "pool" bpmn correspondant au service à invoquer.
- Choisir "provide (single task) / répéter ceci pour chaque méthode à invoquer
- Relier les "Task" des deux "pools" via des flèches pointillées (avec des ronds au début)
- Intalio placera alors automatiquement des icônes représentant des requêtes et réponses :



1.5. Paramétrage des assign-bpel via le "mapper" d'intalio bpmn

- Sélectionner une tâche "**invoke xxx**" ou "**reply yyy**" au sein du digramme BPMN
- Dans l'onglet "**mapper**" de "intalio bpmns designer" , développer les variables nécessaires aux "copy (from/to)" selon l'algorithme propre au processus.
- Tirer graphiquement des flèches entre une partie d'une variable source (à gauche) vers une partie d'une variable destination (à droite).
- La partie au milieu de l'assistant "mapper" permet d'exprimer d'éventuelles expressions intermédiaires de type "concat" , "substring" , "div" , "*" , "+" , "-" , ...

NB : pas besoin de préciser explicitement une partie "initialisation de variable BPEL" , ce sera fait automatiquement .



1.6. Génération du descripteur de déploiement

La modélisation BPMN précédente (accompagnée des fichiers XSD et WSDL) suffit à "Intalio Bpms designer" pour générer du code BPEL exécutable .

Le code exécutable sera généré pour le "pool" (du milieu) marqué comme exécutable .

Les autres "pools" (non exécutables) servent à préciser les structures de données échangées.

Par défaut, le code BPEL (et certains fichiers annexes WSDL) sont générés au sein du répertoire "build" au fur et à mesure de l'édition du modèle BPMN .

En cas d'erreur , d'oubli ou d'incohérence , un message apparaît explicitement en rouge au sein du diagramme.

- S'il s'agit d'un oubli (ou bien d'une partie pas encore complètement modélisée) , continuer d'ajouter des éléments jusqu'à ce que le message rouge disparaisse .
- S'il s'agit d'une fausse manipulation ou bien d'une chose interdite par BPMN/Intalio il faut alors supprimer (via clic droit / delete) l'élément en erreur et le reconstruire mieux.
Attention : le "Undo" ne fonctionne pas bien .

Lorsque toute la modélisation est terminée (avec XSD/WSDL) et qu'aucune erreur apparaît, on peut tenter un déploiement pour vérifier si le processus s'exécute sans erreur .

Pour générer le fichier deploy.xml au sein du répertoire build , il faut sélectionner le projet et déclencher "le déploiement" et l' "éventuelle exécution au sein du serveur de test intalio-bpel" .

Ceci s'effectue en cliquant sur l'icône engrenage de la barre d'outils  puis sur le bouton



de l'écran de paramétrage .

Le code BPEL+WSDL généré est assez complexe (nombreux namespaces et import xml) et s'exécute bien dans **ODE** que si l'on prend soin de **changer l'URL !!!!**

Le serveur de test "intalio Bpel Server" prévu par Intalio Designer est un petit peu différent du standard libre ODE .

De façon à ce que le code généré s'exécute bien (sans erreur de chemin relatif) au sein du serveur ODE imbriqué dans tomcat , il faut effectuer (ou ré-effectuer) les petits ajustements suivants dans le répertoire build :

- vérifier la présence de deploy.xml (et lancer "deploy" sinon)
- supprimer le fichier "xxxx-tempo.zip" inutile
- éditer (dans build ou dans un répertoire de copie) le fichier xxx-yyy.wsdl et changer l'URL en la simplifiant de la manière suivante :

```
<soap:address  
location="http://localhost:8080/ode/processes/xxx/yyy/zzz/interface"/>
```

à transformer en :

```
<soap:address location="http://localhost:8080/ode/processes/xxx"/>
```

Le déploiement dans "tomcat+ode" s'effectue alors en :

- créant un nouveau répertoire "xxx" dans TOMCAT_HOME/webapps/ode/WEB-INF/processes .
- recopiant tous les fichiers utiles de build vers ode/WEB-INF/processes/xxx
- vérifiant le déploiement via la console ODE (<http://localhost:8080/ode>)
- vérifiant l'accès au WSDL du processus (<http://localhost:8080/ode/processes/xxx?wsdl>)
- effectuant un test avec soap-ui

NB :

- ODE doit avoir été préalablement installé dans tomcat en déposant ode.war dans TOMCAT_HOME/webapps et en (re-)démarrant tomcat .
- Lorsque l'on redéploie une nouvelle version du processus BPEL dans ODE , il est conseillé de préalablement supprimer l'ancienne version en déclenchant un "**undeploy**" depuis la console ODE .