

SOA

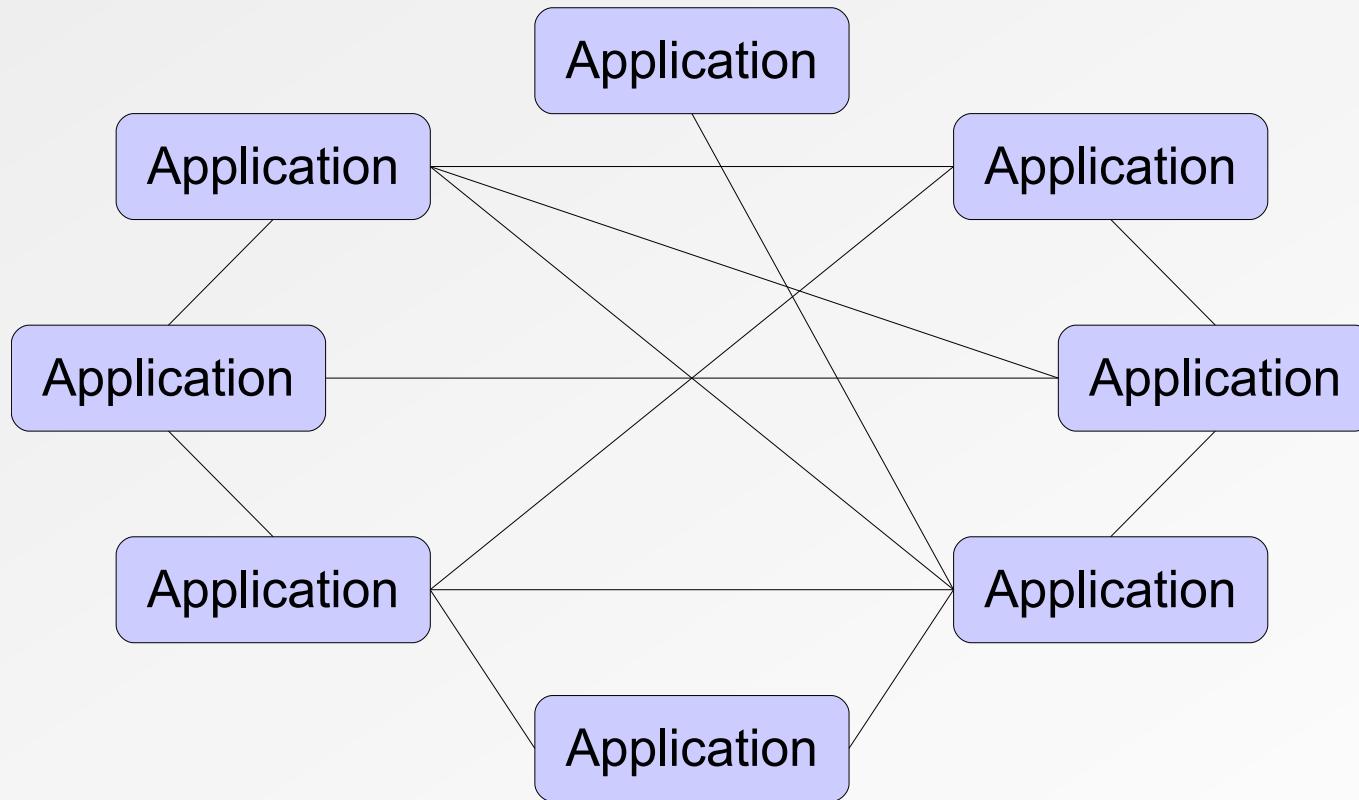
Architecture et intégration d'applications d'entreprise

- Les problématiques d'intégration
- La SOA – Services Oriented Architecture
- La problématique universelle – Echanger des messages
- Les données – XML, XSD, XSLT, XPATH
- Les Web Services – SOAP, REST
- Les MOM et les files de messages
- Les ESB – Bus de routage et d'intégration
- Les moteurs BPEL – Orchestrateur de services métiers

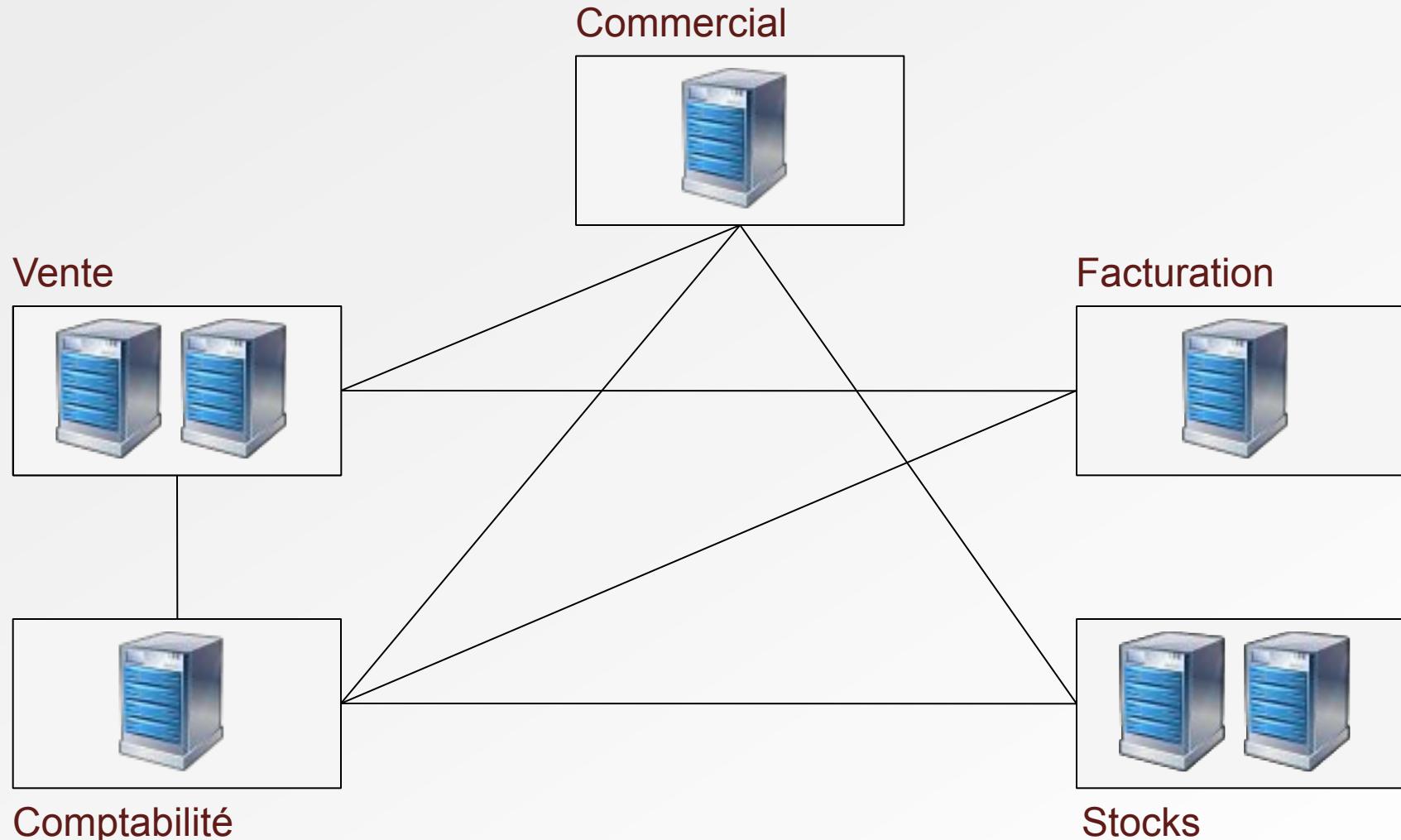
Les problématiques d'intégration

- L'architecture des systèmes d'information
- Un monde complexe et hétérogène
- Synchronisme et asynchronisme
- Le couplage faible / couplage fort

Un système d'information



La problématique d'intégration

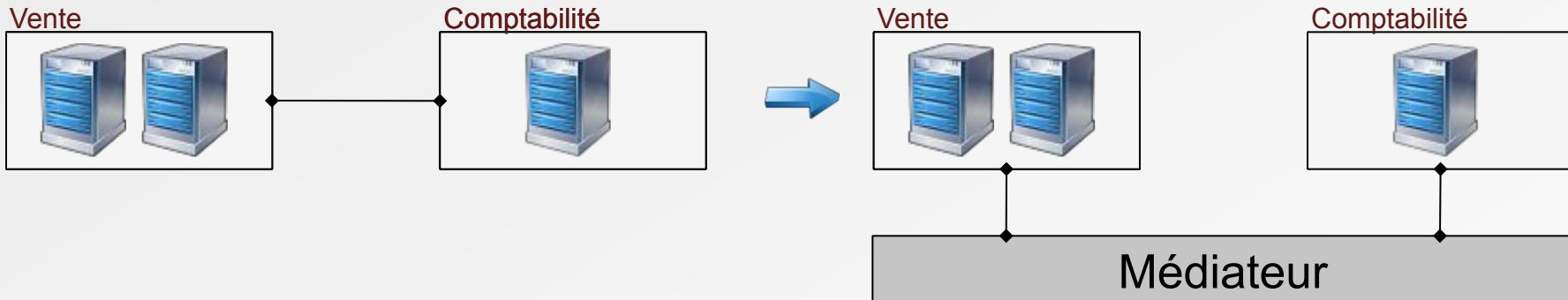


- Les réseaux ne sont pas fiables
 - Système distribué : contraintes fortes
- Les réseaux sont lents
 - Système distribué ≈ Système asynchrone
- Les applications sont différentes à tous les niveaux
 - Système d'exploitation
 - Implémentation
 - Format de données

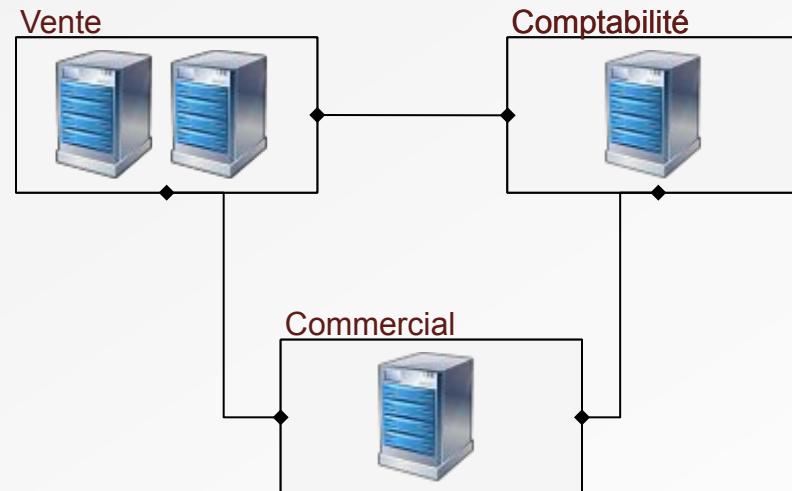
- Le changement est inévitable
 - Changement métier (stratégie, règlement...)
 - Changement technique
 - Changement humain
 - Changement de « mode » (exemple : EJB)

Intégration – Les fonctions de base

- Médiation – Réduction des adhérences entre applications



- Connectivité – support de multiples protocoles de transport

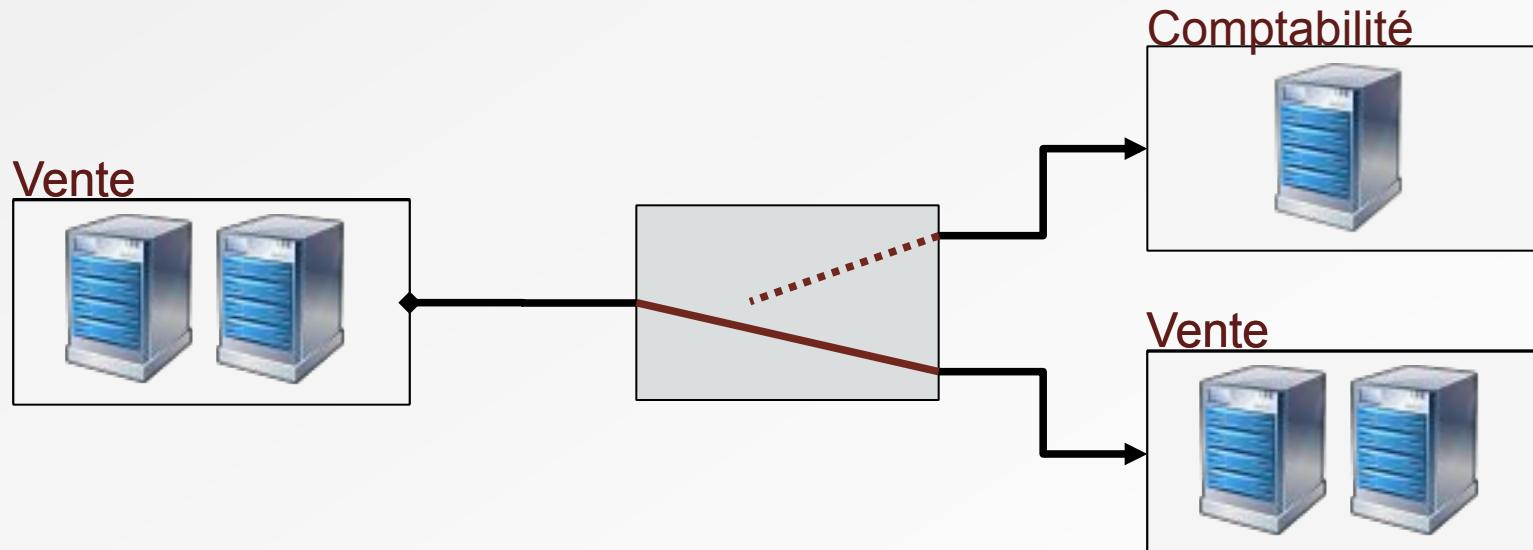


Intégration – les fonctions de base

- Transformation – Support de différents types de données



- Routage – Routage des messages entre applications



- Exemples : FTP, CFT, Copie locale, etc.
- Avantages
 - Découplage des applications
 - Simple à mettre en place
 - Pas besoin d'outils d'intégration
- Inconvénients
 - Format des données
 - Temps de latence → biais de synchronisation
 - Gestion des fichiers complexes

- Exemples : Oracle, PostgreSQL, etc.
- Avantages
 - SQL → format des données unifié
 - Schéma partagé → pas de conflit sémantique
- Inconvénients
 - Couplage fort au schéma → forte sensibilité au changement
 - Logiciel : impossibilité de changement du schéma

- Exemples : CORBA, RMI, etc.
- Avantages
 - Risque de problème sémantique faible
 - Méthode « naturelle » : intégration facilitée
- Inconvénients
 - Méthode « naturelle » : contraintes comparée à un appel local (performances, robustesse)
 - Couplage applicatif fort

Style d'intégration – Messages

- Exemples : MOM, SMTP, Web Services, etc.
- Avantages
 - Découplage des applications
 - Robuste
 - Performant
- Inconvénients
 - Légère latence
 - Adapté aux « petits » paquets de données

- Liaisons entre applications
 - Directes
 - Non standardisées

Avantages

- Facile et rapide à mettre en place (⚠ dans un premier temps)
- Ne nécessite pas d'installer une solution tierce

Inconvénients

- Forte adhérence entre applications
- Complexité maintenance / supervision

- MOM – Message-Oriented Middleware
- JMS – API Java supportée par une grande majorité des éditeurs de MOM, devient un standard de fait
- Exemples : Apache ActiveMQ, IBM WebSphere MQ

Avantages

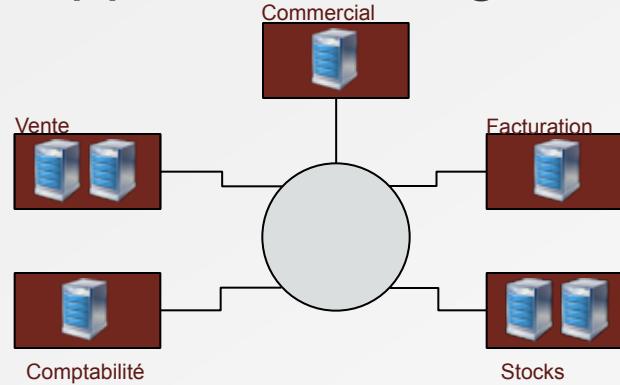
- Solution simple

Inconvénients

- API hétérogènes
- Ne couvrent pas l'ensemble des problématiques d'intégration (exemple : connectivité, routage, etc.)

Histoire de l'intégration – Les EAI

- EAI – Enterprise Application Integration



Avantages

- Solution complète d'intégration

Inconvénients

- Propriétaire
- Prix
- Complexé
- Centralisé → SPOF : « Single Point Of Failure »

Les Enterprise Service Bus

- Besoin d'une solution d'intégration
 - Distribuée
 - S'appuyant sur les standards
- Opposition frontale à la vision EAI (centralisée et propriétaire)
- Technologie s'appuyant sur les bonnes pratiques d'intégration et sur l'écosystème existant
 - MOM
 - Patterns d'intégration : routage / validation / transformation
- L'ESB se compose de deux éléments principaux
 - Le bus de messages
 - Les conteneurs de services

Service Oriented Architecture

- Les grands principes
- Démystifier la SOA
- Pourquoi, comment et pour qui ?

SOA – Pourquoi ?

- L'entreprise doit s'adapter continuellement
- L'entreprise est de plus en plus dépendante des technologies
→ le SI peut devenir un frein
- « *C'est pas notre faute, c'est un problème informatique...* »
- Acteurs métiers  Acteurs techniques
→ **Le SI doit être le moteur du business**

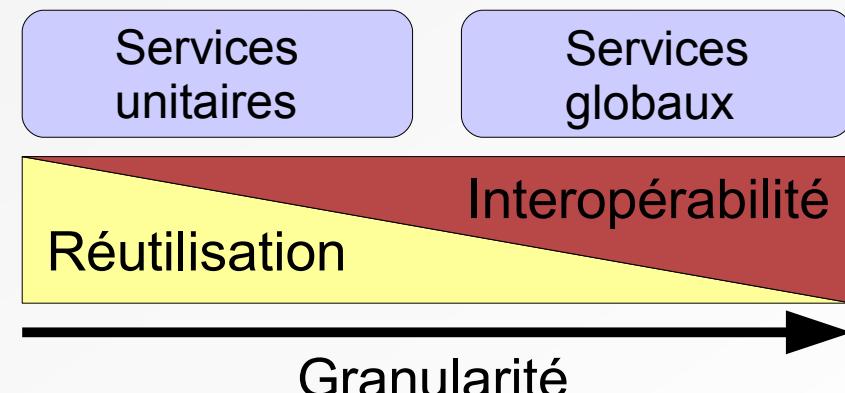
La SOA, qu'est ce c'est ?

- « désigne un type d'**architecture** fournissant un ensemble de **services localisables**, à des applications clientes ou à d'autres services **distribués** sur un réseau, via des **interfaces** publiées »
- SOA = « IOA » (Interface Oriented Architecture)
 - Abstraction
 - Couplage lâche (couplage faible ou léger)
- Couplage lâche
 - Les systèmes qui entrent en interaction ne connaissent pas intimement les détails d'implémentation des uns et des autres

- Simplicité
 - Efficacité
 - Communication métier/technique
- Flexibilité et maintenabilité
 - Pérennité des systèmes
- Réutilisabilité
 - Réduction des anomalies
 - Gain de productivité
- Indépendance vis à vis des technologies
 - S'appuyer sur les standards et les bonnes pratiques

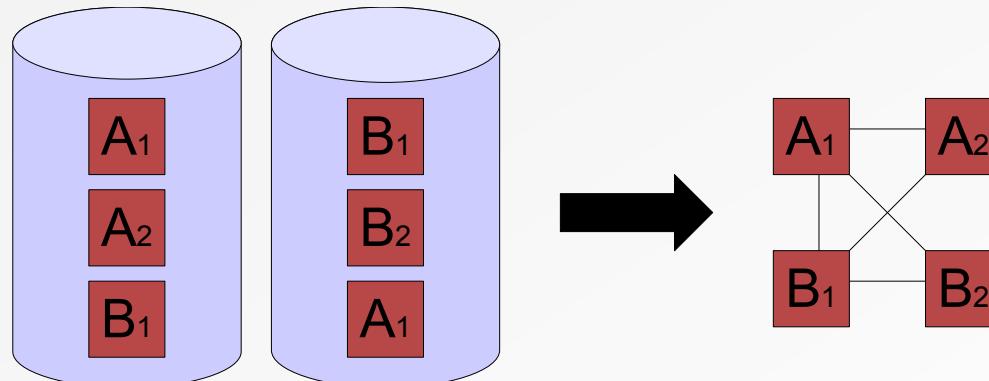
- Service = élément central de la démarche SOA
- Service = Contrat
 - Fonctionnalités métier
 - Interfaces
 - Données
 - Performances, robustesse
- Intégration via des services
 - Les systèmes communiquent via des protocoles et des formats standards
 - La vision « service » remplace la traditionnelle vision « silo »
 - Intégration de l'existant via des services

- Le Service
 - Assure et respecte la mise en œuvre d'un contrat
 - Autonomie
 - Peut être invoqué quelle que soit sa localisation
- Propriétés intrinsèques d'un service
 - Ré-utilisabilité
 - Interopérabilité



SOA – Évolution des SI

- Intégration des systèmes via des Services
 - Les systèmes communiquent via des protocoles et des formats standards
 - La vision *service* remplace la vision *silo*
 - Le SI devient distribué
 - L'intégration de l'existant via des services



The Spaghettis Incident ?

- Une architecture orientée services peut vite très rapidement devenir comme cela



- SOA est autant une démarche métier qu'une démarche technique
 - Les principes de mise en œuvre d'une architecture SOA
 - Approche « service » : conteneurs de services
 - Simplicité, Standard, Réutilisation, Maintenabilité
 - Solutions ESB
 - Solutions BPEL
 - Solutions 100% Web Services (WSOA)
 - Solutions 100% Messaging (JMS, AMQP)
- combinatoire de ces solutions

Principes d'une architecture SOA

- **Contrat Standard** - Assurer l'interopérabilité
- **Couplage lâche** - Minimiser les dépendances entre consommateur et producteur
- **Abstraction** – S'abstraire de l'implémentation
- **Réutilisabilité** – Gagner en productivité
- **Autonomie** – Minimiser les dépendances du service vis à vis de son environnement (services, systèmes existants...)
- **Sans état** - Simplifier la concurrence, la réutilisabilité, la maintenance, les performances ...
- **Localisable** – Trouver le service répondant à un besoin
- **Composable** – Créer de nouveaux services à partir des services existants

- Style d'architecture basé sur
 - un découpage intelligent
 - le respect des contrats d'interface
 - des implémentations simples et efficaces
 - des protocoles libres et ouverts
 - des standards

Keep it simple !

Echanger des messages entre les services

- Les Web Services
- Les MOM – Messages Oriented Middleware
- Les ESB – Enterprise Service Bus
- BPEL – Business Process Execution Language

Web Services

Les Web Services : c'est quoi ?

- Un Web Service est un module logiciel autonome
- Un Web Service fournit une description de ses capacités
- Un Web Service est accessible depuis un réseau donné
 - Intranet
 - Internet
- Un Web Service communique via des protocoles standards de communication (HTTP essentiellement)
- Un Web Service envoie et reçoit des données formatées en XML

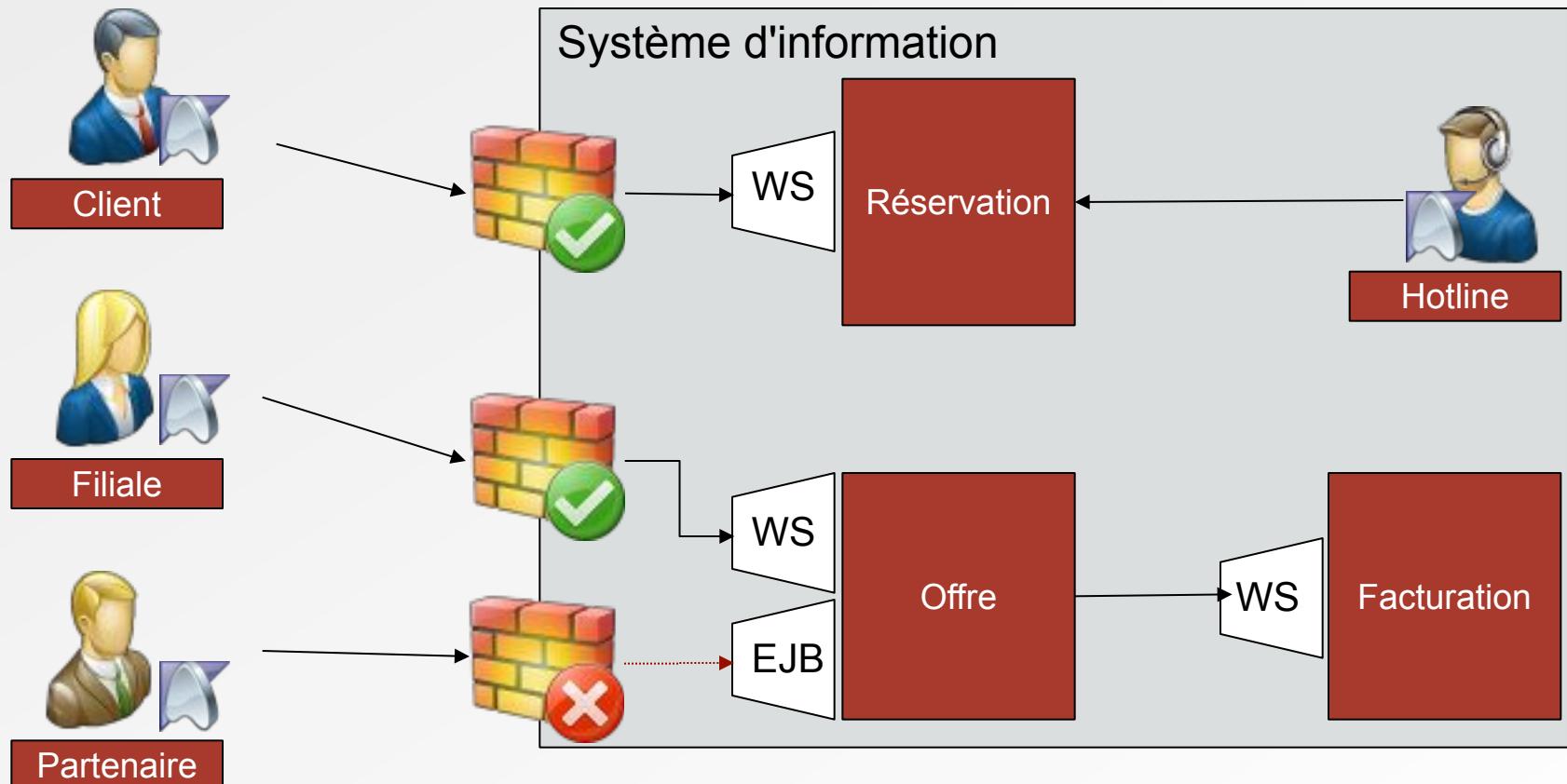
Dans « Web Service », il y a Web !

- World Wide Web ≈ HTTP
- Les Web Services sont accessibles le plus souvent via HTTP
 - Standard
 - Interopérable
 - Simple
 - Universel
 - *Passe-partout* (ie. aucun problème de Firewall)
- Avec le temps, les Web Services ont connu de nouveaux supports
 - SMTP, FTP
 - JMS
 - ...

Histoire de l'informatique distribuée

- L'essor des technologies objet a permis de s'affranchir des problématiques techniques/matérielles liées aux systèmes distribués
- CORBA : Common Object Request Broker Architecture
- Monde Java : RMI
- Monde Microsoft : .NET Remoting
- ...
- Web Services : pourquoi s'appuyer sur une/des nouvelle(s) technologie(s) ? → Le réseau n'est pas homogène

Les Web Services dans l'Entreprise



- SOA est autant une démarche métier qu'une démarche technique
- Les principes de mise en œuvre d'un Web Service sont proches de ceux d'une architecture SOA
 - Approche « Service »
 - Simplicité, Standard, Réutilisation, Maintenabilité
- Web Service n'est pas SOA mais c'est un outil idéal pour la mettre en place
- Les Web Services sont la technologie la plus simple et la plus courante pour mettre en place une architecture orientée services

MOM

Les MOM – Message Oriented Middleware

- Les MOM transportent des messages
- Les communications avec les MOM sont obligatoirement asynchrone
 - Messaging 100% asynchrone
 - Pas de question / réponse nativement
 - Système de « fire and forget »
 - Uniquement acknowledgement technique ou transactionnel

Propriétés des MOM

- Propriétés intrinsèques aux MOM
 - Communication asynchrone
 - Pas de routage natif
 - Pas de transformation de données (autre que l'encoding)
 - Persistance des messages marqués « durable »
 - Fiabilité

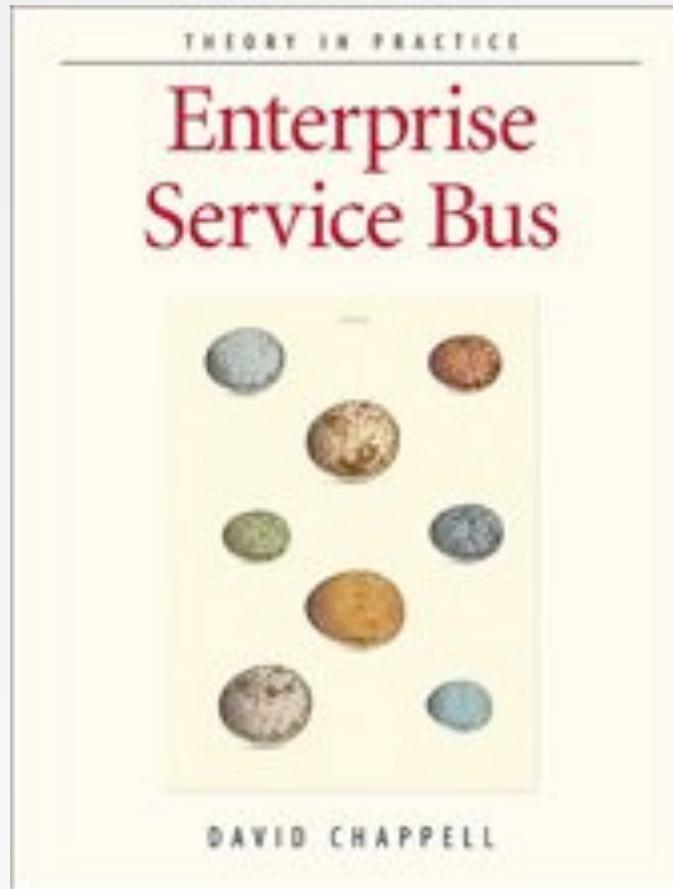
Utilisation des MOM

- De nos jours, les MOM sont partout
- Ils représentent la plupart « des tuyaux » des systèmes d'information
- 2 modes principaux d'utilisation :
 - « direct » : un service produit un message dédié à un autre service
 - « publish / subscribe » : un service produit un message qui peut être récupéré par n'importe quel autre service abonné au canal

ESB

ESB – Enterprise Service Bus

- 2004 : Naissance de l'ESB - David Chappell

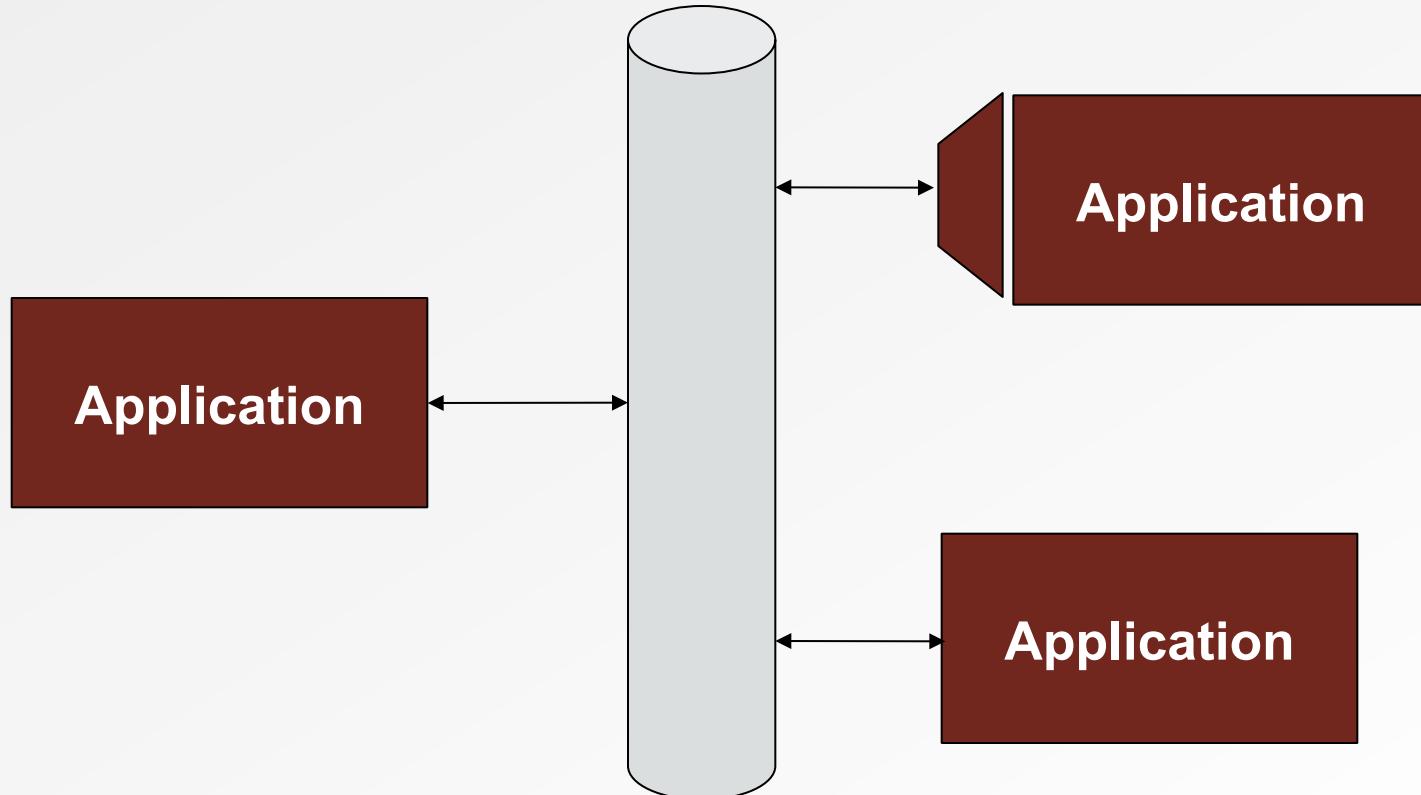


Enterprise Service Bus par D. Chappell

- Besoin d'une solution d'intégration
 - Distribuée
 - S'appuyant sur les standards
- Opposition frontale à la vision EAI (centralisée et propriétaire)
- Technologie s'appuyant sur les bonnes pratiques d'intégration et sur l'écosystème existant
 - MOM
 - Patterns d'intégration : routage / validation / transformation
- L'ESB se compose de deux éléments principaux
 - Le bus de messages
 - Les conteneurs de services

Le bus de messages

- Pattern d'intégration définissant une architecture permettant
 - De faire travailler ensemble des applications distantes
 - De pouvoir ajouter/supprimer des applications sans affecter les autres



Le bus de message en détail

Un bus de message est la combinaison de :

- D'une infrastructure de communication commune
 - Exemple : MOM
 - S'abstraire de l'hétérogénéité du Système d'Information
- D'un ensemble de commandes de gestion
 - Exemples : écrire des données à telle adresse
 - Ensemble des possibilités offertes par le bus
- D'un modèle de données communs
 - Exemple : structure XML (schéma XSD)
 - Langage commun permettant d'assurer la communication entre applications hétérogènes

Les conteneurs de services

- Alors que l'ensemble des services (routage, transformation, connectivité) sont centralisés au sein d'un EAI, ils sont distribués au sein d'un ESB
- Les conteneurs de services fournissent l'ensemble des services d'un ESB
- Un conteneur de services peut être comparé à un serveur d'application léger
 - Facilement et rapidement installable
 - Peu gourmand
- Ils communiquent entre eux en utilisant le bus de messages comme médiateur

Un ESB est distribué par définition

- Un ESB n'est pas centralisé
 - « *Ici avec le portail, mais comme avec l'ensemble des démarches d'intégration (SOA, MDM, ESB), il faut se poser la question : mais pourquoi vouloir tout centraliser ?* » - ***Lu sur un blog***
- Difficile d'imaginer ou de vendre un(e) produit/technologie distribué(e)
- Les ESBs pâtissent de la mauvaise réputation des EAI
 - Propriétaire
 - Lourd
 - Coûteux
- David Chappell propose notamment de s'appuyer sur les MOM et notamment sur JMS pour gérer la distributivité

BPEL

Le langage BPEL

- Business Process Execution Language
- Langage
 - Description de business process
 - Interactions avec des Web Services
 - Écrit en XML
- Standard issu du consortium OASIS
 - WSBPEL → version 2.0 avril 2007
 - Différentes implémentations (Oracle, Apache, etc.)



- Business Process (processus métier)
 - Répond à un besoin d'entreprise
 - S'adapte à un modèle de données métier
 - Réalise un ensemble d'actions et de tâches
 - Est un service
- Exemple de Business Process
 - Réservation de billets d'avion
 - Format d'entrée imposé
 - Vérifier disponibilité, appeler facturation, notifier client

BPEL – Quelle utilité ?

- Processus BPEL implémente un Business Process
- Processus BPEL expose un Web Service
 - Business Process exposé via Web Service (en BPEL)
- Orchestrateur de Web Services
- Abstraction
- Un processus BPEL orchestre l'appel à d'autres WS
 - Processus BPEL
 - Services externes
- Granularité inconnue du service appelé

Un processus BPEL

- Processus BPEL
 - WSDL
 - *Types*
 - *Messages*
 - *Opérations / PortTypes*
 - BPEL
 - *PartnerLinks*
 - *Variables*
 - *Activités*

Un processus BPEL

Processus BPEL

Partner links

Variables

Activités

```
<bpel:process name="HelloBPEL" targetNamespace="http://bpel.resanet.com/hellobpel"  
    suppressJoinFailure="yes" xmlns:tns="http://bpel.resanet.com/hellobpel"  
    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">  
  
    <bpel:partnerLinks>  
        <bpel:partnerLink name="client" partnerLinkType="tns:HelloBPEL"  
            myRole="HelloBPELProvider" />  
    </bpel:partnerLinks>  
  
    <bpel:variables>  
        <bpel:variable name="input" messageType="tns:HelloBPELRequestMessage"/>  
        <bpel:variable name="output" messageType="tns:HelloBPELResponseMessage"/>  
    </bpel:variables>  
  
    <bpel:sequence name="main">  
        <bpel:receive name="receiveInput" partnerLink="client" portType="tns:HelloBPEL"  
            operation="process" variable="input" createInstance="yes"/>  
        <bpel:assign name="assign">  
            <bpel:copy>  
                <bpel:from part="payload" variable="input">  
                    <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[tns:input]]></bpel:query>  
                </bpel:from>  
                <bpel:to part="payload" variable="output">  
                    <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"> <![CDATA[tns:result]]></bpel:query>  
                </bpel:to>  
            </bpel:copy>  
        </bpel:assign>  
        <bpel:reply name="replyOutput" partnerLink="client" portType="tns:HelloBPEL"  
            operation="process" variable="output" />  
    </bpel:sequence>  
</bpel:process>
```

XML

Extensible Markup Language

- XML et XSD : langage universel et standard
- Les namespaces et les types
- Le langage et les requêtes Xpath
- Les transformations XLST

L'origine du XML

- eXtensible Markup Language
- Langage à balise basé sur SGML (Standard Generalized Markup Language)
- Normé depuis 1998 par le W3C
- Langage permettant de séparer la sémantique de l'information même
- Extensible grâce au système de namespaces et de création de grammaire

Format XML

- Déclaration XML
- Blocs délimités par des balises
 - Ouvrante et fermante
 - Balise simple
- Attributs
 - Disponibles dans les balises ouvrantes et simples
 - Associent une clé et une valeur pour l'élément courant

```
<tag attribute="value">  
...  
</tag>
```

- Commentaires

```
<tag />  
<!-- Comment in the XML code -->
```

- Caractères spéciaux

```
<tag>Text with special characters "&acute; &grave; &amp;"</tag>
```

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

```
<complex-tag>  
  <simple-tag />  
</complex-tag>
```

- Langage orienté sur la séparation de contenu et de valeur sémantique
- Nécessité d'imposer des normes strictes
- Normes définies par le W3C
- Tout document XML respecte une syntaxe commune
- Possibilité de valider le XML en général
 - <http://validator.w3.org/>
 - http://www.w3schools.com/xml/xml_validator.asp
- A terme l'intérêt est de spécifier encore plus la syntaxe pour obtenir un document clair et concis

Utilité du langage

- Langage servant de pont entre humains et machines
 - Structuré
 - Lisible
 - Interopérable (basé sur du texte ascii et non exécuté)
- Selon les besoins peut être parsé ou écrit facilement manuellement ou automatiquement
- Possibilité de manipulation, requêtes et transformation de contenu
 - Transformation en XML selon le contenu
 - Transformation en binaire
 - Requêtes par des langages tel que XQuery/XPath ou XQL

Document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<formation>
    <titre>Web Services</titre>
    <durée>3</durée>
    <!-- liste des publics -->
    <publics>
        <public>Développeur</public>
        <public>Architecte</public>
    </publics>
    <!-- liste des sessions -->
    <sessions>
        <session du="17/04/10" au="19/04/10"/>
        <session du="24/09/10" au="26/09/10"/>
    </sessions>
</formation>
```

Déclaration

Élément racine

Élément

Commentaire

Attribut

Un document XML bien formé (1/2)

- L'élément XML peut être représenté par
 - Une balise ouvrante (ie. <balise>) et une balise fermante (ie. </balise>)

```
<titre>Web Services</titre>
```

- Une balise vide (ie. <balise/>)

```
<session du="17/04/10" au="19/04/10"/>
```

- La structure balise ouvrante/balise fermante peut encadrer
 - Du texte
 - D'autres éléments
 - Un mélange de texte et d'éléments

Un document XML bien formé (2/2)

- Un balise ouvrante ou vide peut contenir des attributs

```
<session du="17/04/10" au="19/04/10"/>
```

- Les balises doivent être correctement imbriquées

```
<Italique><Gras>Document mal formé</Italique></Gras>
```

```
<Italique><Gras>Document bien formé</Gras></Italique>
```

- Remarque : XML est sensible à la casse

- XSD = XML Schema Definition
- Un schéma XML a pour objectif de définir la structure d'un document XML
- XSD est le successeur de DTD
- Un schéma XML définit
 - Les éléments et attributs pouvant apparaître
 - La hiérarchie et l'ordre des éléments du document
 - Si un élément peut/doit contenir du texte
 - Les types des éléments (ex : chaîne, entier ...)
 - ...

Exemple XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://www.zenika.com/disques"
         xmlns:tns="http://www.zenika.com/disques"
         elementFormDefault="qualified">

    <element name="disque" type="tns:Disque"/>

    <complexType name="Disque">
        <sequence>
            <element name="titre" type="string"/>
            <element name="auteur" type="string"/>
        </sequence>
    </complexType>

</schema>
```

Namespace

Les namespaces (1/3)

- Namespace = « espace de nommage »
- Un namespace a pour objectif d'assurer l'unicité d'un élément XML

```
<disque>
  <titre>Beautiful Freak</titre>
  <auteur>Eels</auteur>
</disque>
```



```
<disque ref="CD_0001">
  <titre>Road To Ruin</titre>
  <artiste>Ramones</artiste>
  <annee>1979</annee>
</disque>
```

- La convention est d'utiliser une URI pour définir un namespace
Exemple : <http://www.zenika.com>

Les namespaces (2/3)

- Déclaration d'un namespace ► **xmlns:prefix="namespace"**

```
<ns:disque xmlns:ns="http://www.zenika.com/disques">
```

- Association d'un élément à un namespace

```
<ns:disque xmlns:ns="http://www.zenika.com/disques">
  <ns:titre>Beautiful Freak</ns:titre>
  <ns:auteur>Eels</ns:auteur>
</ns:disque>
```

- Possibilité de déclarer plusieurs namespaces

```
<cd:disque ref="CD_0001"
  xmlns:cd="http://www.cdiscues.fr"
  xmlns:txt="http://www.cd-text.fr">
  <cd:titre>Road To Ruin</cd:titre>
  <txt:artiste>Ramones</txt:artiste>
  <txt:annee>1979</txt:annee>
</cd:disque>
```

Les namespaces (3/3)

- Définition d'un namespace par défaut ► **xmlns="namespace"**

```
<disque xmlns="http://www.zenika.com/disques">
    <titre>Beautiful Freak</titre>
    <auteur>Eels</auteur>
</disque>
```

- La définition d'un/de namespace(s) est optionnelle mais est fortement conseillée
- Si aucun namespace n'est défini, les éléments sont associés au namespace par défaut
- Il est possible d'associer un namespace aux attributs
 - Même namespace que l'élément père
 - Autre namespace que l'élément père

Des éléments et des types

- Il existe des types pré existants, considérés comme simples
 - **xs:string**
 - **xs:decimal**
 - **xs:integer**
 - **xs:boolean**
 - **xs:date**
 - **xs:time**
- Une fois un type choisi il est possible de le restreindre grâce à d'autres balises spécifiées dans les restrictions

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="simpleType">
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="120"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Les attributs XSD

- Une fois les éléments et leurs types déclarés, il est possible de spécifier des attributs pour chaque élément
- Les attributs sont accompagnés d'un type simple et sont déclarés directement dans l'élément
- Comme avec DTD, il est possible de leur assigner une valeur par défaut ou les rendre obligatoires

```
<?xml version="1.0" encoding="utf-8"?>
<xss:schema elementFormDefault="qualified" xmlns:xss="http://www.w3.org/2001/XMLSchema">
    <xss:complexType name="elementType">
        <xss:attribute name="attribute-name" type="xss:string" />
        <xss:attribute name="attribute-name2" type="xss:string" default="test" />
        <xss:attribute name="attribute-name3" type="xss:string" use="required" />
    </xss:complexType>
</xss:schema>
```

Faire le bon choix

- Quel système choisir entre DTD et XSD ?
 - DTD
 - *Hérité de SGML, compatible avec d'anciens systèmes, largement porté*
 - *Syntaxe légère mais obsolète et incomplète*
 - XSD
 - *Système spécifique pour le XML, possède beaucoup plus de capacités pour restreindre et spécifier une grammaire*
 - *Les espaces de nom permettent d'utiliser différents XSD en même temps*
- Il existe encore d'autres alternatives dont une des plus fameuse est RELAX-NG, qui est plus concise mais plus jeune
- Tout dépend des besoins de l'application

XPATH – Langage de requêtes XML

- En l'état, le XML ne permet pas de faire de requêtes pour récupérer des éléments précis
 - C'est pour cela que XPath a été développé
 - Standard créé par le W3C en 1999
-
- Le but est de pouvoir sélectionner des parties de document
 - Il est possible de naviguer dans un fichier XML
 - L'idée est de filtrer les données par de simples opérations sur les éléments XML

Fonctionnement par noeuds

- Comme pour les parseurs XML, XPath se base sur des nœuds
 - Root, la racine du document
 - Element, une balise
 - Text, du contenu textuel
 - Attribute, un attribut d'une balise
 - Comment, commentaire XML

XML d'exemple

```
<?xml version="1.0" encoding="utf-8"?>
<customers>
  <customer>
    <fullname usual-name="Steve">Steven O'daily</fullname>
    <nationality>irish</nationality>
    <age>45</age>
    <child>
      <fullname usual-name="Tommy">Thomas O'daily</fullname>
      <age>13</age>
    </child>
  </customer>
  <customer>
    <fullname>Francis Pirelli</fullname>
    <nationality>italian</nationality>
    <age>17</age>
  </customer>
  <customer>
    <fullname usual-name="Angie">Angelina Ferrando</fullname>
    <nationality>spanish</nationality>
    <age>19</age>
  </customer>
  <customer>
    <fullname>Karina Drovodek</fullname>
    <nationality>russian</nationality>
    <age>34</age>
    <child>
      <fullname usual-name="Vlad">Vladimir Drovodek</fullname>
      <age>3</age>
    </child>
  </customer>
  <customer>
    <fullname usual-name="Tony">Antonio Tival</fullname>
    <nationality>italian</nationality>
    <age>24</age>
  </customer>
</customers>
```

Recherche par location 1/2

- La requête XPath est séparée en plusieurs étapes par des "/"
- Chaque étape a une valeur définissant le/les nœuds sélectionnés
 - / : Sélection depuis la racine
 - *"/customers"* sélectionne la balise *customers* à la racine
 - // : Sélection dans tous les descendants
 - *"//age"* sélectionne tous les tags "age"
 - *NomDeNoeud* : sélection de l'élément
 - *"/customers/customer"* sélectionne tous les tags *customer* dans la balise racine *customers*

Recherche par location 2/2

- .. : Sélection du nœud courant
 - *"/customers/customer/."* fait la même chose qu'au dessus
- .. : Nœud parent
 - *"//age/.."* sélectionne tous les nœuds ayant un sous nœud "age"
- @ : Attribut
 - *"//@usual-name"* sélectionne tous les attributs *usual-name*

Quelques exemples

- //child/..
 - Sélectionne les clients ayant un enfant
- //@usual-name/../../nationality
 - Sélectionne la nationalité des clients ayant un nom d'usage
- /customers/customer/age
 - Sélectionne l'age de tous les clients (sans les enfants)
- //child/age
 - Récupère l'age de tous les enfants
- //child/../fullname/@usual-name
 - récupère les noms d'usage des personnes ayant un enfant

Une sélection sous condition

- Il est possible de conditionner les sélections grâce au crochets
- Les prédictats sont
 - [1] : Premier élément correspondant
 - *//child[1]* premier enfant du document
 - [@attribute] : Ayant l'attribut donné
 - *//customer[@usual-name]* clients ayant un nom d'usage
 - [age>12] : Ayant un âge supérieur à 12
 - *//customer[age>21]* personnes âgées de plus de 21 ans
- Les conditions peuvent être utilisés avec les opérateurs suivant
 - | + - * div = != < <= > >= or and mod

Quelques exemples

- `//child[../fullname/@usual-name]`
 - Sélectionne les enfants ayant pour parent une personne avec un nom d'usage
- `//@usual-name[../../nationality='italian']`
 - Sélectionne le nom d'usage des personnes italiennes
- `//child/age[../../nationality!='irish']`
 - Sélectionne l'age des enfants ayant un parent non irlandais
- `//customer[age>30][age<4*child/age]`
 - Sélectionne les clients de plus de 30 ans ayant au plus 4 fois l'age de leur enfant

- Il est aussi possible de se déplacer sur différents axes
 - ancestor:: – Les ancêtres d'un nœud
 - ancestor-or-self:: – Les ancêtres et soit
 - attribute:: – Les attributs du nœud
 - child:: – Les enfants du nœud
 - descendant:: – Les enfants/petits enfants du nœud
 - descendant-or-self:: – Les descendants du nœud et soit
 - following:: – Tous les nœuds "après" le nœud courant
 - following-sibling:: – Les nœuds frères après le nœud courant
 - parent:: – le parent du nœud courant
 - preceding:: – Les nœuds précédents
 - preceding-sibling:: – Les nœuds frères précédents
 - self:: – Le nœud courant

Quelques exemples

- //customer[nationality='italian']/following-sibling::customer[1]
 - Sélectionne le premier client après chaque italien

- //customer[1]/descendant::fullname
 - Séctionne tous les fullname situés dans le dossier du premier client (enfants inclus)

Des combinaisons et méthodes

- Lorsque l'on veut sélectionner quelque chose pouvant répondre à deux conditions différentes, l'opérateur « | » permet de faire une union des deux résultats
- Méthodes
 - Sur les nœuds
count(), last(), name(), position(), text()
 - Sur les chaînes
concat(a, b), contains(a, b), start-with(a, b), string-length(a), ...
 - Sur les nombres
ceil(a), floor(a), sum(a, b, ...)

XSLT, pourquoi ?

- Il est parfois intéressant de faire une transformation du contenu XML en un autre
- Changer un document XML en XHTML
- Convertir un format XML vers un autre
- Filtrer des données ne devant pas figurer sur un autre format XML
- XSLT, eXtensible Stylesheet Language Transformations est fait pour ces opérations délicates

Fichier XLST (XLS template)

- Le fichier XSLT doit commencer par la balise racine `<xsl:stylesheet>` ou `<xsl:transform>`, les deux étant synonymes
- Le contenu sera une suite de templates appliqués sur des sections du fichier XML pris en entrée

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/customers/customer[1]">
  <html>
    <head><title><xsl:value-of select="fullname/@usual-name" /></title></head>
    <body>
      <h1>Customer</h1>
      Name: <xsl:value-of select="fullname/text()" /><br />
      Age: <xsl:value-of select="age/text()" /><br />
      Nationality: <xsl:value-of select="nationality/text()" />
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Définition d'un template principal

- Le template principal sélectionne une portion du document XML à convertir et fourni un résultat en réintégrant les valeurs du fichier XML
- L'insertion de données se fait avec l'aide de quelques balises
 - <xsl:value-of select="xpath">
 - *Affiche une valeur*
 - <xsl:for-each select="xpath">
 - *Boucle sur les résultats de sélection*
 - <xsl:if test="expression">
 - *Execute une partie de template si le teste est réussi*
 - <xsl:choose> : Permet de faire une condition et son opposée
 - <xsl:when test="expression"> : *un cas du choose*
 - <xsl:otherwise> : *cas par défaut*

Exemple complet de transformation

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/customers">
  <html>
    <head><title>Customers list</title></head>
    <body>
      <h1>Customers</h1>
      <xsl:for-each select="customer">
        <div>
          Name: <xsl:value-of select="fullname/text()" /><br />
          Age: <xsl:value-of select="age/text()" /><xsl:if test="age/text() &lt;
21"> Not major in every country</xsl:if><br />
          Nationality: <xsl:value-of select="nationality/text()" /><br />
          <xsl:choose>
            <xsl:when test="child">
              <strong>Has a child</strong><br />
              Name: <xsl:value-of select="child/fullname/text()" /><br />
              Age: <xsl:value-of select="child/age/text()" />
            </xsl:when>
            <xsl:otherwise>
              <strong>Hasn't a child</strong>
            </xsl:otherwise>
          </xsl:choose>
        </div>
        <hr />
      </xsl:for-each>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Délégation de tâche

- Lorsqu'un seul template devient important, il est rapidement impossible de le maintenir
- Une option permet de découper le template principal en appliquant des sous templates
- **<xsl:apply-templates/>** est utilisé pour invoquer d'autres templates
- Lors d'un apply-templates, tout template matchant l'élément courant sera invoqué
- Il est possible de changer l'élément actuel uniquement pour le apply-template via l'attribut **select**

Templates par l'exemple

- Voici l'utilisation d'un second template pour déléguer l'affichage d'un enfant

```
<xsl:template match="child">
  <strong>Has a child</strong><br />
  Name: <xsl:value-of select="fullname/text()" /><br />
  Age: <xsl:value-of select="age/text()" />
</xsl:template>
```

- Pour faire appel à ce template, il suffit remplacer l'ancien bloc par :

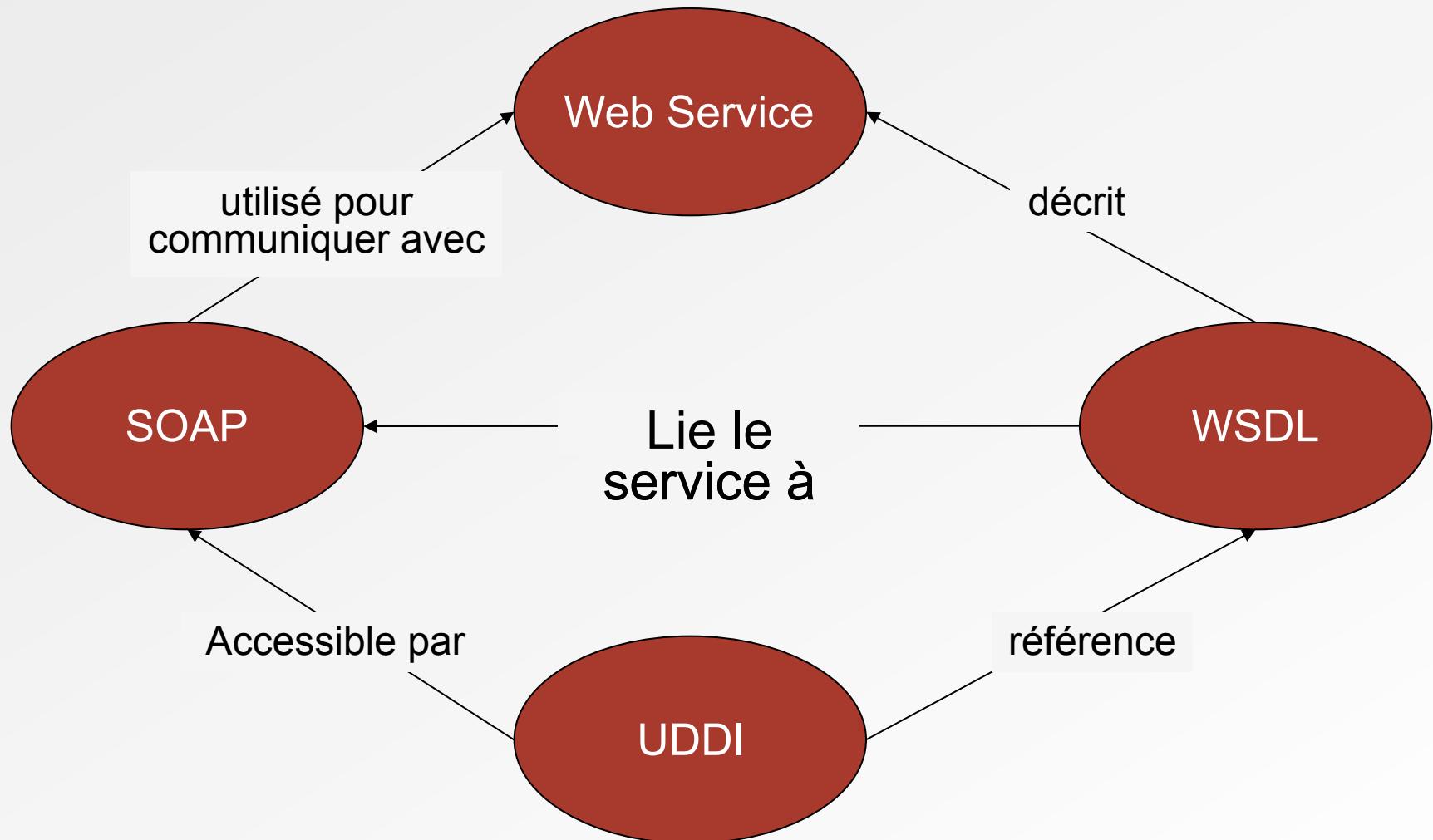
```
<xsl:apply-templates select="child" />
```

Les Web Services

- Deux grandes familles : SOAP & REST
- Java et les Web Services (frameworks)
- Web Services asynchrones
- Le déploiement
- Les bonnes pratiques en Web Services
- Aller plus loin (WSDL, WS-*, MTOM, etc.)

- WS-*
 - Solution standard
 - RPC synchrone et/ou asynchrone
 - S'appuie sur SOAP (communication) & WSDL (contrat)
 - Intégration d'un écosystème orienté entreprise
 - *WS-Security / WS-Addressing / Ws-ReliableMessaging ...*
- REST
 - S'appuie sur les fondations HTTP
 - « Architecture Orientée Ressources »
 - Opérations de base : GET / POST / PUT / DELETE
 - Alternative sérieuse au monde WS-*
 - *Simple à mettre en œuvre*
 - *Pas de nouvelles technologies à maîtriser*

Les fondations des Web Services WS-* (1/3)



Les fondations des Web Services WS-* (2/3)

- WSDL
 - Décrit le service
 - WSDL = Contrat
- SOAP
 - Langage/Protocole de communication
 - Permet la transmission de messages standardisés entre acteurs d'un système distribué
- Web Services
 - Implémentation du service
 - Indépendant des technologies : Java, .Net, Python ...
- UDDI
 - Référence les services
 - « Parent pauvre » du monde WS-*

REST : retour vers le futur

- REST = REpresentational State Transfer
- Crée par Roy T. Fielding en 2000
- Le principe de REST est de revenir aux fondamentaux de l'architecture du Web
 - URL → identifie les ressources
 - HTTP → fournit les opérations
 - *GET* : récupérer une ressource
 - *POST* : créer une ressource
 - *PUT* : mettre à jour une ressource
 - *DELETE* : supprimer une ressource
- REST s'appuie sur les principes du web pour mettre en œuvre des web services CRUD (ie. Create/Read/Update/Delete)

WS-* et REST : la guerre ?

- WS-* (SOAP / WSDL)
 - Majoritaire au sein des systèmes d'informations
 - Web Service = SOAP + WSDL
 - Gestion de la sécurité des données
- REST
 - Architecture de plus en plus courante
 - Adaptée aux opérations CRUD
 - Adaptée à la gestion des ressources

*On a souvent tendance à opposer ces deux mondes
pourtant
WS-* et REST sont complémentaires*

Web Services et Java

- Le standard WS-* au sein de la plateforme JEE est JAX-WS
- JAX-WS est le successeur de JAX-RPC qui est aujourd'hui obsolète (API « pruned » dans JEE 6)
- JSR 181 fournit à JAX-WS les annotations Web Services
- SAAJ est une spécification/API permettant de manipuler des enveloppes SOAP à un niveau d'abstraction plus bas que JAX-WS (SAAJ est notamment utilisé dans le cadre des *Handlers*)
- JAX-RS est l'API Java de référence pour la mise en œuvre d'une architecture REST

JAX-WS et JAX-RS : les implémentations

- JAX-WS
 - Apache CXF
 - Apache Axis 2
 - Oracle/Sun Metro (RI)
- JAX-RS
 - Apache CXF
 - Oracle/Sun Jersey (RI)
- Chacune des implémentations apporte des fonctionnalités équivalentes et/ou différentes
 - Binding XML (JAXB, Aegis, XMLBeans ...)
 - Support WS-* (WS-Addressing, WS-Security ...)

- CXF est un projet Apache dédié à la mise en œuvre des Web Services WS-* et REST
- CXF est le successeur de XFire au sein de la fondation Apache
- CXF nous offre les avantages suivants
 - Respecte les standards Web Service WS-* et REST
 - Léger
 - Performant
 - Intégration à Spring
 - Lien fort avec les projets « Intégration » de la fondation Apache : ActiveMQ, ServiceMix, Camel

- JAX-WS est née officiellement en 2006
 - Adoption par l'industrie
 - Retour d'expérience
- Il est tout à fait possible de ne pas utiliser JAX-WS pour développer des Web Services au sein de la plateforme JEE
 - Servlet + XML
 - JAX-RPC
 - API propriétaire
- JAX-WS est malgré tout conseillée pour les développements Web Services
 - Standard
 - Simple

JAX-WS en détail

- JAX-WS permet de simplifier le développement de Web Services
- Pour cela, JAX-WS s'appuie sur
 - JAXB pour le mapping Java/XML
 - JSR 175 pour les annotations Java
 - JSR 181 pour les annotations Web Services
- Comparé à JAX-RPC, JAX-WS
 - Simplifie les développements Web Services
 - Gère les appels asynchrones
 - Gère les appels non HTTP

Développer un Web Service SOAP en Java

Comment développer un WS en Java ?

- Java-First (ou Code-First ou Bottom-Up)
 1. Implémentation du service
 2. Génération automatique du fichier WSDL et du schéma XSD
- Contract-First (ou Top-Down)
 1. Crédit : Création du fichier WSDL et du schéma XSD
 2. Génération automatique des classes Java
 - . Classes JAXB (requête/réponse/erreurs)
 - . Interface du service
 3. Implémentation du service

Étapes de création d'un WS en Java-First

1. Créer un Service *Endpoint Interface* (SEI)

Le SEI consiste en une interface Java décrivant les opérations exposées par le Web Service : c'est le contrat liant le service aux clients

1. Implémenter le Web Service

Il s'agit d'une classe Java implémentant le SEI

1. Annoter les classes afin d'assurer la compatibilité avec JAX-WS

2. Déployer le Web Service

Créer l'interface Java (SEI)

```
package com.resanet.ws;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

@WebService(targetNamespace = "http://www.resanet.partenaires.com")
public interface VoyageService {

    @WebMethod(operationName = "verifierDisponibilite")
    @WebResult(name = "disponibilite")
    public boolean estDisponible(
        @WebParam(name = "reference")
        String ref
    );
}
```



SEI = Contrat du Web Service

Toutes les méthodes de l'interface sont exposées

Implémenter le service

```
package com.resanet.ws;

import javax.jws.WebService;

@WebService(serviceName = "voyageService", endpointInterface =
"com.resanet.ws.VoyageService")
public class VoyageServiceImpl implements VoyageService {

    public boolean estDisponible(String ref) {
        ...
    }
}
```

- ➡ Implémente les opérations décrites par le SEI
- ➡ L'implémentation utilise la même annotation
`@WebService` que le SEI

Recommandations

- La mise en œuvre du SEI est optionnelle mais fortement conseillée
 - Une interface « cache » l'implémentation
 - Permet de respecter l'approche « Contrat » des Web Services
- L'implémentation explicite (ie. `implements`) est optionnelle mais fortement conseillée afin d'éviter des incohérences SEI/SIB
- Chacune des annotations fournit plusieurs paramètres → il est important de fournir le plus d'informations possibles afin d'éviter toute ambiguïté

Publication d'un Web Service

```
package com.resanet.ws;

import javax.xml.ws.Endpoint;

public class ServerVoyage {

    public static void main(String[] args) {

        VoyageService service = new VoyageServiceImpl();
        String address = "http://localhost:9000/voyage";
        Endpoint.publish(address, service);
    }

}
```

La publication « *Endpoint.publish* » est intéressante dans le cadre de POC / Prototypage mais ne doit pas être utilisée en production

Développer un client Web Service

```
package com.resanet.ws;

import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;

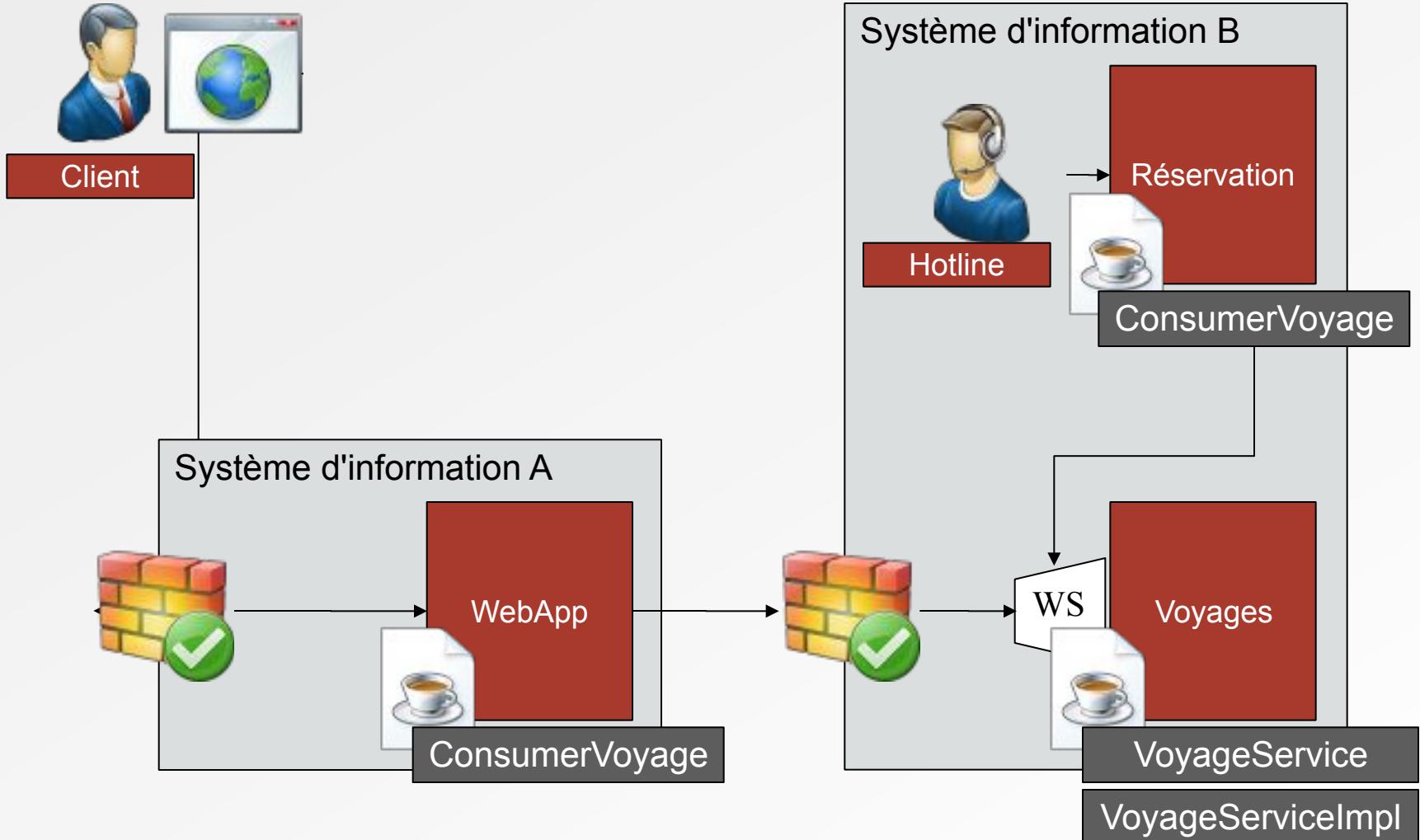
public class ConsumerVoyage {

    public static void main(String[] args) {
        try {
            URL wsdlURL = new URL("http://localhost:9000/voyage?wsdl");
            QName serviceName = new QName("http://
www.resanet.partenaires.com",
"VoyageService");

            Service service = Service.create(wsdlURL, serviceName);
            VoyageService client = service.getPort(VoyageService.class);

        [...]
```

Appel à un Web Service



Derrière le Java-First (coté Serveur)

- La publication du Web Service
 1. Génère le WSDL (ie. « Contrat »)
 2. Ouvre le port spécifié
- La réception de la requête SOAP
 1. Désérialise la requête SOAP (ie. XML → Java)
 2. Appelle la fonction Java correspondante
 3. Génère une réponse SOAP (ie. Java → XML)
 4. Envoie la réponse SOAP via HTTP

Derrière le Java-First (coté Client)

- L'appel au Web Service
 1. Génère une requête SOAP (ie. Java → XML)
 2. Envoie la requête SOAP via HTTP
 3. Met en attente le client
- La réception de la réponse SOAP
 1. Désérialise la requête SOAP (ie. XML → Java)
 2. Stoppe l'attente du client
 3. Renvoie la réponse Java

Format de la requête/réponse SOAP

JAVA

Document/Literal
« Wrapped »

```
@WebService
public interface AccueilService {

    @WebResult(name = "retour")
    public String afficherMessage(
        @WebParam(name = "nom")
        String nom
    );

}
```

SOAP

```
<Envelope ...>
  <Body>
    <ws:afficherMessage>
      <nom>Raphaël</nom>
    </ws:afficherMessage>
  </Body>
</Envelope>
```

```
<Envelope ...>
  <Body>
    <ws:afficherMessageResponse>
      <retour>
        Bienvenue Raphaël
      </retour>
    </ws:afficherMessageResponse>
  </Body>
</Envelope>
```



SOAP

SOAP : Protocole de communication

- SOAP : protocole et format de messages défini par W3C
 - A l'origine, SOAP = *Simple Object Access Protocol*
 - SOAP n'est plus un acronyme
 - SOAP aujourd'hui ≈ « *SOA Protocol* »
- Permet la transmission de messages standardisés entre acteurs d'un système distribué
- SOAP : successeur de XML-RPC
 - XML-RPC moins complet que SOAP
 - Orienté RPC uniquement
- Versions
 - 1.1 : mai 2000 – version la plus couramment utilisée
 - 1.2 : juin 2003

SOAP, c'est quoi ?

- Un message SOAP
 - Est un message XML
 - Possède une structure propre (ie. enveloppe SOAP)
 - Est indépendant de la couche transport : HTTP, JMS, ...
- Un message SOAP contient les éléments
 - **Envelope** qui en est est la racine
 - **Header** qui fournit des informations techniques :
 - *Routage*
 - *Adressage* ...
 - **Body** qui contient le contenu utile (ie. payload) du message :
 - *Requête*
 - *Réponse*
 - *Erreur*

L'enveloppe SOAP

Enveloppe

Entête

Corps

```
<soapenv:Envelope ...>

<soapenv:Header>
  <add:MessageID>
    132465789
  </add:MessageID>
</soapenv:Header>

<soapenv:Body>
  <ns:rechercherVoyage>
    <ns:depart>Paris</ns:depart>
    <ns:arrivee>Rennes</ns:arrivee>
    <ns:date>2009-12-31</date>
  </ns:rechercherVoyage>
</soapenv:Body>

</soapenv:Envelope>
```

- La racine de l'enveloppe SOAP doit spécifier la version SOAP
 - SOAP 1.1

```
<soapenv:Envelope  
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
```

- SOAP 1.2

```
<soapenv:Envelope  
    xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
```

- Remarques
 - SOAP 1.1 et 1.2 sont relativement proches
 - SOAP 1.2 clarifie les ambiguïtés présentes dans SOAP 1.1 ; malgré cela, SOAP 1.1 reste majoritaire

Header SOAP

- Le Header SOAP est optionnel
- Le Header SOAP peut contenir n'importe quelle structure XML
- Il peut contenir des données utilisateur « techniques » (p. ex. identifiant, timestamp ...) ou des informations liées aux spécifications WS-* (p. ex. WS-Addressing, WS-Security ...)

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:ns="http://www.resanet.com">  
  <Header>  
    <ns:id>ID_0123</ns:id>  
  </Header>  
  <Body>  
    ...  
  </Body>  
</Envelope>
```

Body SOAP (1/2)

- Le Body SOAP contient le message à transmettre au destinataire final
- Le Body SOAP est obligatoire
- Body SOAP = « Payload »
- Une ou plusieurs structures XML peuvent être transmises via le Body SOAP
 - Le format du contenu est spécifié par le fichier WSDL
 - Document ou RPC

Body SOAP (2/2)

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:ns="http://www.resanet.com">  
  <Body>  
    <ns:enregistrerUtilisateur>  
      <ns:utilisateur>  
        <ns:nom>Holmes</ns:nom>  
        <ns:prenom>Sherlock</ns:prenom>  
        <ns:adresse>221B Baker Street</ns:adresse>  
        <ns:ville>Londres</ns:ville>  
      </ns:utilisateur>  
    </ns:enregistrerUtilisateur>  
  </Body>  
</Envelope>
```

Le contenu du Body est déterminé par les paramètres JAX-WS (ie. annotations)

→ Plus d'informations au chapitre WSDL

- Unique structure XML du Body SOAP définie par les spécifications SOAP
- Permet de remonter une erreur au client SOAP
- Une Fault SOAP est remontée au sein du Body uniquement en cas d'erreur afin d'informer le client du problème
- JAX-WS s'appuie sur les exceptions Java afin de générer les Fault SOAP

L'élément *soap:Fault* contient quatre sous-éléments

- faultcode : identifiant de l'erreur basé sur types prédéfinis → déterminé automatiquement par CXF
 - VersionMismatch : version de SOAP non valide
 - MustUnderstand : header non « compris »
 - Client : message mal formatée ou non valide
 - Server : erreur de traitement du message
- faultstring : description → attribut message de l'exception
- faultactor : URL du composant à l'origine de l'erreur → non pris en charge par le mapping JAX-WS (pris en charge par les Handlers)
- detail : informations concernant l'erreur → Élément correspondant à l'exception

Mapping Exception/Fault par défaut

```
@WebService(targetNamespace = "http://business.resanet")
public interface BusinessService {
    ...
    public int reserver(...) throws VoyageNonDisponibleException;
    ...
}
```

Java



```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Body>
        <Fault>
            <faultcode>soap:Server</faultcode>
            <faultstring>Le voyage n'est pas disponible</faultstring>
            <detail>
                <ns1:VoyageNonDisponibleException xmlns:ns1="http://
business.resanet"/>
            </detail>
        </Fault>
    </Body>
</Envelope>
```

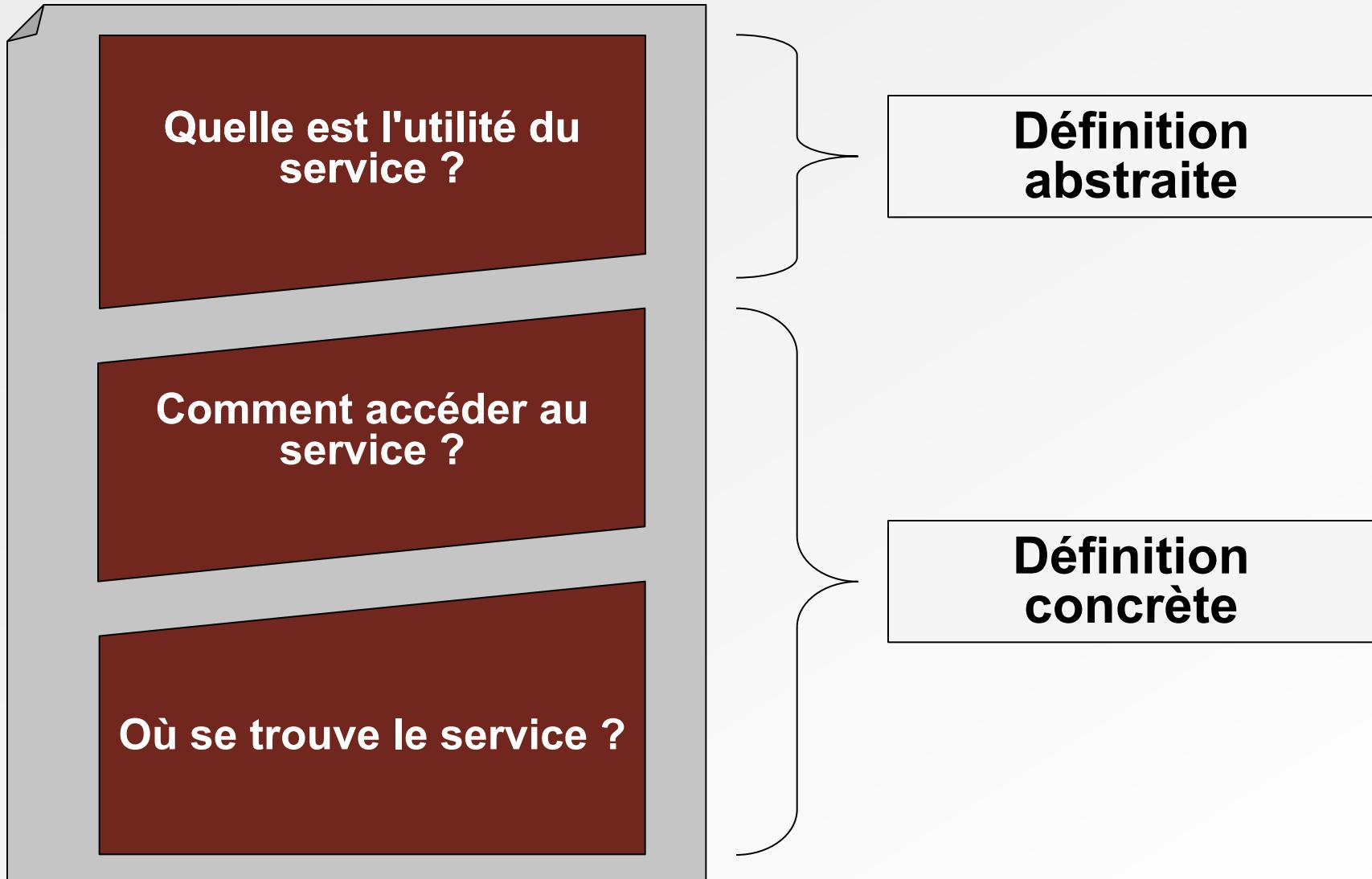
SOA

WSDL

Qu'est ce que le contrat d'un Web Service

- Un contrat de Web Service doit fournir
 - Son usage (opérations fournies)
 - Les messages échangés
 - Modèle de données des messages
 - Informations précisant comment utiliser le service
 - Informations précisant où trouver le service
- Un contrat de Web Services est organisé en 3 parties
 - **Quelle** est l'utilité et les possibilités offertes par le service ?
 - **Comment** accéder au service ?
 - **Où** accéder au service ?

Structure d'un contrat



Comment écrire un contrat

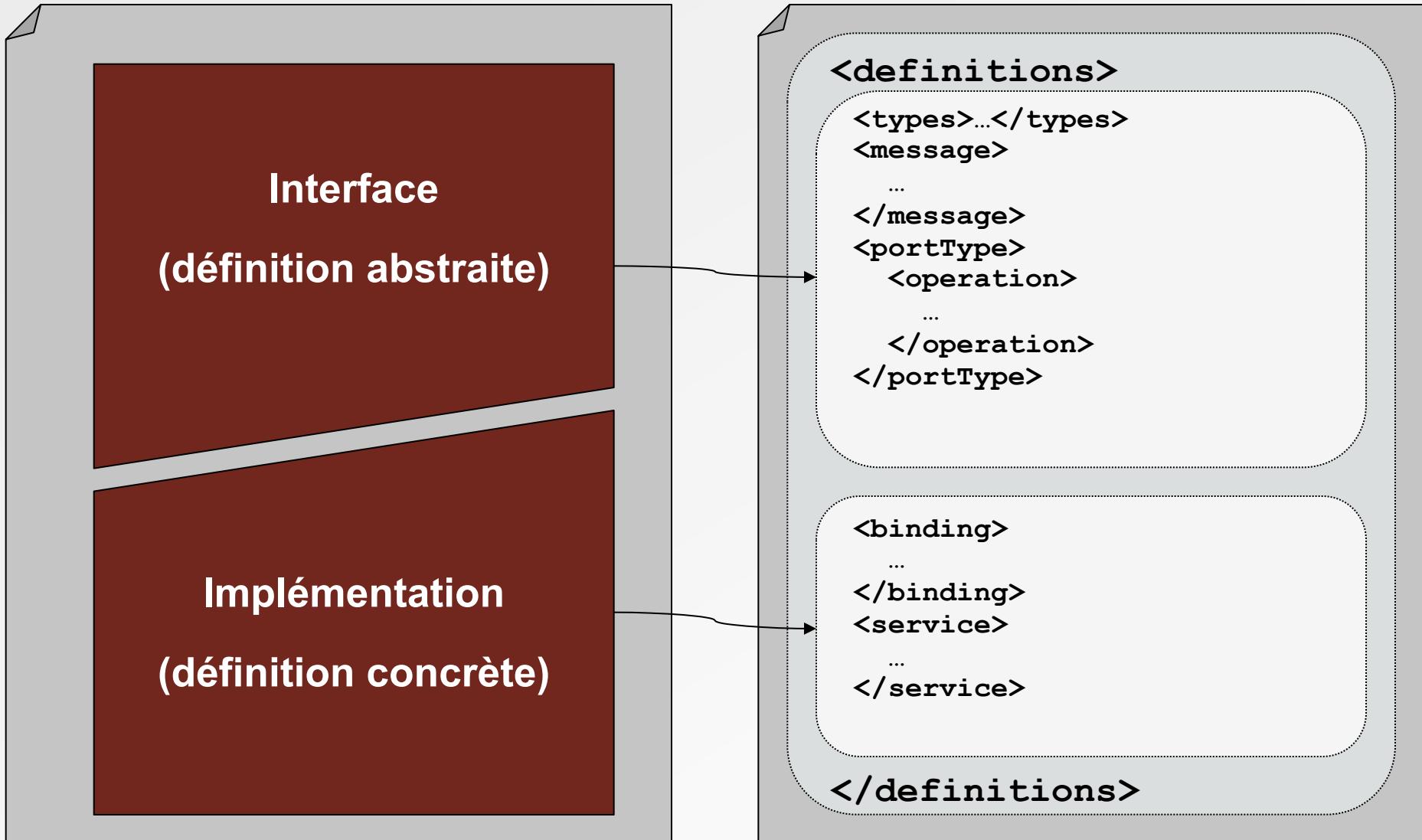
- Tout document contenant une définition abstraite et concrète est un contrat de Web Services
- Quel format utilisé ?
 - Texte
 - Word
 - Mail
 - XML
 - ...
- Démarche SOA → **s'appuyer sur les standards**



Standard contrat WS = WSDL

- WSDL : *Web Services Description Language*
 - *ie. Description de Web Services*
- WSDL = définition du service
- WSDL = langage XML
- Standard défini par le W3C
 - En mars 2001 (soit après SOAP)
 - Dernière version : 2.0 (non prise en charge par CXF)

Structure d'un WSDL



WSDL : définition abstraite/concrète

- Définition abstraite (ou logique)
 - Contenu des messages (entrée/sortie)
 - Opérations possibles
 - Paramètres*
 - Erreurs*
- Définition concrète (ou physique)
 - Implémentation du service
 - Format des messages*
 - Type de communication*
 - Emplacement du service
 - Adresse réseau*

Éléments WSDL : definitions (1/2)

- <definitions>
 - Racine du document
 - Définit le nom du service et son *namespace*
 - Peut contenir les *namespaces* utilisés par le service
- Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    name="VoyageService"
    targetNamespace="http://ws.resanet.com/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://ws.resanet.com/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    ...
</definitions>
```

Éléments WSDL : types

- <types>
 - Définition des types/éléments XSD
 - Possibilité d'importer des schémas XSD
- Exemple

```
<wsdl:types>
  <xsd:schema
    targetNamespace="http://ws.resanet.com/agence"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <xsd:import
      namespace="http://ws.resanet.com/voyages"
      schemaLocation="voyages.xsd"/>
    ...
  </xsd:schema>
</wsdl:types>
```

Éléments WSDL : messages

- <message>
 - Définition des messages (entrée/sortie/erreur)
 - S'appuie sur les éléments/types des schémas XSD
- Exemple

```
<message>
  <part      name="parameters"
              element="tns:voyageService"/>
</message>
<message>
  <part      name="return"
              element="tns:voyageServiceResponse"/>
</message>
<message>
  <part      name="estDisponibleFault"
              element="tns:estDisponibleFault"/>
</message>
```

Éléments WSDL : portType

- <portType>
 - *PortType* = Interface
 - Définit les opérations fournies par le service (<operation>)
 - Un message peut être composé de un ou plusieurs éléments/types
- Exemple

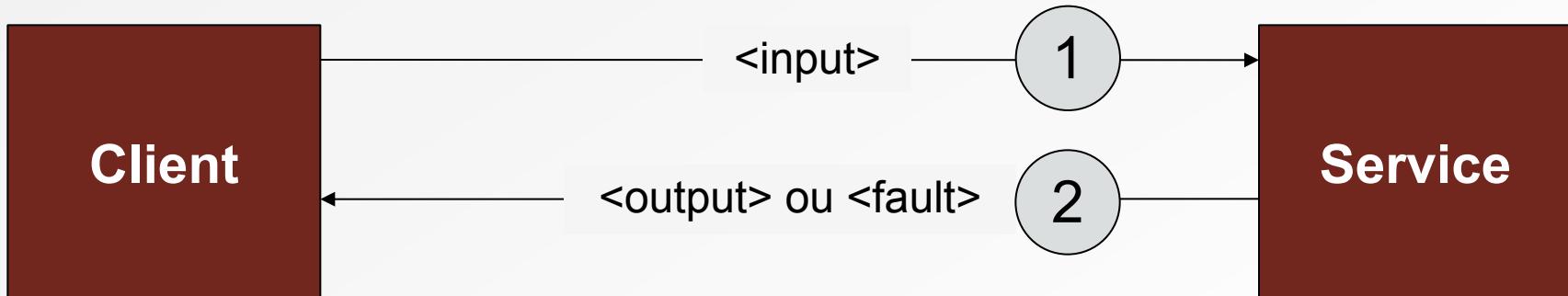
```
<portType name="VoyageService">
  <operation name="operation1">
    ...
  </operation>
  <operation name="operation2">
    ...
  </operation>
</portType>
```

Éléments WSDL : les opérations

- WSDL 1.1 définit 4 types d'opérations
- Chaque type d'opération est défini par un MEP
- MEP : *Message Exchange Pattern*
- MEP disponibles
 - *One-Way*
 - *Request-Response*
 - *Solicit-Response*
 - *Notification*

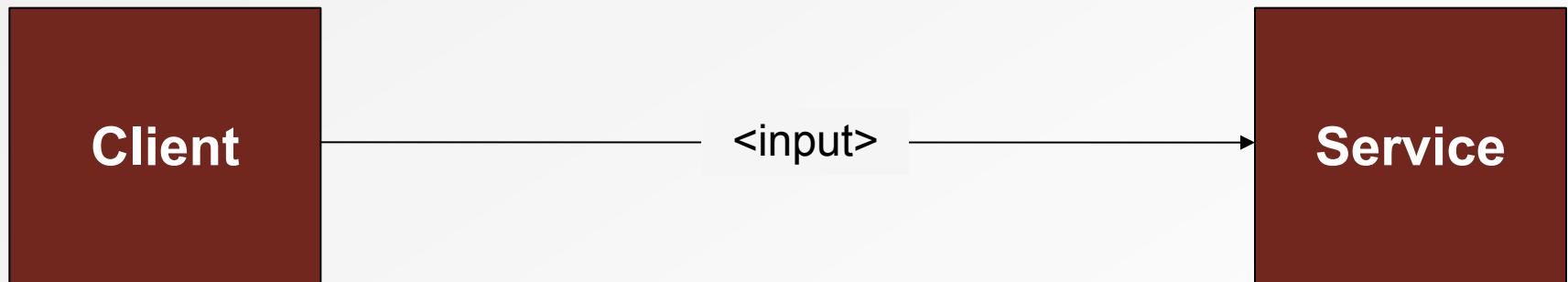
Request-Response

- L'opération reçoit une requête (*input*) et envoie une réponse (*output*) ou une erreur (*fault*)
- Type d'opération le plus courant
- Paramètres
 - input (obligatoire) : requête
 - output (obligatoire) : réponse
 - fault (optionnel) : erreur



One-Way

- L'opération reçoit une requête (*input*) ; aucune réponse n'est renvoyée à l'appelant
- Paramètres
 - input (obligatoire) : requête



Request-Response / One-way : exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ...>
...
<portType name="VoyageService">

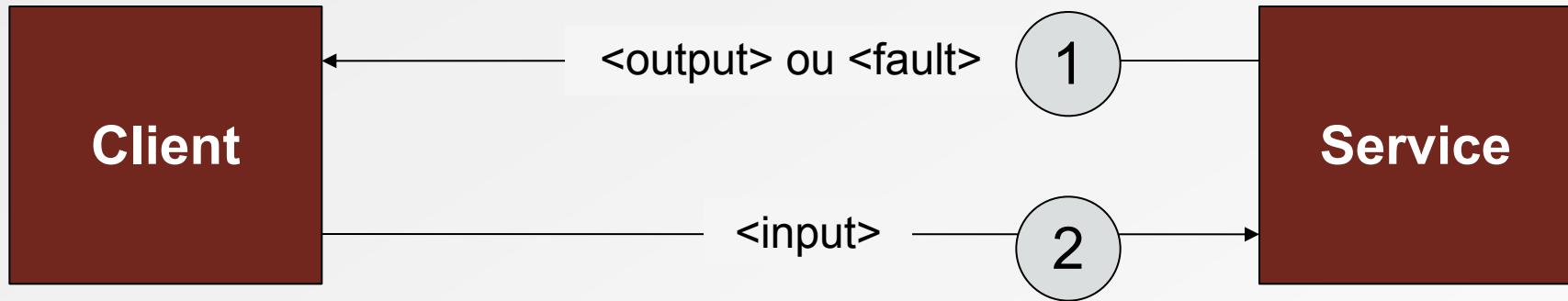
    <operation name="estDisponible">
        <input message="tns:voyage"/>
        <output message="tns:disponibilite"/>
        <fault name="fault" message="tns:estDisponibleFault"/>
    </operation>

    <operation name="commander">
        <input message="tns:commande"/>
    </operation>

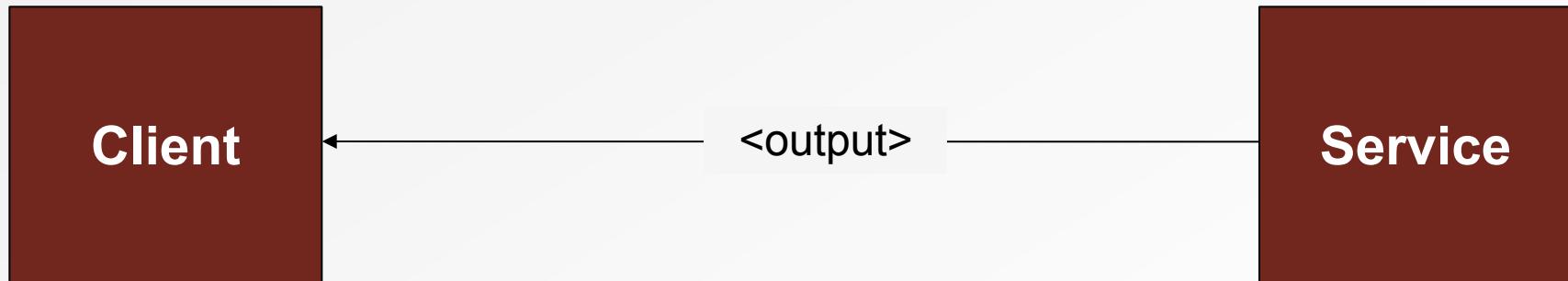
</portType>
...
</definitions>
```

Solicit-Response et Notification (1/2)

- **Solicit-Response**



- **Notification**



Solicit-Response et Notification (2/2)

- Le Web Service est à l'initiative de la requête
 - Nécessite l'envoi initial d'une adresse de callback client
 - Le principe de callback n'a jamais été standardisé pour ces MEP
- Opérations interdites par le WS-I Basic Profile

R2303 - A DESCRIPTION MUST NOT use **Solicit-Response** and **Notification** type operations in a wsdl:portType definition.

- Types d'opérations rarement implémentés
 - Non pris en charge par CXF

Éléments WSDL : binding

- <binding>
 - Relie un *portType* (ie. description abstraite) à des technologies spécifiques de communication
 - Permet au consommateur final de savoir comment appeler le service
- Exemple

```
<wsdl:binding
  name="VoyageServiceImplServiceSoapBinding"
  type="tns:VoyageService">
  <soap:binding
    style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="estDisponible">
    ...
  </wsdl:operation>
</wsdl:binding>
```

Binding SOAP Document/Literal via HTTP

```
<binding
  name="voyage_doc_soap11"
  type="tns:voyageServicePT">
  <soap:binding
    style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="estDisponible">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output><soap:body use="literal" /></output>
    <fault name="fault">
      <soap:fault name="fault" use="literal" />
    </fault>
  </operation>
  ...
</binding>
```

Deux types de binding sont couramment utilisés

- **RPC/Literal**
 - + Le nom de l'opération est stockée directement dans le *Body* du message
 - La validation des données est difficile
- **Document/Literal**
 - + les <message> sont stockés directement dans le *Body* du message
SOAP → la validation est simple
 - Le nom de l'opération appelée n'est pas disponible
- Remarque sur l'encodage ***Literal*** : l'encodage des requêtes/
réponses s'appuie uniquement sur les schémas XSD
(contrairement à SOAP/encoded)

Requête RPC/Literal

```
<message name="requete">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:int"/>
</message>
<message name="void"/>

<portType name="portType">
    <operation name="methode">
        <input message="requete"/>
        <output message="void"/>
    </operation>
</portType>
```

WSDL

```
<soap:envelope>
    <soap:body>
        <methode>
            <x>5</x>
            <y>5</y>
        </methode>
    </soap:body>
</soap:envelope>
```

SOAP

Requête Document/Literal

```
<types>
  <schema>
    <element name="xElement" type="xsd:int"/>
    <element name="yElement" type="xsd:int"/>
  </schema>
</types>

<message name="requete">
  <part name="x" element="xElement"/>
  <part name="y" element="yElement"/>
</message>
<message name="void"/>

<portType name="portType">
  <operation name="methode">
    <input message="requete"/>
    <output message="void"/>
  </operation>
</portType>
```

WSDL

```
<soap:envelope>
  <soap:body>
    <xElement>5</xElement>
    <yElement>5</yElement>
  </soap:body>
</soap:envelope>
```

SOAP

Binding : les bonnes pratiques

- Document/Literal « Wrapped » : standard de facto dans le monde des Web Services → permet de bénéficier des avantages respectifs de RPC et de Document
- Document/Literal « Wrapped » = Document/Literal + conventions de typage/nommage
- Conventions « Wrapped »
 - Message d'entrée
 - *Un part est défini*
 - *Le part est un élément*
 - *L'élément prend le même nom que l'opération*
 - Message de sortie
 - *L'élément respecte les mêmes conventions mis à part pour le nom qui prend le nom de l'opération suffixé par « Response » (exemple : reserver → reserverResponse)*

Requête Document/Literal « Wrapped » (1/2)

```
<xs:schema elementFormDefault="unqualified" targetNamespace="http://ws.resanet.com/voyages"
version="1.0" xmlns:tns="http://ws.resanet.com/voyages"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="estDisponible">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="depart" type="xs:string" />
      <xs:element name="arrivee" type="xs:string" />
      <xs:element name="date" type="xs:dateTime" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="estDisponibleResponse">
  ...
</xs:schema>
```

Requête Document/Literal « Wrapped » (2/2)

```
<Envelope
    xmlns="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:voy="http://ws.resanet.com/voyages">

    <Header/>

    <Body>
        <voy:estDisponible>
            <voy:depart>Paris</voy:depart>
            <voy:arrivee>Rennes</voy:arrivee>
            <voy:date>2009-07-31</voy:date>
        </voy:estDisponible>
    </Body>

</Envelope>
```

Éléments WSDL : service

- <service>
 - Spécifie l'emplacement du service (ie. endpoint)
 - Il est possible de spécifier l'adresse correspondante à chaque binding
- Exemple

```
<service name="voyageService">
  <port
    name="voyageServicePort"
    binding="tns:voyage_doc_soap11">
    <soap:address
      location="http://localhost:8080/voyages" />
  </port>
</service>
```

Étapes de création d'un WS en Contract-First

- 1.Créer un WSDL
- 2.Générer le code JAX-WS à partir du WSDL
- 3.Implémenter le Web Service à partir du code généré
- 4.Déployer le Web Service

Génération Java à partir du WSDL

- CXF fournit l'outil wsdl2java afin de générer les classes Java à partir d'un fichier WSDL
- Exemple

```
> wsdl2java -p com.resanet.ws -impl -d <destination> voyages.wsdl
      ↑           ↑
    Nom du package des classes   Implémentation du service
```

- L'outil wsdl2java génère
 - L'ensemble des classes JAXB
 - L'interface du service (nom de la classe = nom du *portType*)
 - Une implémentation du service à compléter (nom de la classe = nom du *portType* + « *Impl* »)
 - Une classe cliente (nom de la classe = nom du *service*)

```
...
@WebService(targetNamespace = "http://ws.resanet.com", name = "voyagePortType")
@XmlSeeAlso({ObjectFactory.class})
public interface VoyagePortType {

    @RequestWrapper(localName = "estDisponible", targetNamespace = "http://
ws.resanet.com", className = "com.resanet.ws.EstDisponible")

    @ResponseWrapper(localName = "estDisponibleResponse", targetNamespace =
"http://ws.resanet.com", className = "com.resanet.ws.EstDisponibleResponse")

    @WebResult(name = "disponible", targetNamespace = "http://ws.resanet.com")

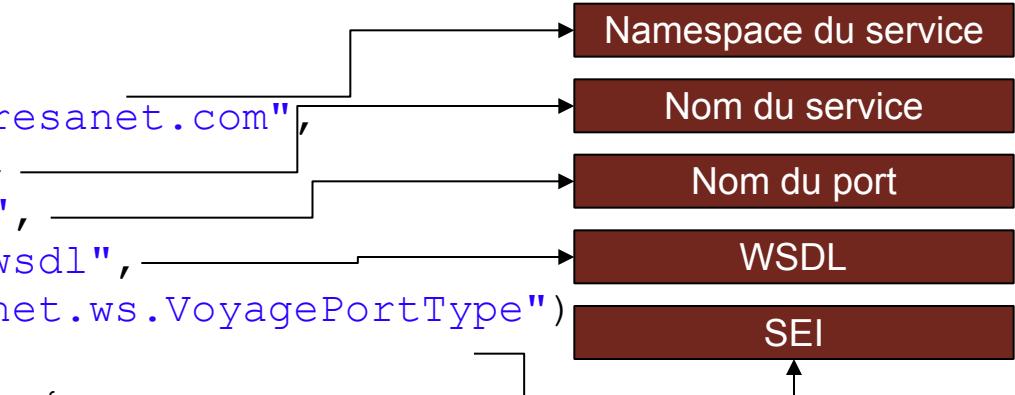
    @WebMethod

    public boolean estDisponible(
        @WebParam(name = "ref", targetNamespace = "http://ws.resanet.com")
        String ref
    );
}
```

Implémentation du service

- L'implémentation du service est un POJO annoté avec `@WebService` (ie. comme le SEI)
- Il est possible de créer le squelette de l'implémentation via le paramètre `-impl` de `wsdl2java`

```
...  
@WebService (  
    targetNamespace = "http://ws.resanet.com", _____) → Namespace du service  
    serviceName = "voyageService", _____) → Nom du service  
    portName = "voyageServicePort", _____) → Nom du port  
    wsdlLocation = "wsdl/voyages.wsdl", _____) → WSDL  
    endpointInterface = "com.resanet.ws.VoyagePortType") → SEI  
public class VoyagePortTypeImpl  
    implements VoyagePortType {  
  
    public boolean estDisponible() { ... }  
    ...  
}
```



The diagram illustrates the mapping of Java annotations to service metadata. It shows a sequence of annotations from a Java code snippet and arrows pointing to corresponding brown boxes labeled with service details:

- `targetNamespace` points to "Namespace du service".
- `serviceName` points to "Nom du service".
- `portName` points to "Nom du port".
- `wsdlLocation` points to "WSDL".
- `endpointInterface` points to "SEI".

Comparaison Code-First/Contract-First

- Code-First
 - Est facile et rapide à mettre en œuvre pour des cas simples (ie. XML basique)
 - Devient plus complexe et long à mettre en œuvre sur des services manipulant des données structurées
 - contrat (ie. WSDL) difficile à maîtriser → peut poser des problèmes d'interopérabilité
- Contract-First
 - Nécessite de maîtriser WSDL et XSD
 - Plus long à mettre en œuvre
 - Garantit l'interopérabilité du service



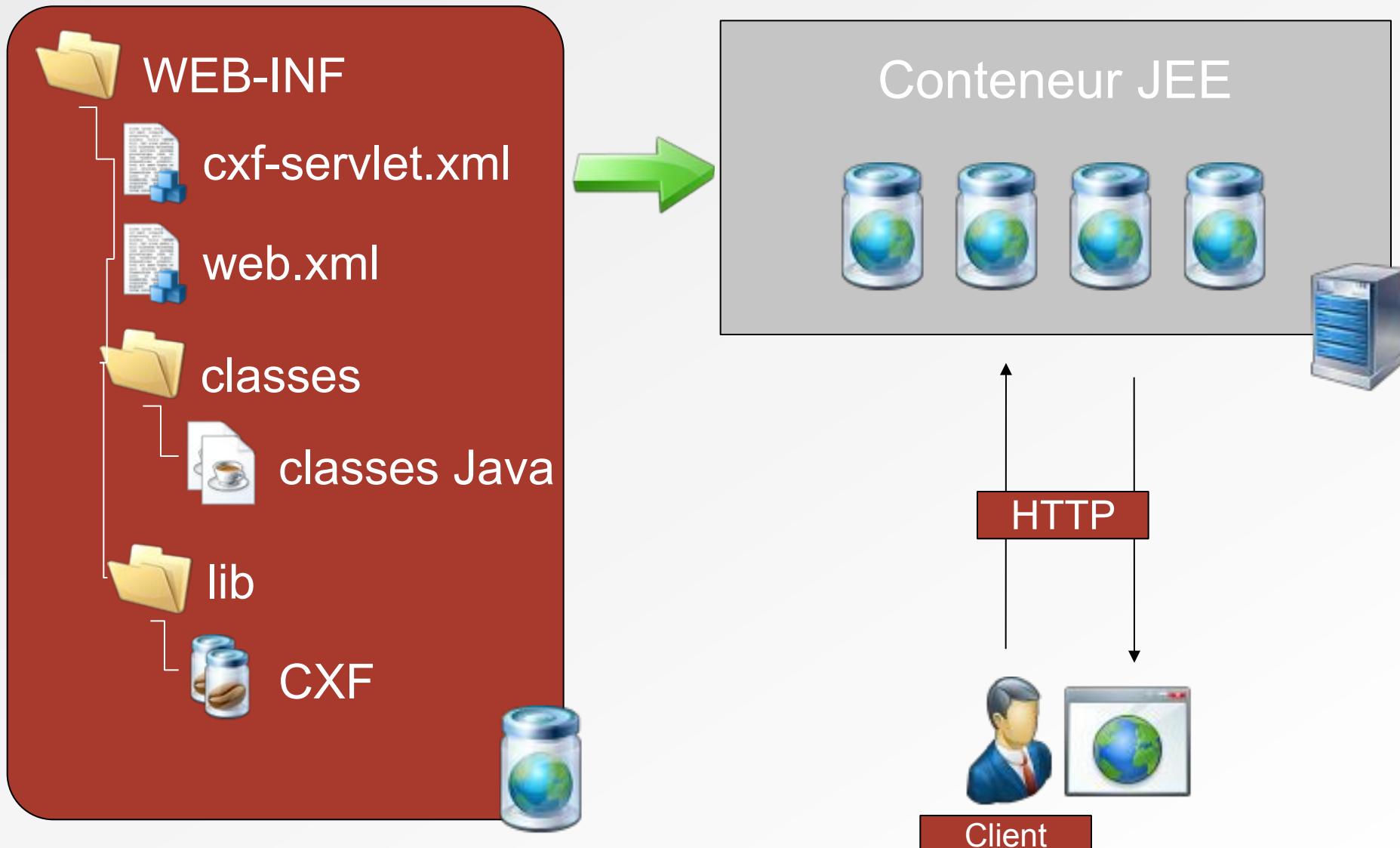
Il est fortement conseillé de développer en **Contract-First**

Déploiement Serveur

Déployer des Web Services en production

- En production, il est important de ne pas mettre en œuvre un déploiement standalone (ie. *Endpoint.publish*)
 - Robustesse
 - Performances
 - Monitoring
 - Scalabilité
- Il est nécessaire de s'appuyer sur des conteneurs dédiés
 - Conteneur/Serveur JEE
 - Moteur OSGi

Déployer CXF dans un conteneur JEE

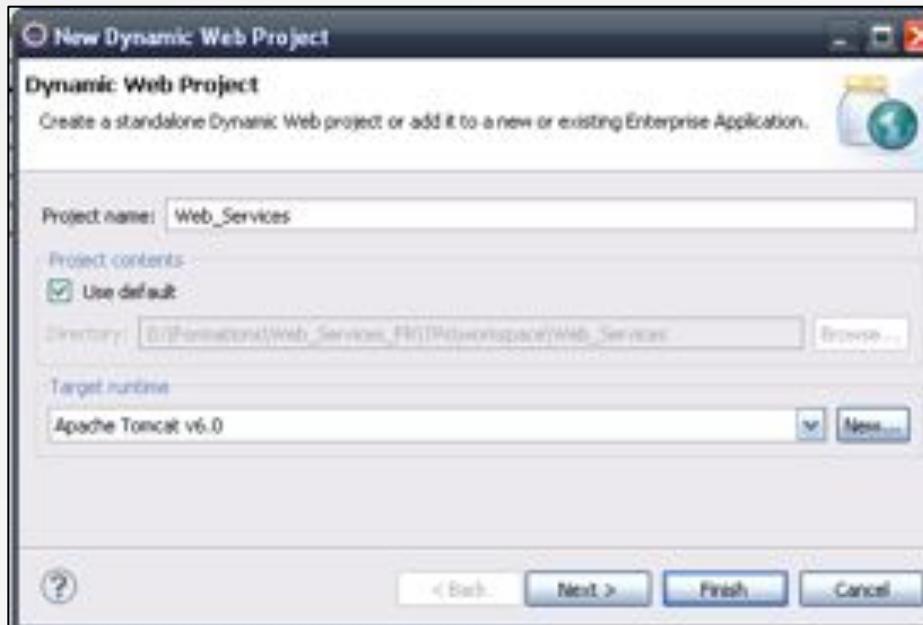


Créer une application CXF sous Eclipse (1/6)

1. Créer une application Web



2. Associer l'application Web à un conteneur/serveur JEE



Créer une application CXF sous Eclipse (2/6)

- Associer l'ensemble des bibliothèques CXF et leurs dépendances



- Configurer l'application
 - web.xml
 - cxf-servlet.xml

Créer une application CXF sous Eclipse (3/6)

- web.xml : Déclaration du point d'entrée des Web Services (ie. Servlet)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>cxf</display-name>
    <servlet>
        <servlet-name>cxf</servlet-name>
        <servlet-class>
            org.apache.cxf.transport.servlet.CXFServlet
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>cxf</servlet-name>
        <url-pattern>/services/*</url-pattern>
    </servlet-mapping>
</web-app>
```

web.xml

Créer une application CXF sous Eclipse (4/6)

- cxn-servlet.xml : Déclaration des endpoints Web Services

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

    <jaxws:endpoint
        id="welcomeService"
        implementor="com.resanet.ws.WelcomeServiceImpl"
        address="/welcome" />

</beans>
```

CXI-
servlet.xml

Créer une application CXF sous Eclipse (6/6)

- Développer les Web Services
- Déployer l'application sur le serveur cible



- Démarrer le serveur



- Accéder au Web Service via l'adresse du WSDL

`http://localhost:8080/[NOM_PROJET]/services/[ENDPOINT_WS]?wsdl`



WebServices REST

La naissance de REST

- Terme inventé par Roy Fielding en 2000 pour sa thèse
- Part du constat que le WEB est partout, performant et scalable
- Comment amener ces principes dans une architecture d'application ?
- Principalement grâce au protocole HTTP
- Créer un principe qui s'appuie sur les mêmes outils

- REST est un type d'architecture
 - S'appuyant sur les principes fondateurs du web
 - Orientée Ressources
- REST n'est pas
 - Un protocole
 - Une technologie
- REST = REpresentational State Transfer

Adressabilité

- Tout est une ressource
- Chaque ressource possède un identifiant unique

scheme://host:port/path?queryString#fragment

- Définit une adresse unique sur le réseau (URI)
- L'adresse est paramétrable
- Il est primordial de bien définir les URIs

Interface uniforme

- REST utilise 4 opérations pour manipuler les ressources
- Une API restreinte mais suffisante
- L'interopérabilité est facilitée : API connue et maîtrisée
- Les opérations ont un comportement prévisible

Représentation

- Une ressource peut être représentée par plusieurs formats de données
 - XML
 - JSON
 - YAML
 - autres...
- Le meilleur format est celui qui est le plus adapté au client
- Le client peut négocier la représentation des données avec le serveur.

Stateless

- Stateless ne veut pas dire qu'il n'y pas d'état
- L'état est porté par la ressource, pas par le serveur
- Les actions sont répétables et prévisibles
- Pas de session côté serveur, donc pas de réPLICATION
- Le client est responsable de la session et des transitions

HATEOAS

- **Hypermedia As The Engine Of Application State**
- Fournir des liens pour naviguer entre les états d'une ressource
- Pour le client, cela permet de découpler une action d'une implémentation
- Le client reçoit une liste de transitions possibles

- CRUD
 - Create / PUT
 - Read / GET
 - Update / POST
 - Delete / DELETE
- Suffisant pour stocker et/ou gérer les états d'une ressource

- Ce dit d'une application qui respecte l'ensemble des principes REST
- Pas toujours facile à respecter, la conception est déterminante
- REST n'est pas forcément RESTful
- Attention aux écueils de trop vouloir appliquer la règle !
- Il n'est pas toujours souhaitable d'appliquer la règle à la lettre
 - Des compromis peuvent être salutaires...

Les implémentations

- JSR 311 : LA spécification (<http://jsr311.java.net/>)
- Jersey : l'implémentation de référence
- Restlet : la plus ancienne
- RESTeasy : Jboss
- CXF : Apache foundation

- Utiliser l'API sans se soucier de l'implémentation
- Package javax.ws.rs
- Configuration des beans avec des annotations
- Sérialisation automatique, personnalisable, extensible
- Utilitaire pour construire la réponse : ResponseBuilder

Le service

- Gestion de livres
- URI principale de la ressource : /book
- Commandes :
 - Lister tous les livres
 - Trouver un livre
 - Ajouter un livre
 - Modifier un livre
 - Supprimer un livre

- Une ressource = un POJO JAVA
- Endpoint = Une classe JAVA qui délègue au manager
- Convention : BookResource = Classe qui gère la ressource
- `@Path("/path")`
 - Sur la classe : définit l'URI de base des commandes
 - Sur une méthode : ajoute à l'URI de base la localisation / paramétrage de la commande

```
@Path("/book")
public class BookResource {
}
```

`@Path`

Lister tous les livres

- URI : /book

@GET

@GET

```
public Response list() {
    return Response.ok(library.list()).build();
}
```

HTTP Request
GET /book HTTP/1.1

HTTP Response : 200 OK

```
<livres>
  <livre>
    <auteur>Albert Camus</auteur>
    <isbn>2070360024</isbn>
    <titre>L'étranger</titre>
  </livre>
  <livre>
    <auteur>Robert Louis Stevenson</auteur>
    <isbn>2253003689</isbn>
    <titre>L'île au trésor</titre>
  </livre>
</livres>
```

Trouver un livre

- URI : /book/2070360024

HTTP Request

GET /book/2070360024 HTTP/1.1

@GET

```
@GET
@Path("/{isbn}")
public Response findByISBN(final @PathParam("isbn") String isbn) {
    return Response.ok(library.find(isbn)).build();
}
```

HTTP Response : 200 OK

```
<livre>
  <auteur>Albert Camus</auteur>
  <isbn>2070360024</isbn>
  <titre>L'étranger</titre>
</livre>
```

Ajouter un livre

- URI : /book

HTTP Request

```
PUT /book HTTP/1.1
Content-Type: application/xml; charset=UTF-8

<livre>
  <auteur>Albert Camus</auteur>
  <isbn>2070360024</isbn>
  <titre>L'étranger</titre>
</livre>
```

@PUT

```
@PUT
public Response create(final Book book) {
    return Response.created(library.create(book)).build();
}
```

HTTP Response : 201 Created

```
Location: http://api.zenika.com/book/2070360024
```

Modifier un livre

- URI : /book

HTTP Request

```
POST /book HTTP/1.1
Content-Type: application/xml; charset=UTF-8

<livre>
  <auteur>Albert Camus</auteur>
  <isbn>2070360024</isbn>
  <titre>L'étranger (version originale)</titre>
</livre>
```

@POST

```
@POST
public Response update(final Book book) {
    return Response.ok(library.update(book)).build();
}
```

HTTP Response : 200 OK

```
Content-Location: http://api.zenika.com/book/2070360024
Content-Type : application/xml; charset=UTF-8

<livre>
  ...
```

Supprimer un livre

- URI : /book/2070360024

HTTP Request

DELETE /book/2070360024 HTTP/1.1

@DELETE

```
@DELETE
@Path("/{isbn}")
public Response delete(final @PathParam("isbn") String isbn) {
    library.delete(isbn);
    return Response.ok().build();
}
```

HTTP Response : 204 No Content



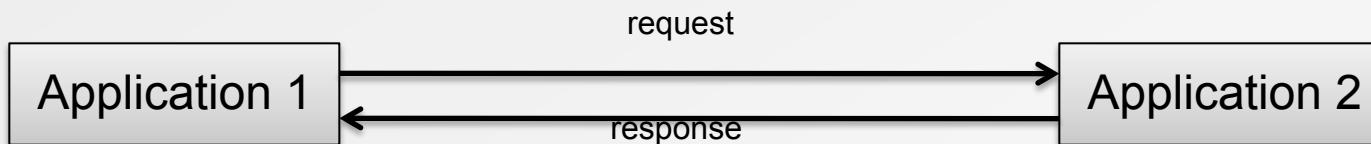
Messages Oriented Middleware

- L'API JMS
- Le standard AMQP
- Messaging AMQP / JMS
- L'approche Message-Driven (ou Event-Driven)
- Transaction, intégrité et fiabilité

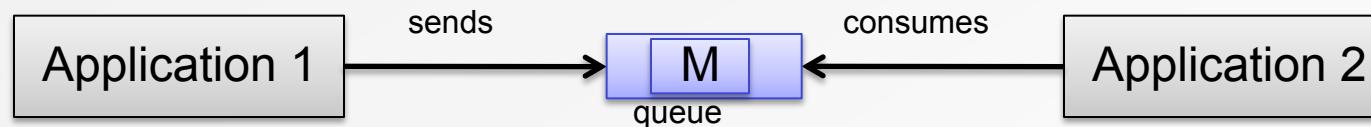
- Les MOM peuvent véhiculer n'importe quel type de messages
 - Texte
 - XML
 - *String*
 - Binaire
 - *Objet*
 - *Byte*
- Les messages contiennent des headers
 - Métiers
 - Techniques

Synchrone VS Asynchrone

- Echange synchrone (non supporté sur les MOM nativement)



- Echange asynchrone



Messaging et couplage

- Producer
 - Celui qui produit le message
- Consumer
 - Celui qui consomme le message
- Les brokers découplent totalement les producers des consumers
 - Pas besoin d'être côté à côté
 - Pas besoin d'une réponse immédiate (potentiellement jamais)
 - Ils ne se connaissent pas
 - N'ont pas besoin d'être implémenter avec la même technologie (Java, Cobol)



JMS

L'API JMS (Java Message Service)

- JMS est une API JAVA
 - Ensemble d'interfaces
 - Ensemble de principes de fonctionnement du MOM
- Définition standard Java et abstraction du broker de messages
 - Très pratique pour changer d'implémentation / driver JMS

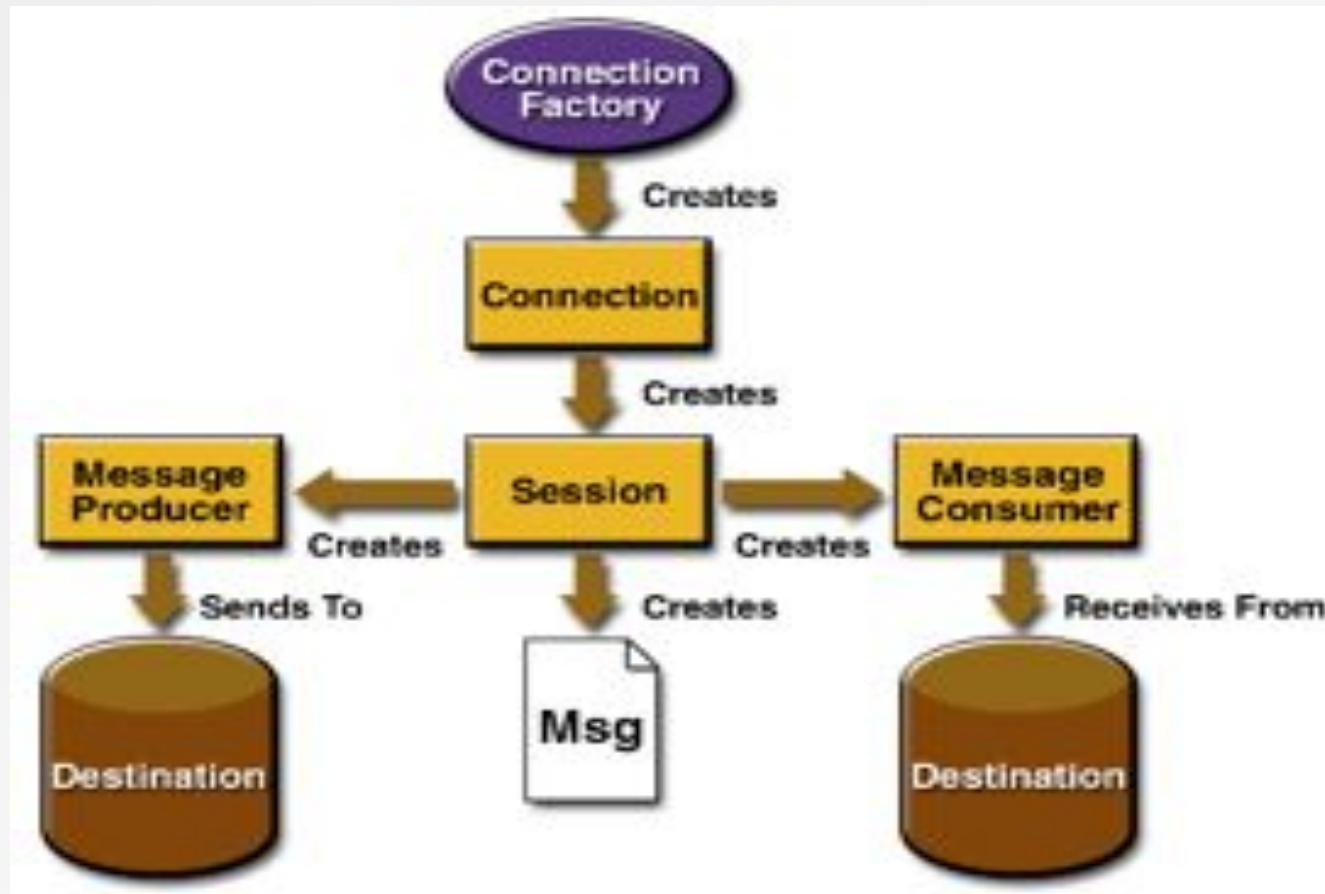
- Les messages sont stockés sur des files de messages
 - Queue
 - Topic
- La queue
 - Un seul consommateur reçoit le message
- Le topic (publish / subscribe)
 - Tous les consommateurs connectés reçoivent une copie du message

L'API JMS (Java Message Service)

- JMS est une API JAVA
 - Ensemble d'interfaces
- Définition standard Java et abstraction du broker de messages
 - Très pratique pour changer d'implémentation / driver JMS
- Elle spécifie au niveau développement les conventions d'accès
 - Au broker
 - Au message
 - À la connexion

L'API JMS (Java Message Service)

- API « old school »



Envoi d'un message JMS

- Récupération d'une ConnectionFactory JMS
 - Généralement via un lookup ou bean Spring

```
@Resource  
private ConnectionFactory connectionFactory;
```

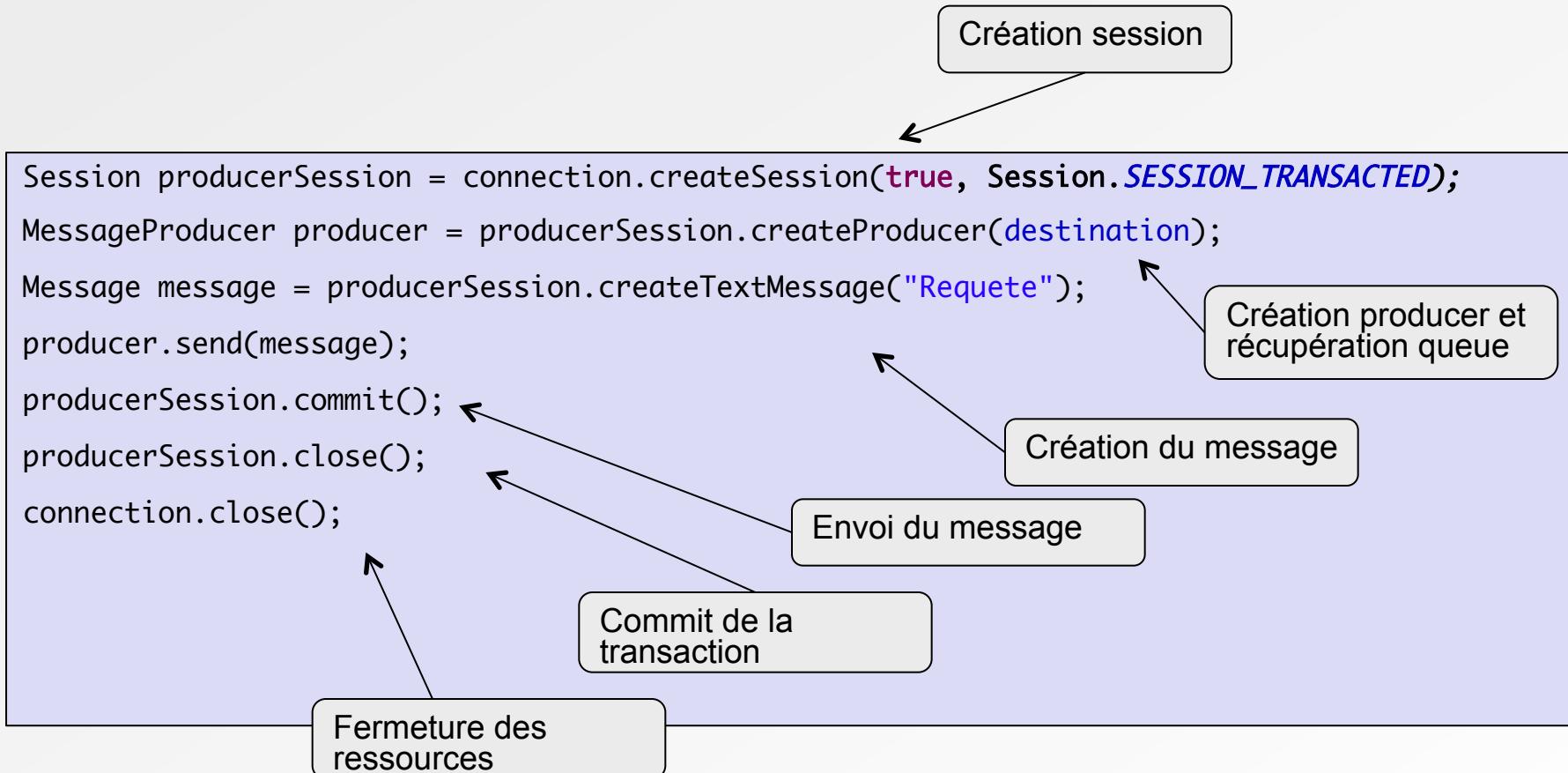
- Récupération de la queue / topic
 - Généralement via un lookup ou bean Spring

```
@Resource  
private Destination destination;
```

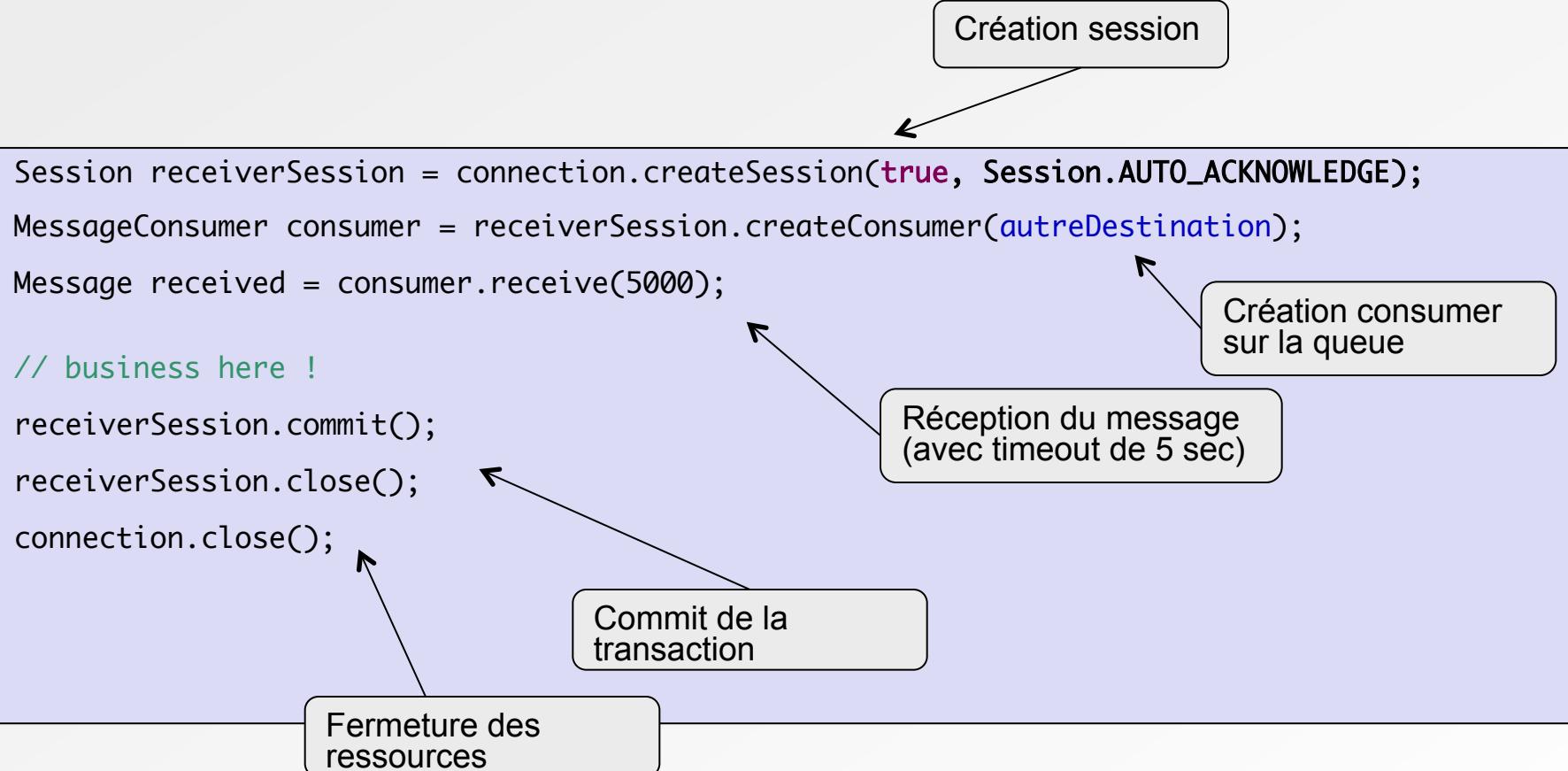
- Création et établissement d'une Connection

```
Connection connection = connectionFactory.createConnection();  
connection.start();
```

Envoi d'un message JMS



Réception d'un message JMS





AMQP

Les standards des MOM

- Jusqu'en 2012 (!), aucun standard pour les MOM
 - Il existait bien JMS, mais ce n'est pas un standard

- En 2012 est sorti la version 1.0 du standard AMQP
- Pourquoi AMQP ?
 - Besoin d'intégrer des systèmes hétérogènes avec les propriétés des MOM
 - Se baser sur un protocole standard et ouvert
 - Le langage et l'implémentation n'importe pas



asynchrone

synchrone

SMTP

HTTP

?

IOP

unreliable

reliable

AMQP – Advanced Message Queuing Protocol

- En 2012 est sorti la version 1.0 du standard AMQP
- Pourquoi AMQP ?
 - Besoin d'intégrer des systèmes hétérogènes avec les propriétés des MOM
 - Se baser sur un protocole standard et ouvert
 - Le langage et l'implémentation n'importe pas

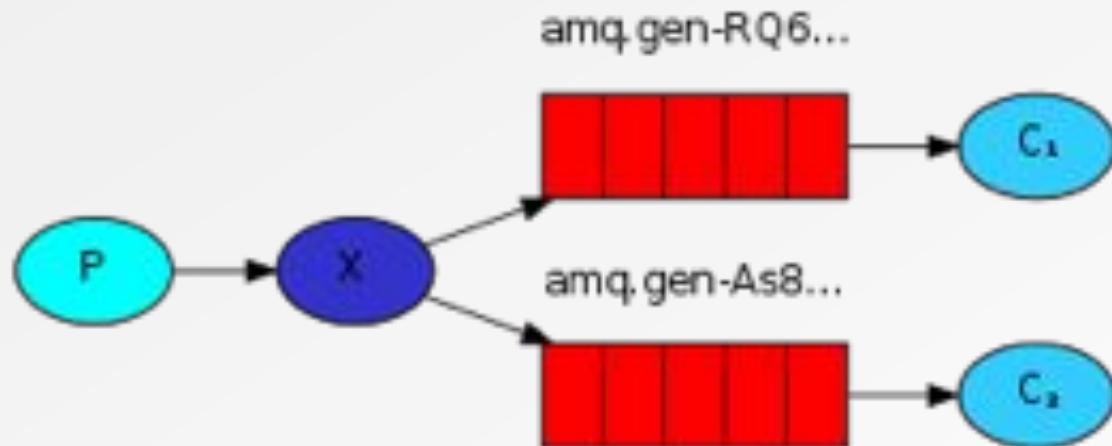
AMQP – Advanced Message Queuing Protocol

AMQP != JMS

RabbitMQ – Broker AMQP performant

- Broker de messages multi-protocoles
 - AMQP, SMTP, STOMP, XMPP, etc.
- Construit autour du protocole AMQP
- Des « bindings » dans la plupart des langages
 - Java, .NET, Python, JavaScript, Ruby, PHP, etc.

Le fonctionnement d'AMQP

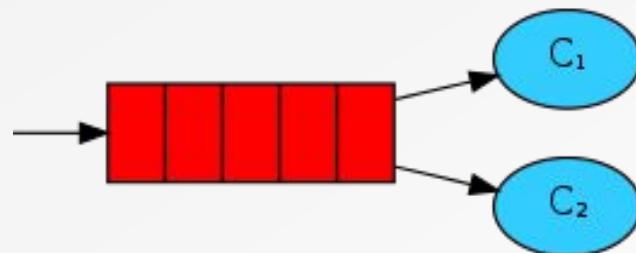


Le fonctionnement d'AMQP

- Queue
 - Les messages sont consommés à partir des *queues*
 - Les messages sont stockés dans les *queues*
 - Les *queues* sont FIFO
- Exchange
 - Les messages ne sont pas directement envoyés sur les *queues*
 - Ils sont routés vers les *queues* au travers des *exchanges*
- Binding
 - Les *queues* sont connectées aux *exchanges* grâce aux *bindings*
 - Les *bindings* se font grâce à des *patterns*

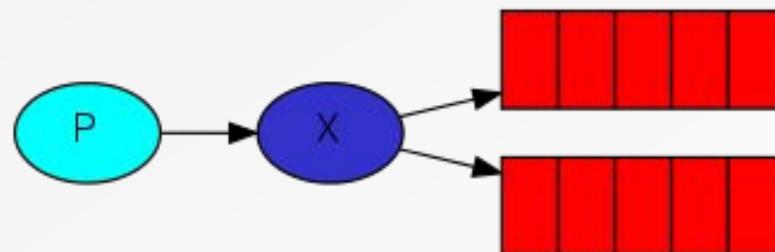
Le fonctionnement d'AMQP

- Consumer
 - Les *consumers* sont directement connectés aux *queues*
 - Plusieurs *consumers* peuvent être connectés à la même *queue*
 - Dans ce cas, le message n'est délivré qu'à un seul *consumer*
 - Le mode de dispatch est round-robin



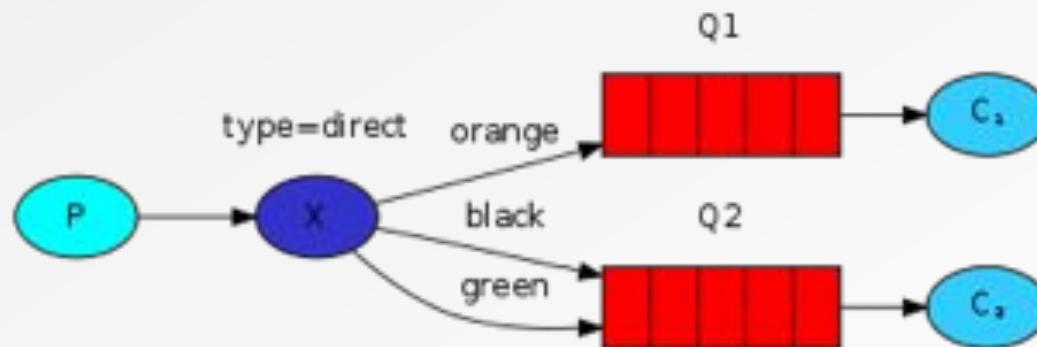
Le fonctionnement d'AMQP

- Producer
 - Les *producers* ne sont pas connectés au *queue*
 - Ils sont connectés aux *exchanges*
 - Plusieurs *producers* peuvent être connectés au même *exchange*



Le fonctionnement d'AMQP

- Le routage
 - Les *messages* sont publiés avec une *routing-key*
 - Les messages sont routés jusqu'aux *queues* par *matching* entre la *routing-key* et les patterns des *bindings*



Le fonctionnement d'AMQP

- Les différents types d'Exchange
 - fanout
 - Pas de pattern, pas de routing-key, le lien est direct
 - direct
 - le pattern du binding est simplement le nom de la queue
 - topic
 - le pattern du binding est une expression qui vérifie le nom de la queue (pattern matching)
 - headers
 - Pattern appliqué sur un header défini

Le fonctionnement d'AMQP

- Attention
 - Le topic en AMQP n'est pas un publish / subscribe !
 - Uniquement un type de routage
- Pour faire un mode pub/sub en AMQP
 - Chaque consumer doit avoir sa propre queue

Envoyer un message AMQP

- Il n'y pas d'API autour d'AMQP, les codes sont donc dépendants des implémentations et des API des drivers des brokers
- Création de la connexion AMQP vers le broker RabbitMQ

```
ConnectionFactory factory = new ConnectionFactory();
factory.setUsername("guest");
factory.setPassword("guest");
factory.setVirtualHost("/");
factory.setHost("localhost");
factory.setPort(5672);
Connection connection = factory.newConnection();
```

Envoyer un message AMQP

- Création du channel et envoi du message

```
Channel channel = connection.createChannel();
byte[] message = "message".getBytes();
channel.basicPublish("quotations", "nasq", null, message);
```

Nom de l'exchange

Routing-key

Recevoir un message AMQP

- Création du channel et envoi du message

```
GetResponse response = channel.basicGet("quotations", true);
```

Nom de la queue

Auto acknowledge flag

Message driven

- Jusqu'à présent, nous avons vu uniquement des consommateurs « actifs », qui sont en attente de réception de message bloquante
- Il est évidemment possible de réaliser une attente de façon asynchrone, et d'être notifié lorsqu'un message arrive sur la queue.

- Réception asynchrone JMS

```
import javax.jms.Message;
import javax.jms.MessageListener;

public class JMSMessageDriven implements MessageListener {
    public void onMessage(Message message) {
        // business here
    }
}
```

- Une instance (ou n instances) est ensuite enregistré sur la connexion, et peu ainsi être notifié de façon totalement asynchrone
- Peut aussi être enregistré dans le contexte Spring

AMQP Message Driven – via Spring AMQP

- Réception asynchrone AMQP (Spring AMQP)

```
Import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageListener;

public class AMQPMessageDriven implements MessageListener {
    public void onMessage(Message message) {
        // business here
    }
}
```

- Uniquement les imports Java changent (abstraction sur le broker)

Transactions

- Il est également possible de grouper des messages dans une même transaction
 - Plusieurs messages en mode consumer
 - Plusieurs messages en mode producer
 - Un mélange des deux !

Grâce aux transactions des MOM, on peut donc très simplement assurer une intégrité totale et une très grande fiabilité des traitements

→ les traitements critiques sont très souvent placés entre 1 paire de files

- Une des plus grandes forces des MOM est d'être transactionnel
- Au mettre titre que les opérations sur les bases de données, les opérations faites sur les brokers peuvent faire l'object d'un :
 - Commit (validation)
 - Rollback (annulation)
- On peut donc par exemple :
 - Lire un message sur une file en mode transactionnel
 - Faire un traitement
 - Puis finalement commiter le message

Si un erreur survient lors du traitement, le message non commité sera automatiquement remis sur la file

AMQP – Acknowledgement / Transaction

- En AMQP, on parle plus d'Acknowledgement plutôt que de transaction, bien que les 2 existent
- Le système d'acknwoledgement est plus souple et permet de grouper plusieurs messages ensemble (émission ou réception)
- Le système de transaction est utilisé lorsque l'on souhaite coupler dans la même transaction la réception et l'émission.

- Acknowledgement d'un message seul

```
boolean autoAck = false;
GetResponse response = channel.basicGet("queueName", autoAck);

if (response != null) {

    AMQP.BasicProperties props = response.getProps();
    byte[] body = response.getBody();
    long deliveryTag = response.getEnvelope().getDeliveryTag();

    // do business logic here

    boolean multiple = false;
    channel.basicAck(deliveryTag, multiple);

}
```

- Acknowledgement d'un groupe de messages

```
boolean autoAck = false;
GetResponse response1 = channel.basicGet("queueName", autoAck);
// do business with response1

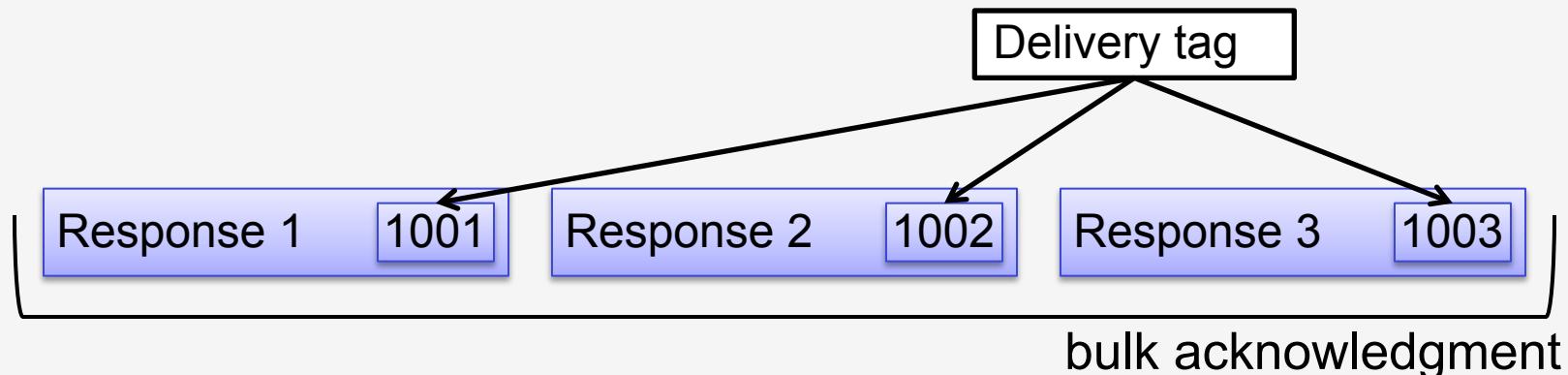
GetResponse response2 = channel.basicGet("queueName", autoAck);
// do business with response2

GetResponse response3 = channel.basicGet("queueName", autoAck);
// do business with response3

long latestDeliveryTag = response3.getEnvelope().getDeliveryTag();

channel.basicAck(latestDeliveryTag, true);
```

- Tous les messages qui
 - Sont reçus sur le même channel
 - Pas encore reçu d'acknowledge
 - Avec un DeliveryTag inférieur ou égal au DeliveryTag passé à la méthode basicAck()



```
channel.basicAck(1003, true);
```

Multiple acks flag



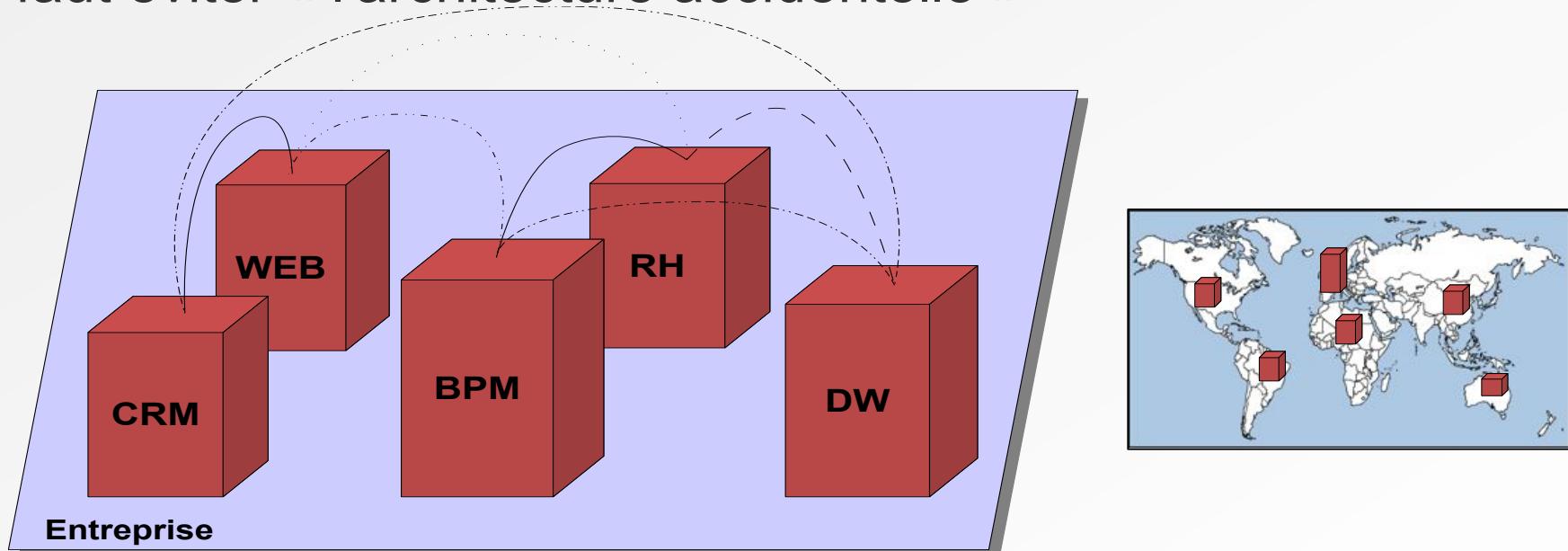
ESB

Enterprise Service Bus

- Comprendre l'utilité ESB
- Les EIP – Enterprise Integration Patterns
- La connectivité
- Le routage des messages
- Les transformations

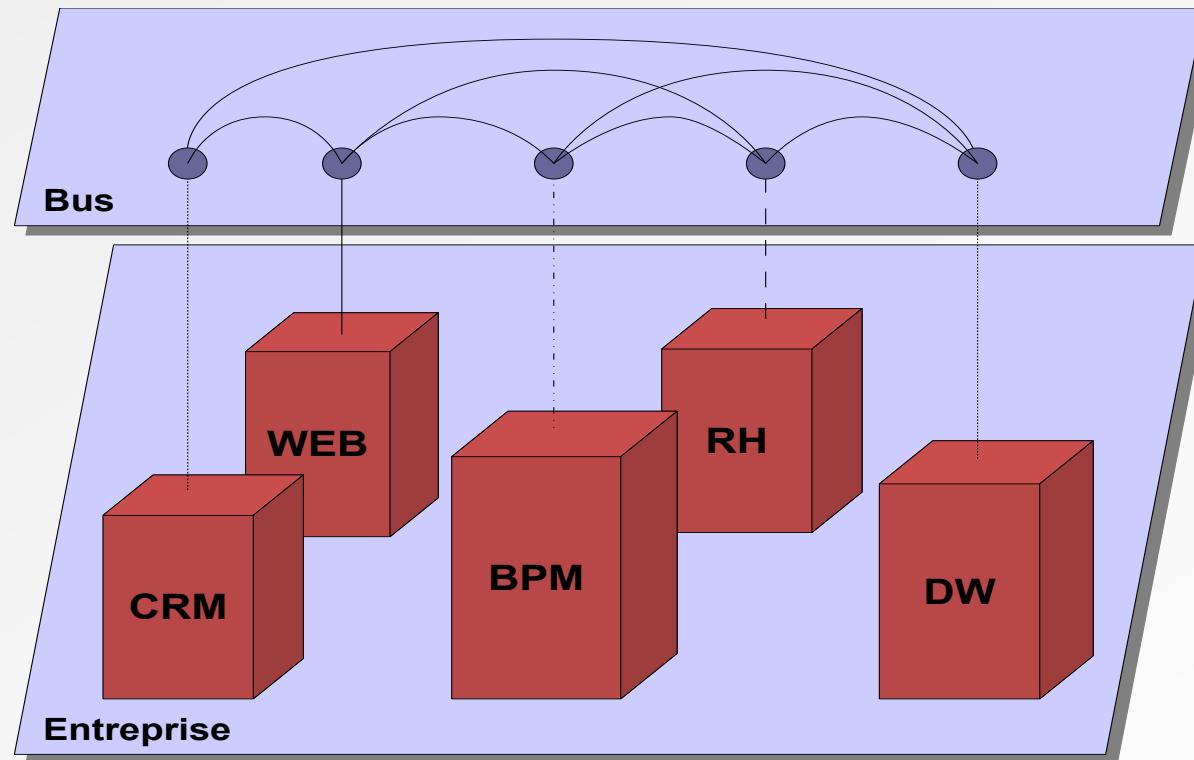
Communication entre des systèmes

- Les applications d'entreprise sont dispersées (géographiquement et logiquement)
- Les applications isolées sont inutiles
- Elles doivent communiquer entre elles par nécessité
- Il faut éviter « l'architecture accidentelle »



Communication entre des systèmes

- L'ESB permet d'adresser cette problématique

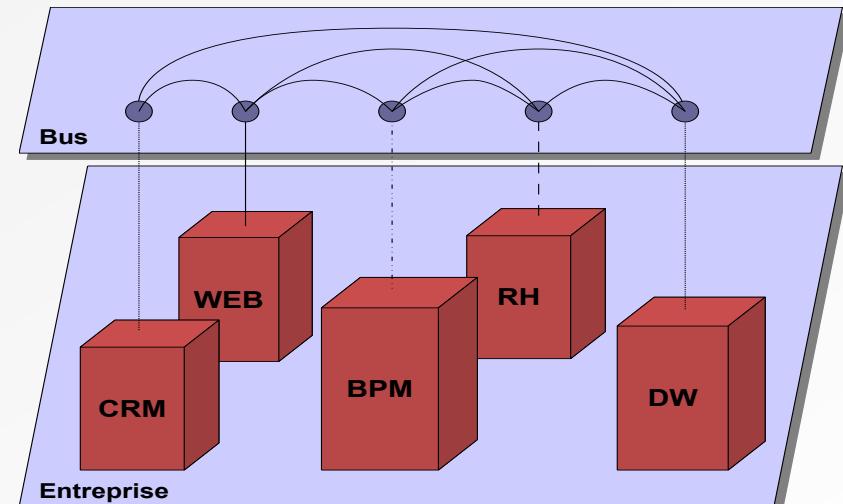
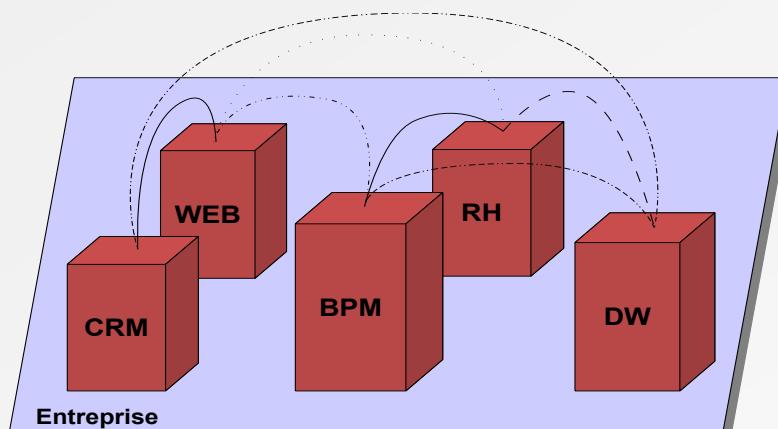


Couplage lâche (interactions)

- Le couplage lâche, aussi appelé couplage faible ou léger (*loose coupling*), se dit d'une interaction entre des composants logiciels où les parties se concentrent uniquement sur la production du message à transmettre
 - exemples : JMS, Webservices, Fichiers
- A l'inverse, dans une interaction à couplage fort, les parties connaissent intimement les détails de l'interface à appeler
 - exemples : RPC, RMI

Couplage lâche (interactions)

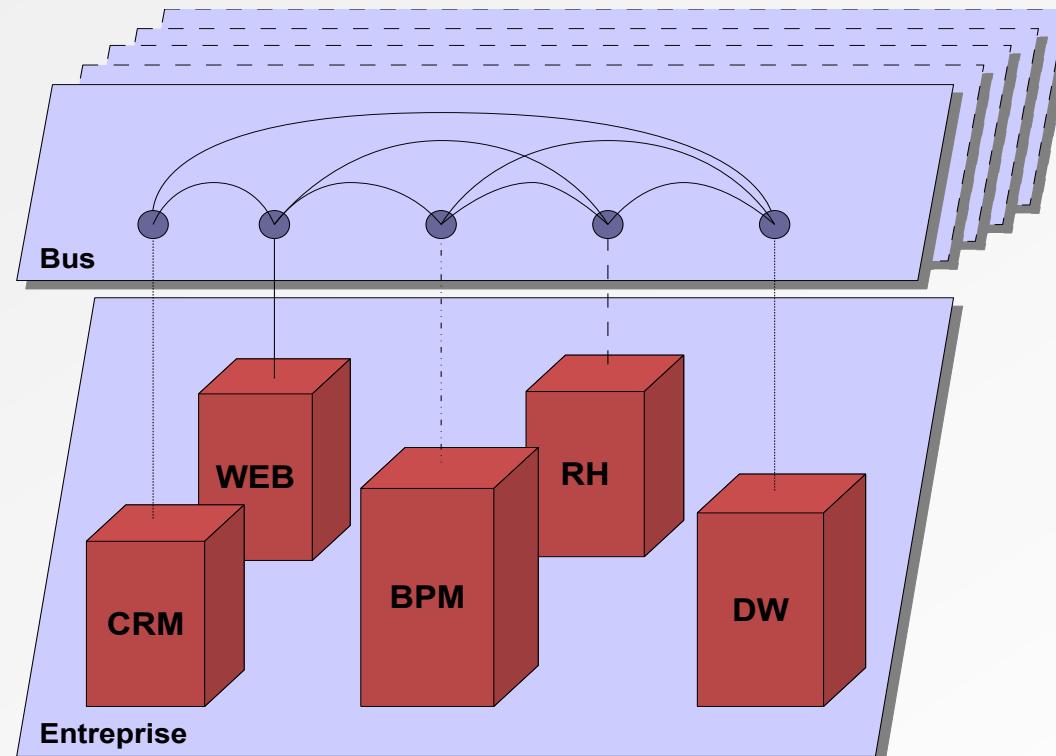
- Les ESB nous permettent de découpler les applications en fournissant une couche d'abstraction
- Les échanges de messages ne se font plus directement entre applications, c'est désormais l'ESB qui fait le médiateur



- Du « plat de spaghetti » vers l'intégration d'un ESB

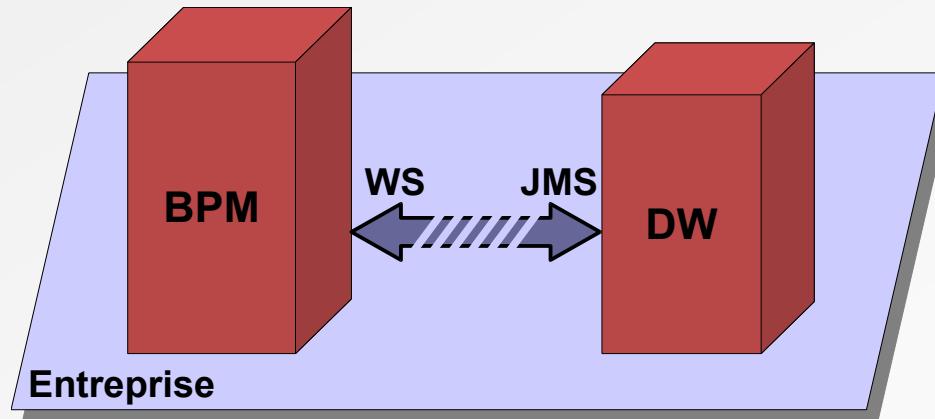
Distributivité

- L'ESB est hautement distribué par définition
- Ne pas reproduire les erreurs des EAI (centralisé et monolithique)



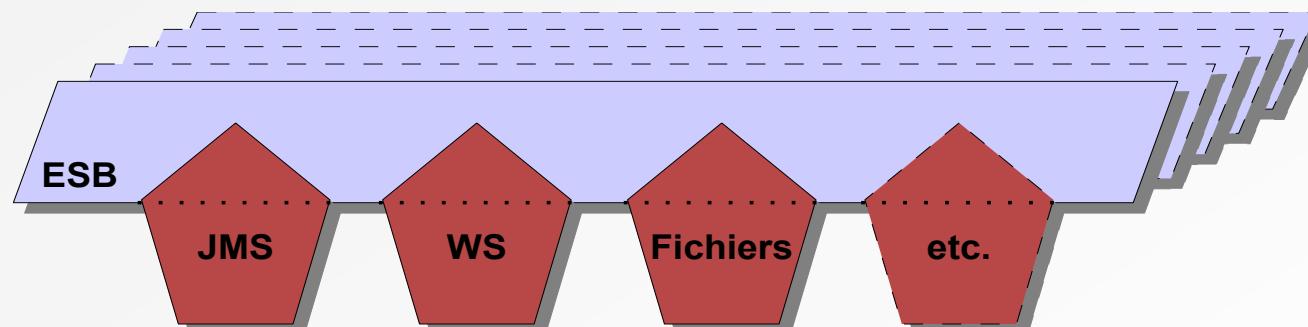
- La distributivité apporte
 - Scalabilité (ajout de nœuds)
 - Fiabilité (failover, load-balancing)
 - Facilité d'intégration (« si vous ne pouvez pas amener l'application au bus, amenez le bus à l'application »)

- Les applications se comprennent rarement entre-elles
- Une multitude de technologies / protocoles différents utilisés par les applications d'entreprise
 - JMS, Webservices, SMTP, Fichiers, JDBC, EJB, etc.



- Relier des protocoles et des applications hétérogènes
- Intégration de nouvelles applications facilitée

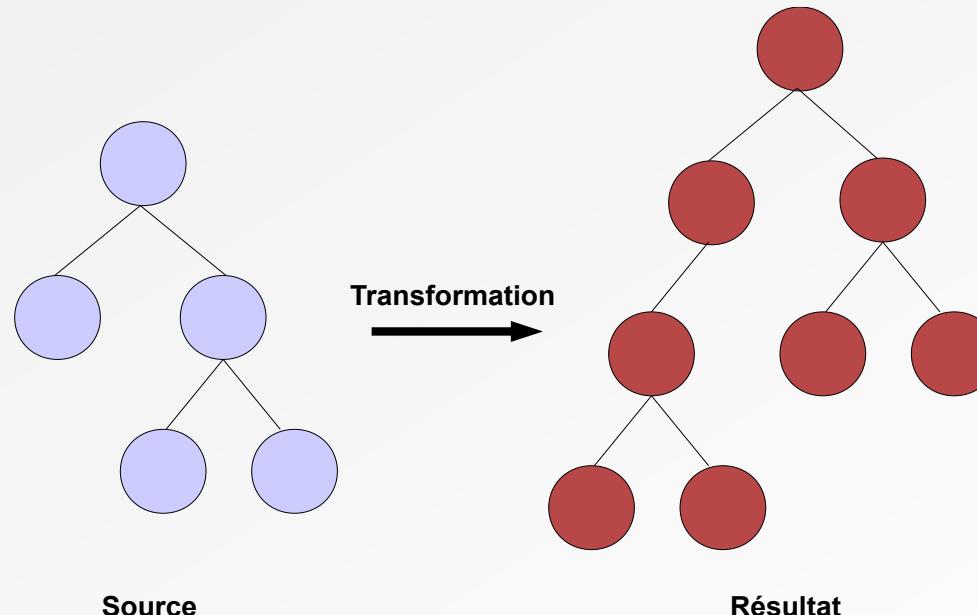
- L'ESB est capable de faire communiquer des applications utilisant des protocoles différents via les adaptateurs (ou connecteurs)
- Extensibilité
 - Installation d'adaptateurs tiers
 - Création d'adaptateurs spécifiques



Transformation et enrichissement

- Transformation

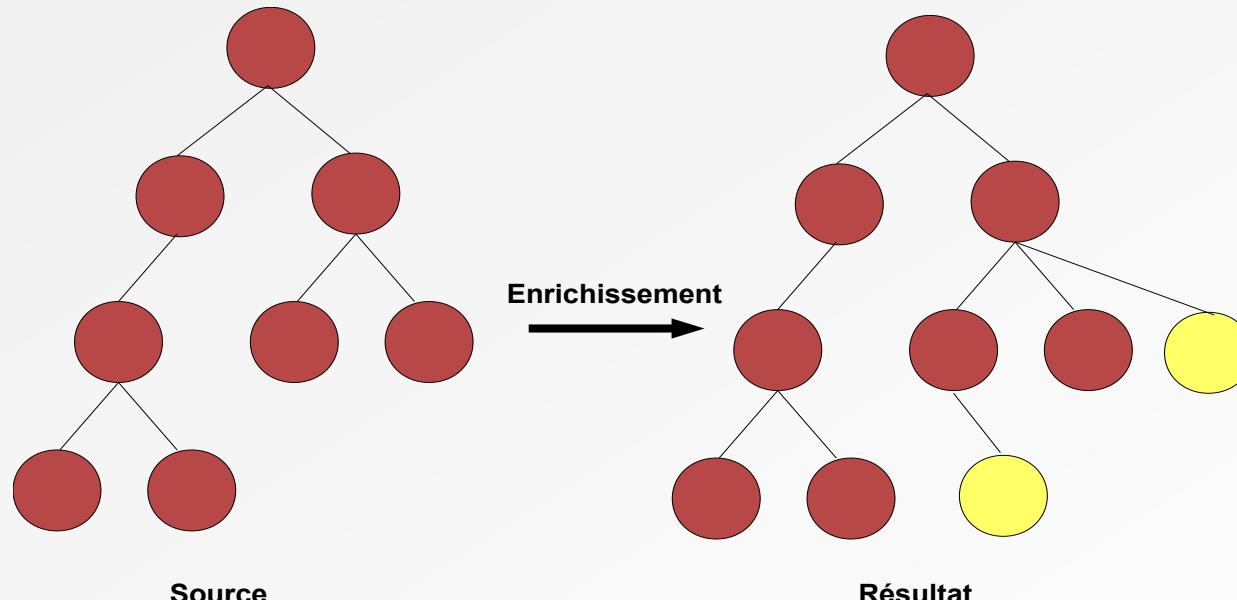
- Les applications ne travaillent pas avec un format commun
- Convertir un message initial vers un format cible
- Transformation de données
- XSLT (eXtensible Stylesheet Language Transformation)



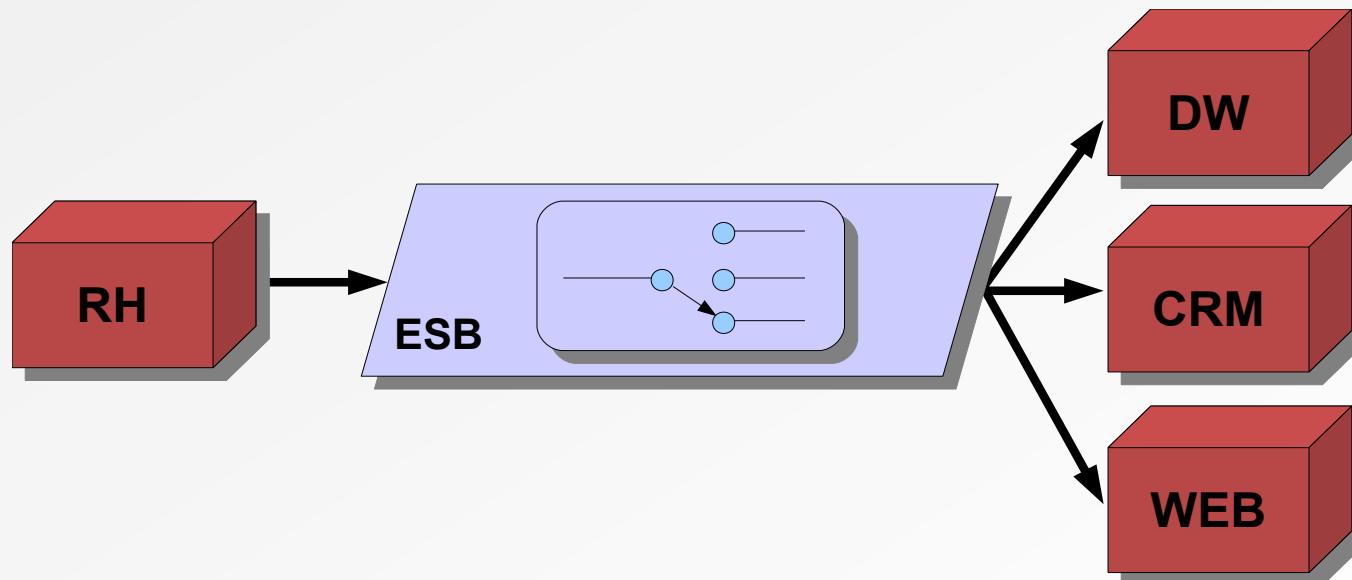
Transformation et enrichissement

- Enrichissement

- Les messages qui transitent ne contiennent pas toutes les données utiles
- Ajouter des données au message
- Collecte de données d'un système tiers (JDBC, Webservices, etc.)



- Logique métier qui détermine la destination finale du message
- Acheminer le message en fonction de critères métiers
- Aiguillage intelligent
- Possibilité de router le message vers plusieurs applications en parallèle ou de façon séquentiel



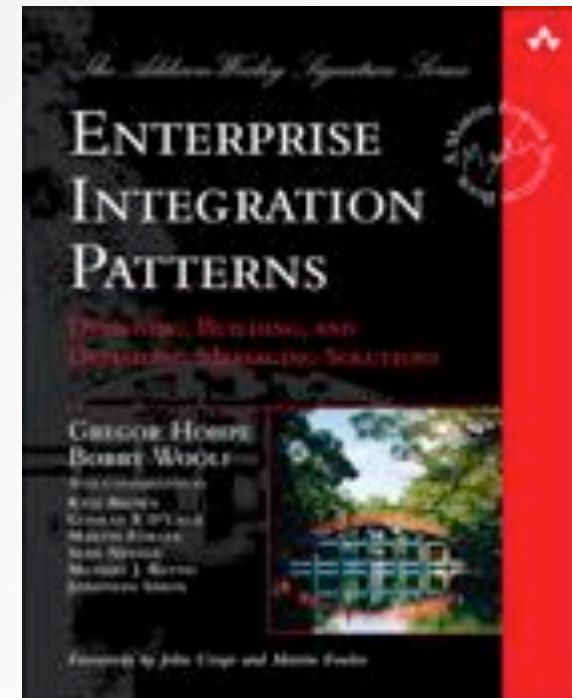
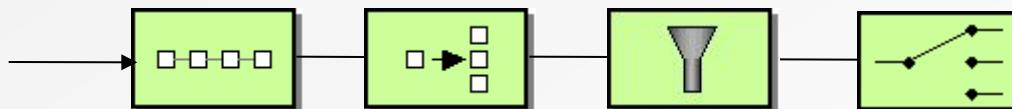
- Intégrité
 - Un ESB doit être fiable
 - Utilisation d'un serveur de messages intrinsèque
 - Application du pattern « *send and forget* »
 - L'ESB garantit que le message arrivera à destination
 - Système de gestion d'erreurs
- Sécurité
 - L'ESB est au cœur du SI, il est donc capital de le protéger
 - Gestion des autorisations via de l'authentification
 - Sécuriser les échanges entre applications
 - Sécuriser les accès des utilisateurs



Entrepri**e** Integration Patterns

- **Enterprise Integration Pattern**

- Auteurs *Gregor Hohpe & Bobby Woolf*
- Design Pattern communs pour résoudre des problèmes connus
- Réponse aux problématiques d'intégration des applications
- Architecture orientée messages



- Chaque EIP répond à un besoin particulier
- Vocabulaire :

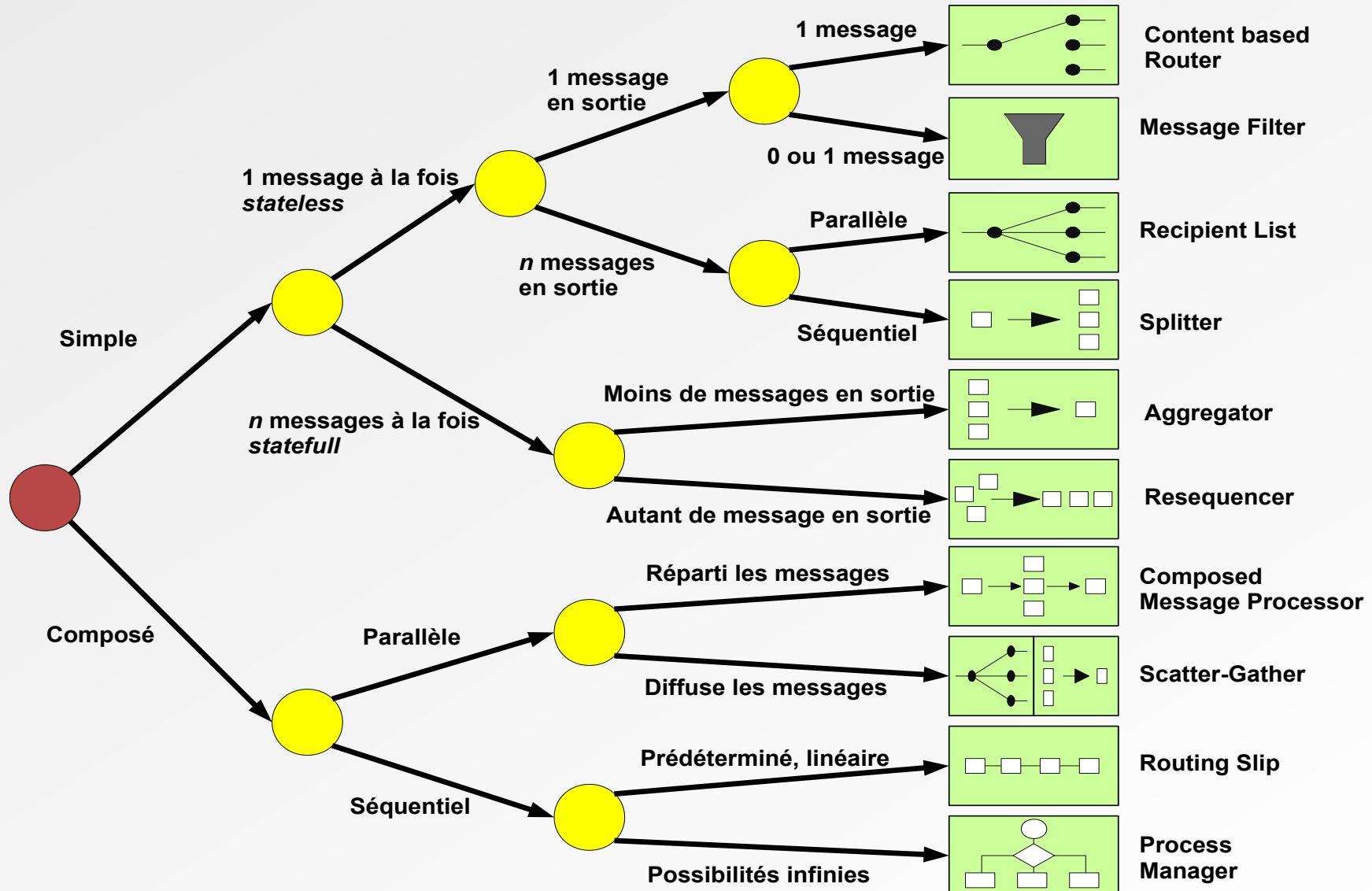
« La succession d'EIP définit une **Route** entre 2 **Endpoints** (un **producteur** et un **consommateur**) »
- La composition des EIP entre eux permet de faire émerger de nouveaux Patterns
- Les 50 EIP permettent de modéliser des solutions à l'ensemble des problématiques d'intégration au travers d'un langage graphique commun

- Les Entreprise Integration Patterns relatifs au routage
 - Routeurs simples
 - *1 canal d'entrée*
 - *N canaux de sorties*
 - *Routage simple*
 - Routeurs composés
 - *Combinaison de plusieurs routeurs simples*
 - *Création de routeurs complexes*
 - *Flux de messages complexes*

- Un des plus gros défi pour un ESB est le routage
 - Quelle est la destination finale du message ?
 - Comment acheminer le message à sa destination finale ?
 - Sur quels critères se baser ?
- La destination est un Endpoint (qui peut être associée à un Service et à une interface)
 - Endpoint physique (fichier, webservices, file JMS)
 - Endpoint logique (*direct:destination*, endpoint interne JBI)

- Aparté sur les EIP relatifs à la transmission de messages
 - Filters (au sens architectural)
 - *Composant qui s'interface entre 2 Pipes*
 - *Ne rempli pas obligatoirement le rôle le filtre*
 - Pipes
 - *Lien entre 2 ou n composants*
 - *Peut avoir plusieurs modes de fonctionnement (pipeline, multicast)*

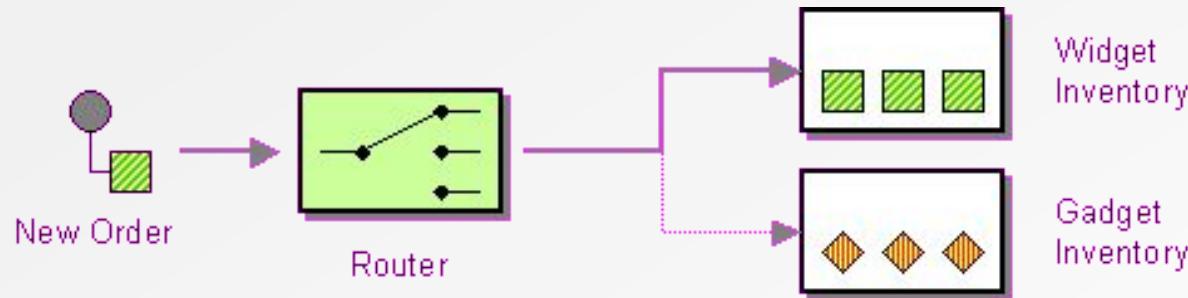
Introduction aux EIP de routage



- Fonctionnalités des EIP dédiés au routage de messages
 - Routage basé sur le contenu
 - Filtrage de messages
 - Construire dynamiquement la liste des destinataires
 - Segmenter des messages en n parties
 - Agréger n messages en un
 - Réordonner un groupe de messages
 - Processeur de messages composés
 - Dispersion – Ré-assemblage
 - Plan de routage
 - Gestion des processus métiers

- Régulateur de débit
- Temporisateur
- Répartiteur de charge
- Diffusion vers n récepteurs
- Répétition

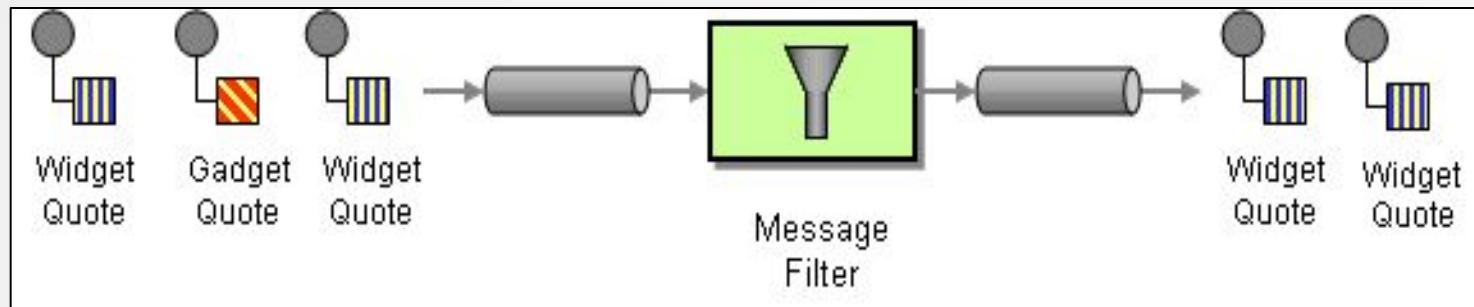
- Content Based Router
 - Comment traiter une situation où l'implémentation d'une seule fonctionnalité est éparpillée sur plusieurs systèmes physiques ?



- Routage basé sur le contenu
- Examine le contenu du message (headers ou body)
 - Existence de champs
 - Valeur de champs

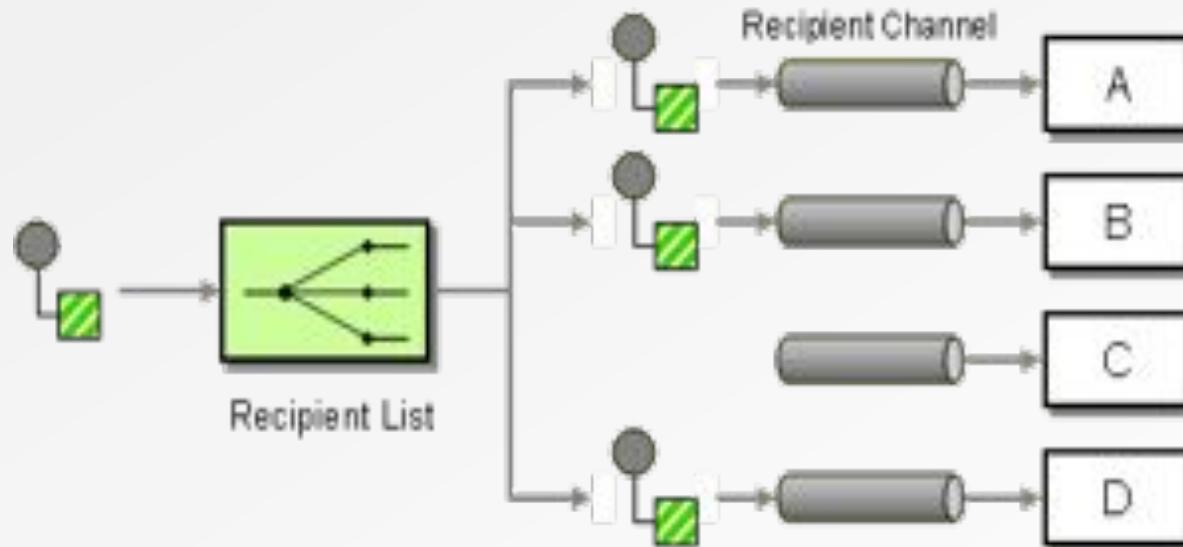
EIP – Message Filter

- Message Filter
 - Comment éviter qu'un composant reçoive des messages qui ne l'intéressent pas ?



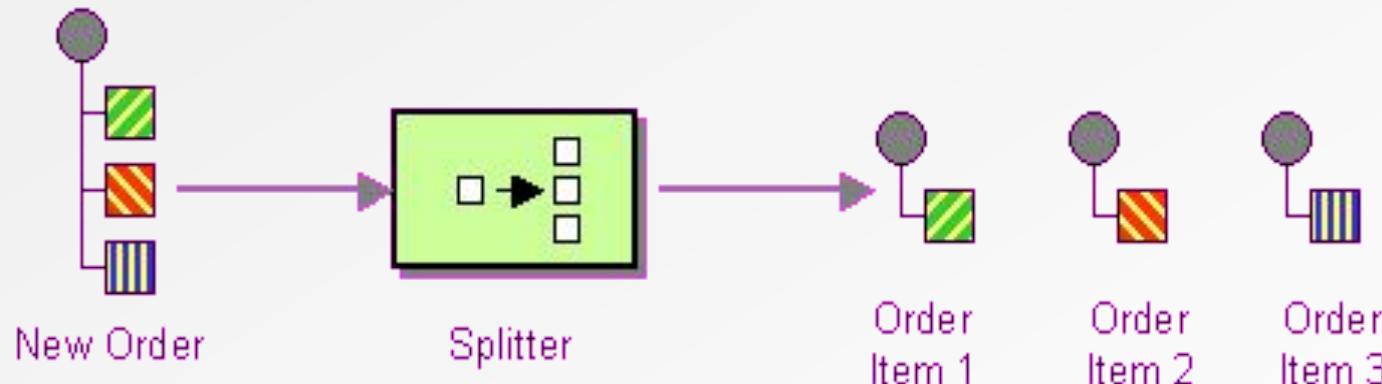
- Filtre les messages pour n'en laisser passer que certains selon un ensemble de critères configurables
- Permet d'éviter de surcharger un système
- Épure les flux de messages

- Recipient List
 - Comment router un message vers une liste (statique ou dynamique) de destinataires ?



- Recipient List
 - Deux axes dans cet EIP
 - *Calculer la liste des destinataires*
 - *Router une copie du message entrant vers chacune des destinations*
 - Les destinations sont calculées à partir du message d'entrée

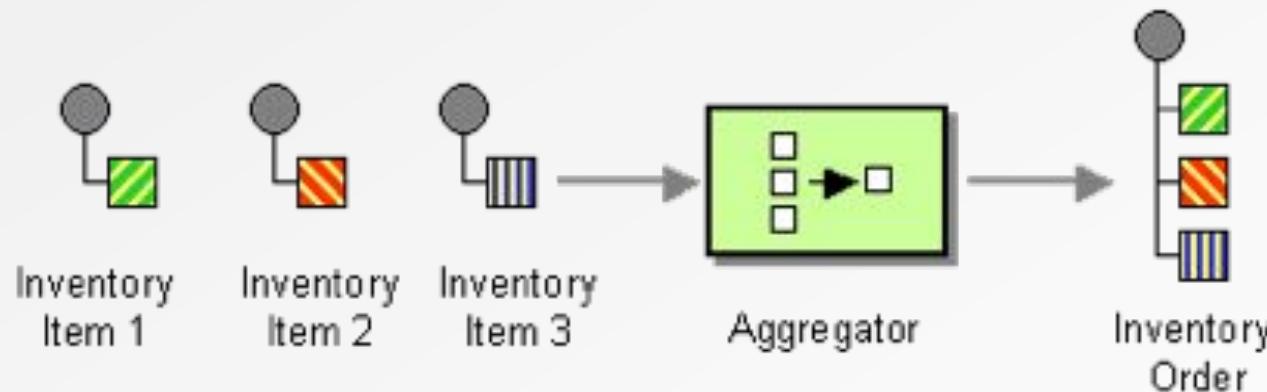
- Splitter
 - Comment traiter un message contenant plusieurs parties, chacune d'entre elles pouvant être traitées différemment ?



- Splitter
 - Casse le message d'origine en une série de messages individuels
 - Chaque nouveau message contient une parties des données d'origine
 - Lorsque couplé à un Content Based Router, permet de n'envoyer que les données utiles à un service
 - Il peut être intéressant de conserver un identifiant de corrélation entre les messages splittés

EIP – Aggregator

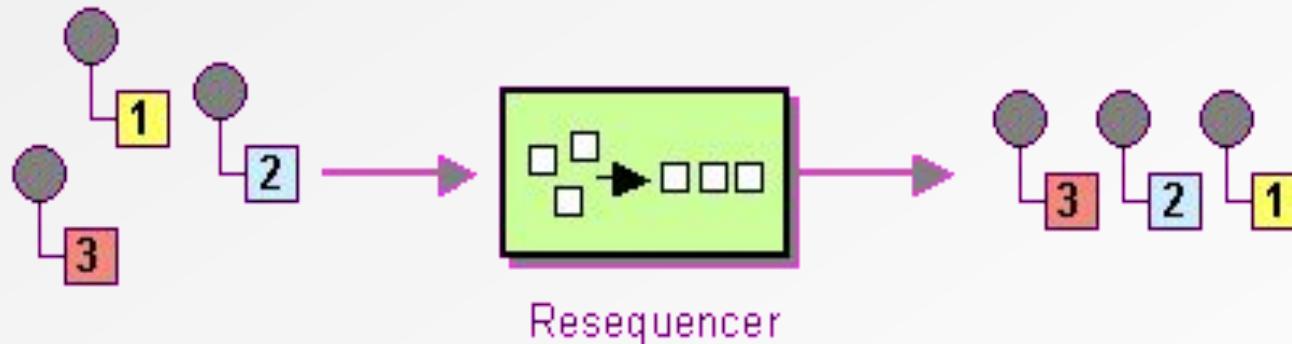
- Aggregator
 - Comment pouvons nous combiner des messages individuels (bien que liés) pour n'en former qu'un seul ?



- Aggregator
 - Combinaison des messages individuels pour ne former qu'un seul message
 - Collection des messages individuels
 - Stockage des messages individuels
 - Critères de liaisons des messages (identifiant de corrélation, header, valeur calculée)

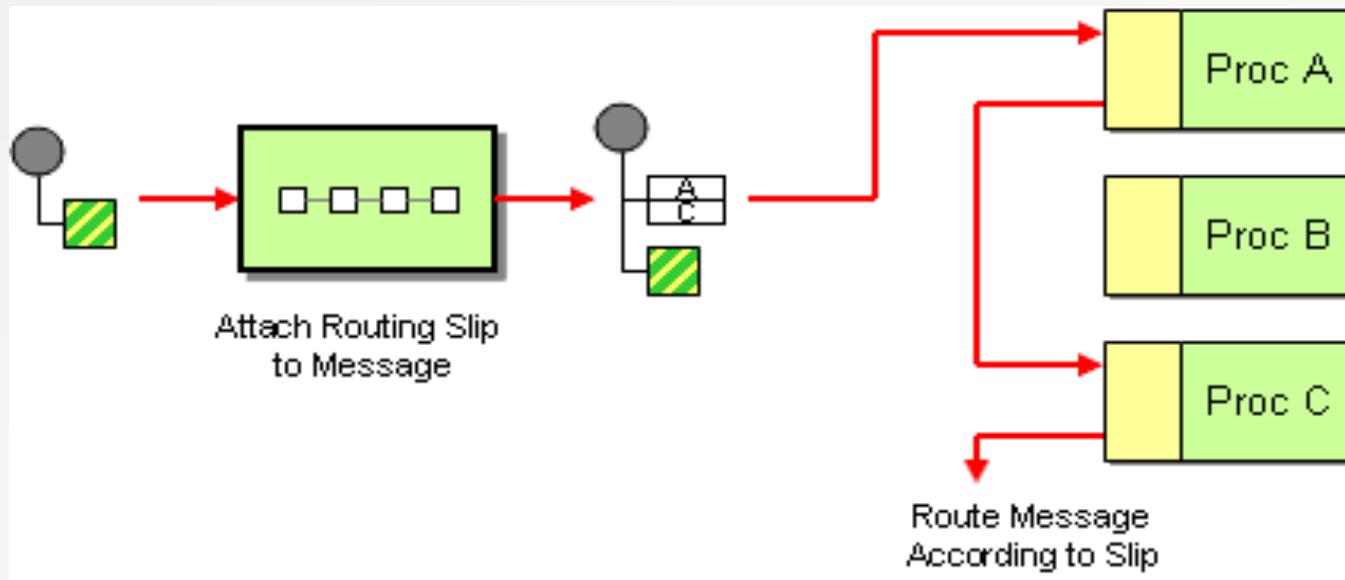
- Aggregator
 - Critères de complétion de la collection
 - *Quand publie-t-on le message résultant ?*
 - *Nombre de messages reçus*
 - *Délai d'attente*
 - Algorithme d'agrégation
 - *Comment réunit-on les messages entrants en un seul message ?*
 - EIP statefull
 - *Stockage des messages individuels en attendant que l'intégralité des messages soit collectée*

- Resequencer
 - Comment pouvons nous réordonner des messages ayant un lien mais arrivés aléatoirement ?



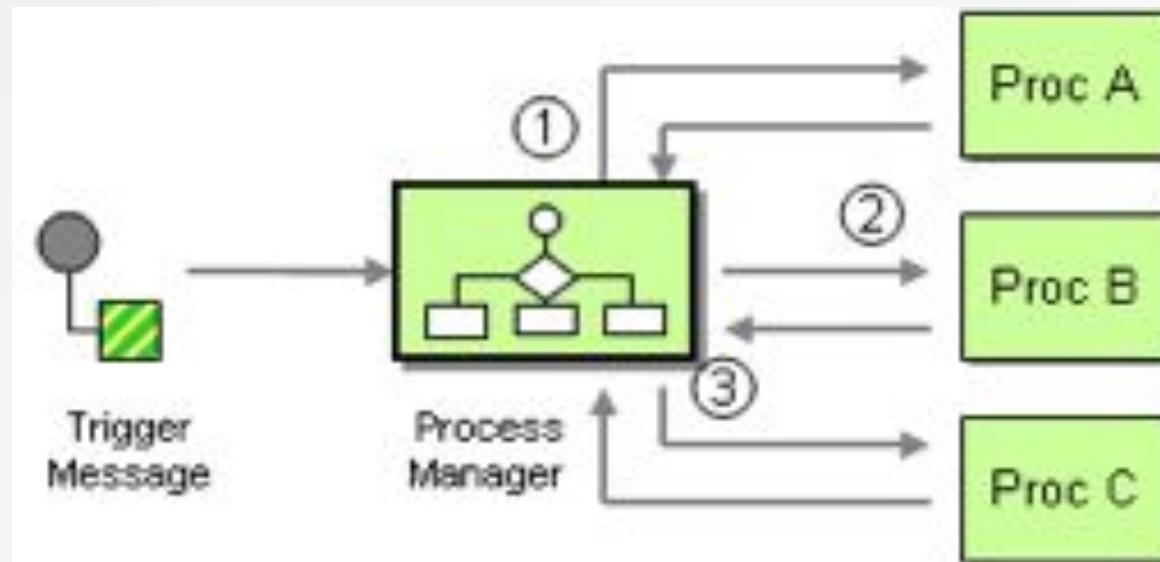
- Resequencer
 - Réception d'un flux de messages désordonnés
 - *Selon un critère métier (ex : numéro de facture)*
 - *Selon un critère technique (ex : id de corrélation)*
 - Rétention des messages dans un buffer interne (*statefull*)
 - Publication ordonnée des messages sur le channel de sortie (ce dernier se doit préserver l'ordre)
 - *ex : une file JMS avec 1 seul consommateur*
 - Comme les autres Routeurs, le Resequencer ne modifie pas (en général) le contenu des messages

- Routing Slip
 - Comment pouvons nous router un message au travers d'une série de destinations, quand la séquence d'appels n'est pas connue à la conception et peut varier selon les messages entrants ?



- Routing Slip
 - Routage dynamique, sans connaître les routes à la conception
 - Flux de messages efficace – Les messages sont envoyés uniquement aux destinations obligatoires, et les facultatives sont évitées
 - Gestion des ressources – Pas de grand nombre de channels, de routeurs
 - Flexibilité et maintenabilité – Les routes sont simples à changer et à mettre à jour

- Process Manager
 - Comment pouvons nous router un message au travers de plusieurs destinations, alors que les appels ne sont pas connus à la conception, et qu'ils ne sont pas séquentiels ?



- Process Manager
 - Utiliser une unité de traitement centrale
 - Maintenir de l'état courant (*statefull*)
 - Déterminer la prochaine étape dynamiquement
 - Véritable orchestrateur
 - Gestion des messages
- Langage BPEL créé pour ce besoin
 - *Business Process Execution Language*

- Les outils de BPM qui supportent le langage BPEL sont de vrais logiciels à part entière
- Se tourner vers d'autres solutions
 - *Oracle BPEL Process Manager*
 - *Apache ODE*
 - *Sun Open ESB*
 - *Autres solutions propriétaires*

Un exemple d'ESB

- L'ESB est un ensemble de 3 principaux composants
 - Un conteneur de services léger
 - Un serveur de messages
 - Des services, des endpoints, des éléments de routage
- ServiceMix est un conteneur de service léger
- ActiveMQ est un serveur de messages compatible JMS/AMQP
- Camel est un framework d'intégration implémentant les EIP
- La réunion des trois éléments forme donc un ESB

Les avantages des ESB open source

- Un ESB open source doit avoir les caractéristiques suivantes
 - Licence (Apache, GPL)
 - Gratuit
 - Code source disponible et public
- Une communauté très présente et très réactive
 - + de 3000 mails/mois sur les mailing lists (SM, AMQ, CML)
 - Les committers et mainteneurs impliqués directement
- Outil de suivi de bugs public (JIRA)

Apache ServiceMix

- Conteneur de services léger
- Conteneur OSGI
- ServiceMix contient
 - Apache Felix / Eclipse Equinox
 - Apache Karaf
 - ServiceMix NMR

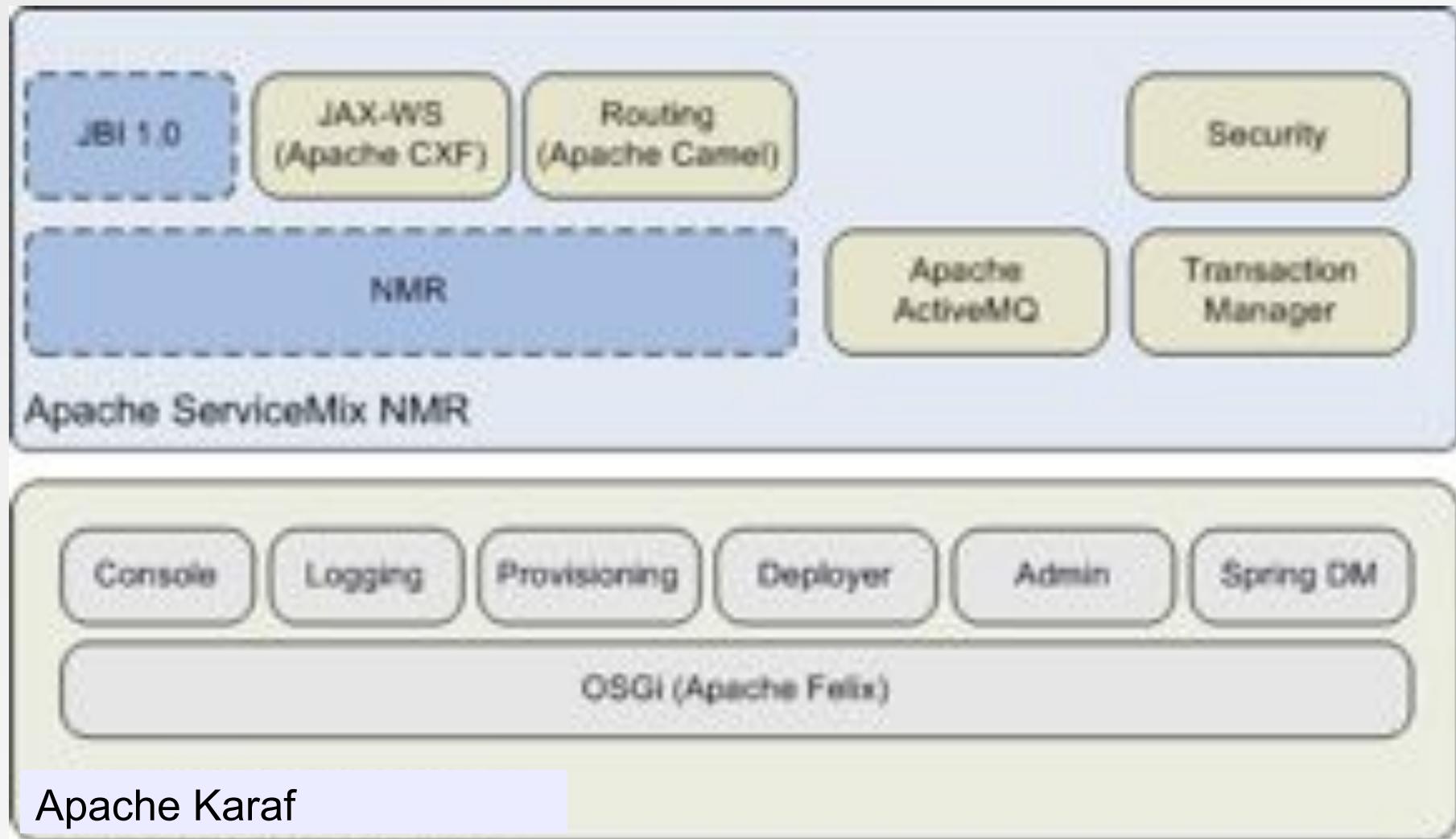


- ServiceMix s'appuie maintenant principalement sur trois technologies
 - OSGi
 - *Modularité*
 - *Classloading dynamique*
 - Camel
 - *Routage*
 - *Connectivité*
 - CXF
 - *Web Services*

Apache ServiceMix : et JBI dans tout ça ?

- JBI 1.0
 - JBI 1.0 est toujours supporté
 - Les concepteurs de ServiceMix présente le support JBI avant tout comme une rétro-compatibilité
- ServiceMix 4 s'appuie sur le principe de « Lightweight ESB »
 - Camel (ou Spring Integration)
 - Approche POJO
- JBI 2.0
 - JSR-312 : Java Business Integration 2.0
 - Statut : « inactive »

Architecture ServiceMix 4



- Serveur de messages
- Compatible spécifications JMS
- Performant et fiable (persistance KahaDB)
- Clustering / Haute Disponibilité
 - Master / Slave
 - Réseau de Brokers
- Supervision
 - Via la console
 - Via JMX



- Broker « à la carte »
 - Standalone, embarqué, serveur d'applications, moteur de servlets, etc.
- Configuration via fichiers XML basé sur Apache XBean
 - Dans le répertoire *conf/*
 - Configuration « à la mode Spring »
- La persistance
 - Dans le répertoire *data/*
 - Gérée par la base de données fichiers KahaDB
 - Possibilité d'utiliser une persistance JDBC

- Apache Camel
 - Implémentation des EIP
 - Définition des routes
 - *DSL (recommandé)*
 - *XML*
 - Framework
 - *Pas un conteneur*
 - *Pas un serveur*
 - Peut être intégré
 - *ServiceMix*
 - *ActiveMQ*



Présentation du framework Camel

- 10 bonnes raisons d'adopter un chameau
 - Excellent framework d'intégration
 - Open source et gratuit (Apache Software Foundation)
 - Support de 50 EIP
 - Support de plus de 70 types de *Endpoint* (connecteurs)
 - Création de règles très intuitives
 - Basé sur le framework Spring
 - Léger et puissant
 - Très bonne intégration à ServiceMix & ActiveMQ
 - Excellente documentation
 - Support de 19 langages (pour expressions et prédictats)

Apache Camel et Spring

- Camel étend Spring
- La configuration XML de Camel est basée sur Spring
- Camel utilise le support Spring pour :
 - La gestion des transactions
- Composants Spring accessibles dans Camel



```
<beans xmlns="http://www.springframework.org/schema/beans">

    <camelContext id="camel"
        xmlns="http://camel.apache.org/schema/spring">
        <package>com.resanet.camel</package>
    </camelContext>

</beans>
```

Fonctionnement de Camel

- Camel permet de relier des Endpoints via des routes
 - Statiques
 - Dynamiques
 - Simples
 - Complexes
- Camel permet de définir des endpoints
 - Associés à des ressources
 - Physiques ou logiques
- Les endpoints et les routes sont définis dans le CamelContext (moteur Camel)

Fonctionnement de Camel

- Exemple de route DSL Java & Xml

```
from("direct:orig").to("direct:dest");
```

```
<from uri="direct:orig" />
<to uri="direct:dest" />
```

- Définition et création du endpoint « *direct:orig* »
- Permet de relier les endpoints logiques Camel « *direct:orig* » au endpoint « *direct:dest* »
- Représente une route à part entière

Fonctionnement de Camel

- Définir les routes en DSL
 - Hériter de la classe `org.apache.camel.builder.RouteBuilder`
 - Redéfinir la méthode abstraite `configure()`
 - Un point d'entrée unique `from()` pour chaque route

```
import org.apache.camel.builder.RouteBuilder;

public class MaRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        from("direct:origine")
            .to("direct:destination");
    }
}
```

- Création des routes
 - L'appel de la méthode *from()* (qui doit être unique au sein d'une route) retourne un « processor type »
 - Ce *processor* est l'action suivante qui doit être effectuée pour poursuivre l'exécution de la route
 - Différents *processors* existent par défaut (*to*, *filter*, etc.)
 - Il est possible de définir ses propres *processors*

- Configuration et utilisation de Camel
 - Déploiement d'un fichier xml avec le DSL Xml
 - Création d'un jar contenant les classes Java Camel et un fichier *camel-context.xml*
 - Déclaration du package Java où sont situées les routes

```
<beans xmlns="http://www.springframework.org/schema/beans">

    <camelContext id="camel"
        xmlns="http://camel.apache.org/schema/spring">
        <package>com.resanet.camel</package>
    </camelContext>

</beans>
```



Les EIP avec Camel

- Content Based Router en Camel
 - Permet de faire des choix basés sur des prédictats (conditions à satisfaire pour valider le test)

```
choice().when(Predicate).to(...).otherwise().to(...).end()
```

- Exemple

```
from("seda:cbr")
    .choice()
        .when(header("orderType").isEqualTo("gadget"))
            .to("seda:gadgetInventory")

        .when(header("orderType").isEqualTo("widget"))
            .to("seda:widgetInventory")

        .otherwise()
            .to("seda:autre")
    .end();
```

- Ensemble de conditions à évaluer
 - Résultat binaire – vrai ou faux (méthode *matches*)
 - Très puissant pour créer les critères de routages
 - Support de nombreux langages
 - *XPath, XQuery, Python, Groovy, etc.*

XPath

```
import org.apache.camel.Predicate;
import org.apache.camel.builder.xml.Namespaces;

Namespaces ns = new Namespaces("ns", "http://esb.resanet.com");
Predicate price = ns.xpath("/ns:order/ns:price/text()='10'");
```

- La classe *PredicateBuilder* offre les fonctions élémentaires
 - *and, or, not, isLessThan, isNull, regex, etc.*

Construction de prédictats

```
import static org.apache.camel.builder.PredicateBuilder.and;
import static org.apache.camel.builder.xml.XPathBuilder.xpath;

Predicate price = xpath("/order/price/text()='10')");
Predicate orderType = header("orderType").isEqualTo("widget");
Predicate body = body().contains("esb");
Predicate priceOrderType = and(price,orderType);
Predicate all = and(body,priceOrderType);
```

Camel – Recipient List (mode statique)

- Recipient List en Camel
 - Critères de routage pré-définis à l'avance

```
multicast().to(Endpoint, Endpoint, ...)
```

- Exemple

```
from("seda:rlsin")
    .multicast()
    .to("seda:rlsout1", "seda:rlsout2");

from("seda:rlsinp")
    .multicast().parallelProcessing()
    .to("seda:rlsout1", "seda:rlsout2");
```

Camel – Recipient List (mode dynamique)

- Recipient List en Camel
 - Critères de routage calculé à l'exécution via une *Expression*

```
recipientList(Expression)
```

- Exemple

```
from("seda:rldin")
    .recipientList(header("to").tokenize("#"));
```

- Permet d'évaluer des expressions sur un échange de messages (utilisé par les Prédicats)
 - Résultat complexe (méthode *evalutate*)
 - Support de nombreux langages
 - *Bean, Groovy, Header, Python, SQL, XPath, etc.*

Python

```
...python("requete.headers['user'])..."
```

Camel – Resequencer

- Resequencer en Camel
 - Exemples

```
// batch resequencer
from("direct:resequencer")
    .resequencer(header("priority"))
    .batch().size(2).timeout(1000L)
    .to("direct:resequencerOut");

// stream resequencer
from("direct:resequencer2")
    .resequencer(header("priority"))
    .stream().timeout(1000L).comparator(new ReverseComparator())
    .to("direct:resequencerOut");
```

La connectivité avec Camel

Les « endpoints » Camel

- Camel propose environ 70 types de endpoints (connecteurs) pour connecter tout types de technologies / protocoles
- Pour communiquer avec le monde extérieur
 - Utilisation des composants Camel

Les composants de Camel

- Camel propose une offre complète
 - Plus d'une quarantaine de composants
 - *ActiveMQ, AMQP, Atom, CXF, CXFRS, File, Freemarker, FTP, Gmail, HDFS, HTTP, Imap, IRC, JBI, JDBC, JMS, LDAP, Mail, Nagios, Pop, Printer, RMI, Servlet, SFTP, SMTP, SNMP, SQL, TCP, UDP, XMPP, etc.*
 - La liste mise à jour régulièrement
<http://camel.apache.org/component.html>

Camel – Le composant File

- Syntaxe

```
file:directoryName [ ?options ]
```

- Quelques options couramment utilisées
 - autoCreate=true
 - delay
 - recursive
 - filter
 - move

```
from("file://order/input?move=.ok&delay=3000") . to("activemq:orders");
```

- Syntaxe

```
jms : [queue : | topic :] destinationName [ ?options ]
```

- Quelques options couramment utilisées

- replyTo
- priority
- selector
- maxConcurrentConsumers

```
from("jms : queue.in") . to("jms : queue.out") ;
```

Camel – Le composant JMS

- Configurer le provider JMS

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

<bean id="monJMS" class="org.apache.camel.component.jms.JmsComponent">
    <property name="connectionFactory">
        <bean class="org.apache.activemq.ActiveMQConnectionFactory">
            <property name="brokerURL" value="vm://bkr"/>
        </bean>
    </property>
</bean>
```

- Et utiliser le provider configuré

```
monJMS : [queue : | topic : ] destination
```



- Pour Apache ActiveMQ, utiliser le composant standard fourni avec la distribution *activemq*

- Le composant *camel-jetty*
 - Fonctionnalités en mode consumer
 - Réception d'une trame HTTP
 - Expose un service HTTP aux services externes à l'ESB
 - Retourne une réponse HTTP
 - Implémentation basé sur Jetty 6
 - Authentification BASIC HTTP
 - Support de SSL

```
jetty:http://hostname[:port] [/resourceUri] [?options]
```

Exemple:

```
jetty:https://0.0.0.0/myapp/myservice/
```

Le composant camel-http

- Le composant *camel-http* ou *camel-http4*
 - Basé sur Apache HttpClient
 - Fonctionnalités en mode producer
 - Émission d'une trame HTTP vers un service distant
 - Configuration de la connexion (SSL, Proxy, timeout)
 - Configuration des headers HTTP
 - Configuration des opérations (POST, GET, ...)

```
http:hostname[:port] [/resourceUri] [?param1=value1] [&param2=value2]
http4:hostname[:port] [/resourceUri] [?options]
```

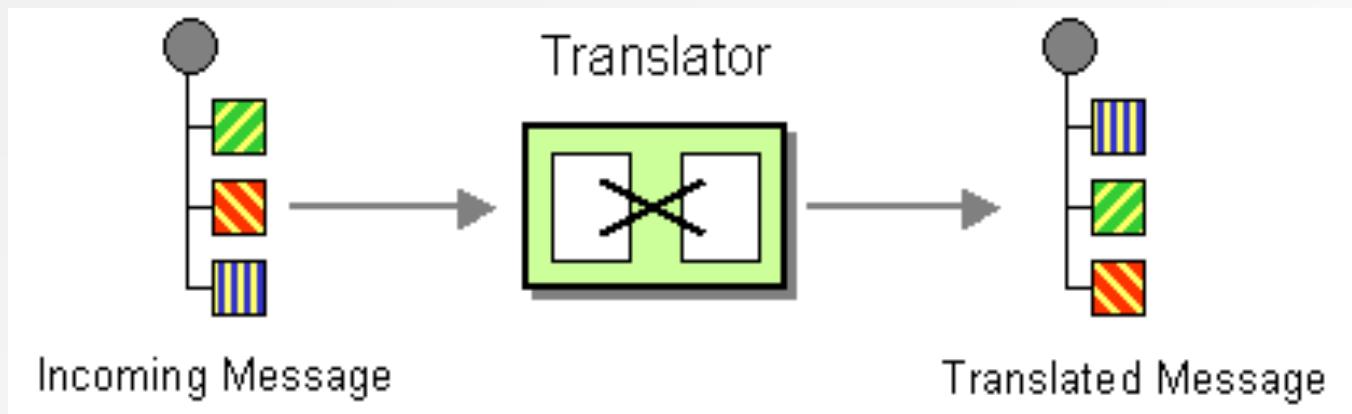
Exemple:

```
http://www.zenika.com?proxyHost=proxy.zenika.com&proxyPort=80
```



Les transformations avec Camel

- Message Translator
 - Comment les services utilisant des formats de données différents peuvent-ils communiquer ensemble ?



- Message Translator
 - Cet EIP réalise une transformation des données présentes dans le message pour en former un nouveau
 - Plusieurs niveaux de transformation possibles
 - *Couche transports (HTTP, JMS, SOAP)*
 - *Représentation des données (charset, cryptage)*
 - *Typage des données (noms des champs, types)*
 - *Couche applications (message, entité)*

Camel – Message Translator

- Message Translator en Camel
 - Transformation du message

```
transform(Expression exp)
```

- Deux manières d'utiliser cet EIP très commun pour effectuer une transformation de format
 - Utiliser le *transform DSL*
 - Faire appel à un *simple Processor*

Camel – Message Translator

- Message Translator en Camel
 - Exemple

```
// processor message translator
from("direct:msgTransProc")
    .process(new WorldProcessor())
    .to("direct:messageTranslatorOut");

// transform message translator
from("direct:msgTransTransf")
    .transform(body().append(" Transform!"))
    .to("direct:messageTranslatorOut");

static class WorldProcessor implements Processor {
    public void process(Exchange exchange) {
        Message in = exchange.getIn();
        in.setBody(in.getBody(String.class) + " World!");
    }
}
```

- Transform
 - L'usage du composant Camel XSLT est recommandé pour la transformation XML

```
to ("xslt:com/esb/resanet/requeteToReponse.xsl")
to ("xslt:http://esb.resanet.com/requeteToReponse.xsl")
```

- Les headers du message sont tous transmis par défaut au moteur XSLT



BPEL

Orchestration de services métiers

- Le langage BPEL – Business Process Execution Language
- Les processus BPEL – Synchrones et Asynchrones
- Intégrer un monde hétérogène avec BPEL
- Quand utiliser un moteur BPEL ?
- Les bonnes pratiques en BPEL

Un processes BPEL

- Business Process (processus métier)
 - Répond à un besoin d'entreprise
 - S'adapte à un modèle de données métier
 - Réalise un ensemble d'actions et de tâches
 - Est un service
- Exemple de Business Process
 - Réservation de billets d'avion
 - Format d'entrée imposé
 - Vérifier disponibilité, appeler facturation, notifier client

BPEL – Quelle utilité ?

- Processus BPEL implémente un Business Process
- Processus BPEL expose un Web Service
 - Business Process exposé via Web Service (en BPEL)
- Orchestrateur de Web Services
- Abstraction
- Un processus BPEL orchestre l'appel à d'autres WS
 - Processus BPEL
 - Services externes
- Granularité inconnue du service appelé

Un processus BPEL

- Processus BPEL
 - WSDL
 - *Types*
 - *Messages*
 - *Opérations / PortTypes*
 - BPEL
 - *PartnerLinks*
 - *Variables*
 - *Activités*

- Écrit en XML – véritable code source
- Tout est encapsulé dans la balise <process>
- Appartient au schéma XML
<http://docs.oasis-open.org/wsbpel/2.0/process/executable>
- Détermine la logique métier, l'orchestration, les transformations

Un processus BPEL

Processus BPEL

Partner links

Variables

Activités

```
<bpel:process name="HelloBPEL" targetNamespace="http://bpel.resanet.com/hellobpel"  
    suppressJoinFailure="yes" xmlns:tns="http://bpel.resanet.com/hellobpel"  
    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable">  
  
    <bpel:partnerLinks>  
        <bpel:partnerLink name="client" partnerLinkType="tns:HelloBPEL"  
            myRole="HelloBPELProvider" />  
    </bpel:partnerLinks>  
  
    <bpel:variables>  
        <bpel:variable name="input" messageType="tns:HelloBPELRequestMessage"/>  
        <bpel:variable name="output" messageType="tns:HelloBPELResponseMessage"/>  
    </bpel:variables>  
  
    <bpel:sequence name="main">  
        <bpel:receive name="receiveInput" partnerLink="client" portType="tns:HelloBPEL"  
            operation="process" variable="input" createInstance="yes"/>  
        <bpel:assign name="assign">  
            <bpel:copy>  
                <bpel:from part="payload" variable="input">  
                    <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[tns:input]]></bpel:query>  
                </bpel:from>  
                <bpel:to part="payload" variable="output">  
                    <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"> <![CDATA[tns:result]]></bpel:query>  
                </bpel:to>  
            </bpel:copy>  
        </bpel:assign>  
        <bpel:reply name="replyOutput" partnerLink="client" portType="tns:HelloBPEL"  
            operation="process" variable="output" />  
    </bpel:sequence>  
</bpel:process>
```

Les composants – Fichier WSDL

- WSDL classique
 - Types, messages, opérations, portTypes, binding, service
- Ajout de la définition des partnerLinks
 - Un partnerLink est une interface d'un processus
 - Toutes les interfaces sont des partnerLinks
 - Permet de faire le lien vers un portType WSDL

```
<plnk:partnerLinkType name="Reservation">
    <plnk:role name="ReservationProvider" portType="tns:ReservationPortType"/>
</plnk:partnerLinkType>
```

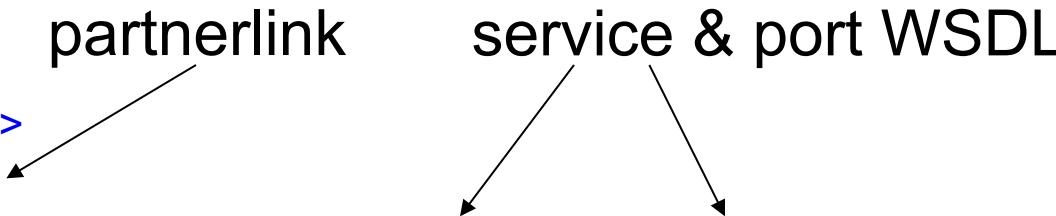
Les composants – Fichier descripteur

- N'est pas défini dans la spécification BPEL
- Tous les moteurs BPEL ont leur propre descripteur
- Permet généralement de relier les partnerLink aux interfaces concrètes
 - Car les partnerlinks sont reliés au portType dans le WSDL, et donc à l'interface abstraite
 - Besoin de rattacher le service à appeler à son adresse physique (endpoint)

Les composants – Fichier descripteur

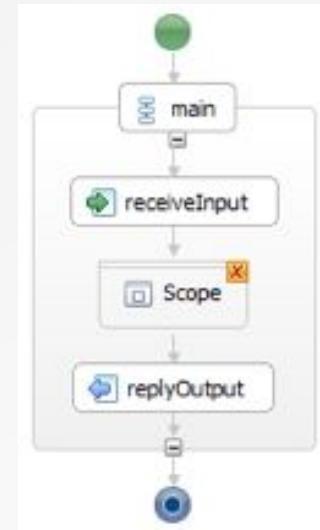
- Exemple de descripteur d'Apache ODE

```
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
        xmlns:BusinessFault="http://bpel.resanet.com/BusinessFault"
        xmlns:gestionerreur="http://bpel.resanet.com/gestionerreur">
<process name="gestionerreur:GestionErreurs">
    <active>true</active>
    <retired>false</retired>
    <process-events generate="all"/>
    <provide partnerLink="client">
        <service name="gestionerreur:GestionErreursService" port="GestionErreursPort"/>
    </provide>
    <invoke partnerLink="businessFault">
        <service name="BusinessFault:BusinessFaultService" port="BusinessFaultPort"/>
    </invoke>
</process>
</deploy>
```



Fonctionnement d'un processus BPEL

- Un processus a un état
 - 1 point de démarrage, n points d'arrêts
 - Sauvegarde de l'état (*dehydration store*)
- Maintient de variables
- Exécutions de manipulations sur des données uniquement en XML
- Appels de services internes ou externes
- Enchainement de traitements



Apache ODE : Un moteur BPEL libre

- Moteur BPEL de la fondation Apache
- Gratuit et open source, licence ASF
- La spécification WSBPEL 2.0 est 100% supportée
- Supporte 2 couches de communication
 - Web Services sur HTTP (Axis2)
 - Endpoints JBI (intégration à l'ESB ServiceMix)
- Supporte les Web Services REST
- Déploiement des processus BPEL à chaud



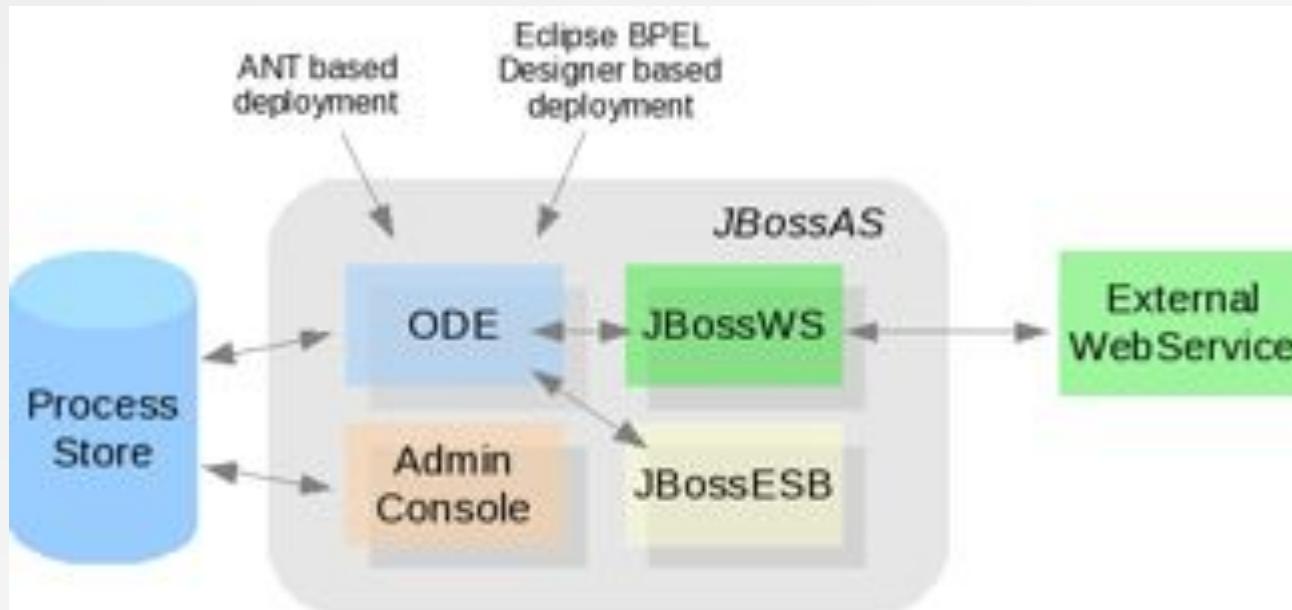
- Plusieurs types de packaging
 - Application Web sous forme de WAR (moteurs de servlets, serveurs JEE)
 - Composant JBI, à déployer sur ServiceMix (ou autre conteneur JBI)
- Support des processus durables
- Support du versionning des processus BPEL

- Sur-couche à Apache ODE
- Intégration d'ODE à JBossAS v5
- Déploiement sur l'architecture JBoss
- Couche d'intégration basée sur JAXWS
 - Implémentation au choix (JbossNative, JAXWS-RI, CXF)
- Utilitaire Ant de déploiement
- Intégration facilitée avec JBoss ESB



JBoss
Community

Jboss Riftsaw

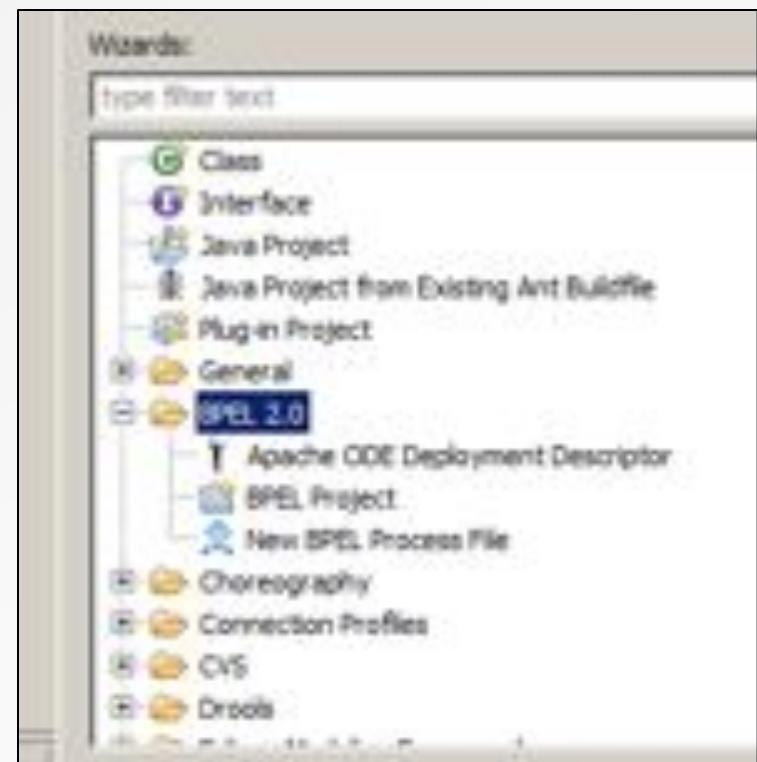


- Basé sur le plugin Eclipse BPEL (WSBPEL 2.0)
- Intégré dans les JBoss tools à partir de la v3.1
 - Nécessite Eclipse Helios 3.6
- Téléchargement des plugins via le site Jboss
 - Plugins SOA Development

<http://download.jboss.org/jbosstools/updates/stable/helios/>

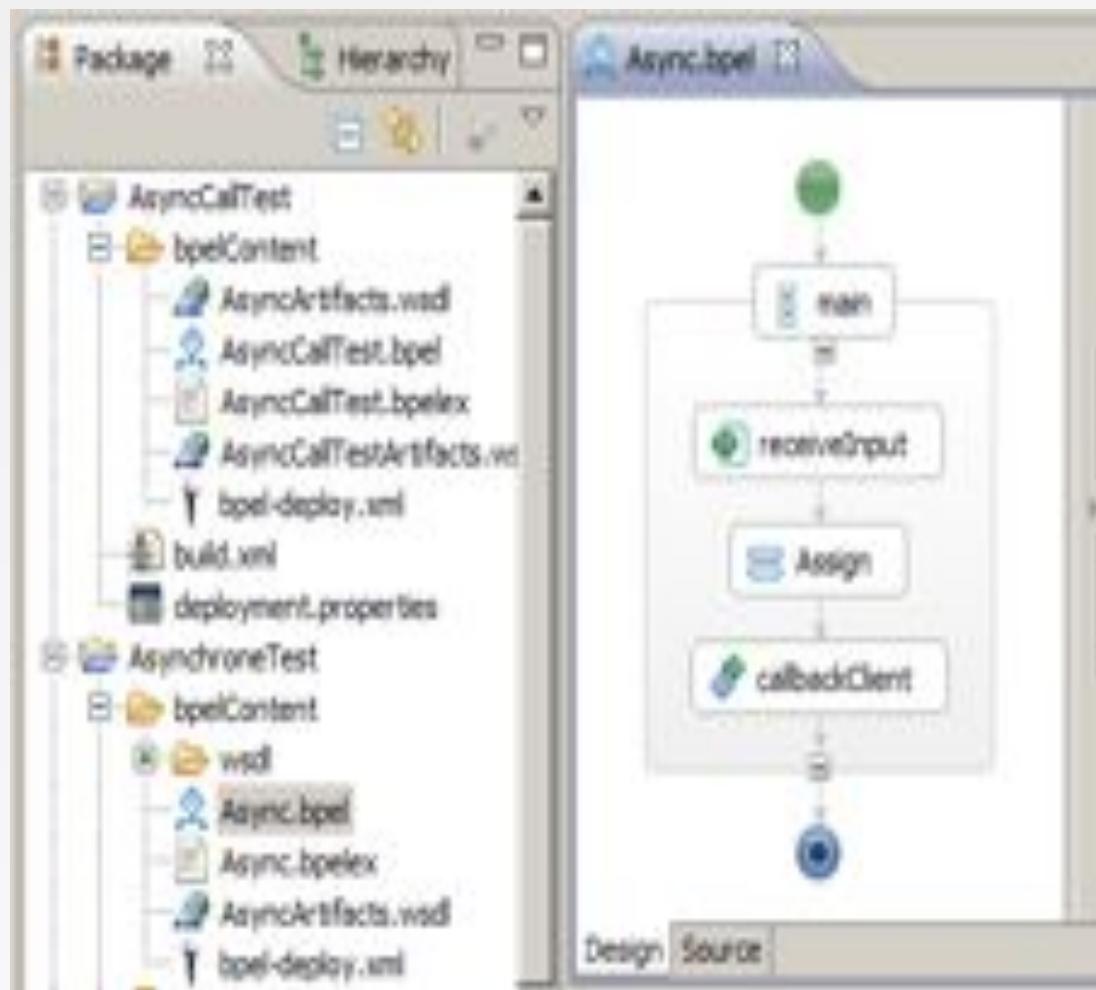
L'IDE BPEL – Présentation

- L'IDE propose désormais un onglet BPEL 2.0 dans les wizards de création
 - Descripteur de déploiement
 - Projet BPEL
 - Templates de processus BPEL
- Pas de perspective spécifique !



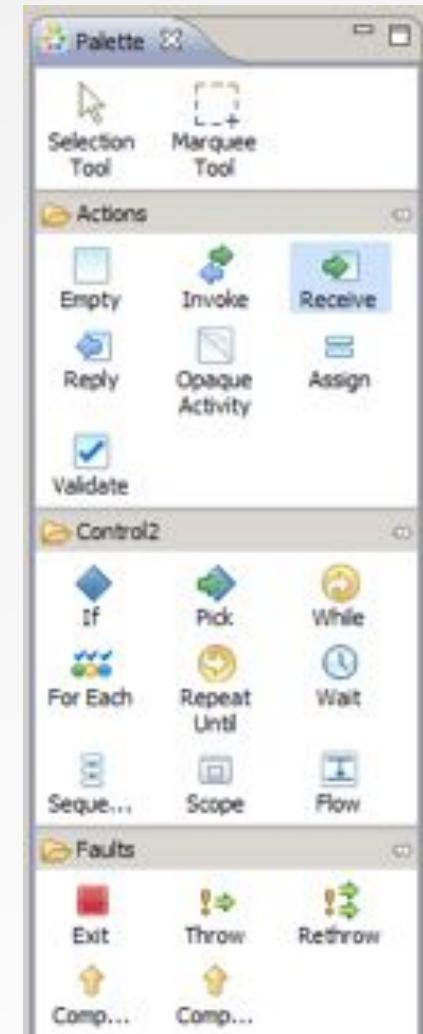
L'IDE BPEL – Édition

- L'IDE permet d'éditer graphiquement les processus BPEL
- Onglets Design & Source pour visualiser et interagir directement sur le XML source



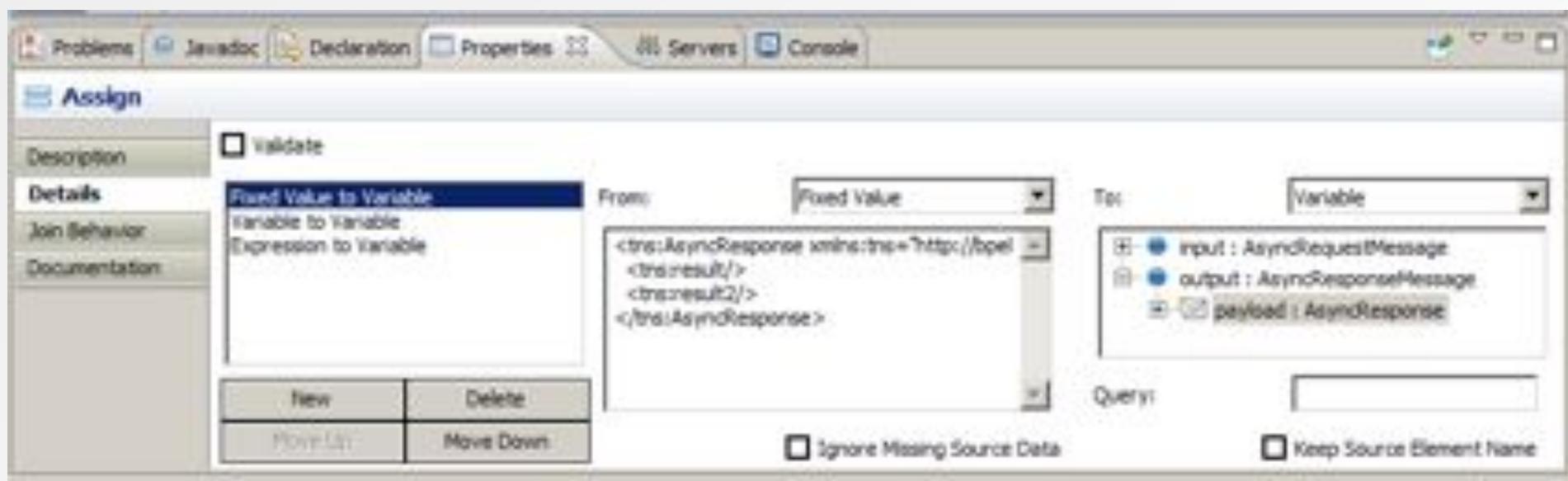
L'IDE BPEL – Édition

- La palette graphique permet de créer et modifier les processus avec *drag & drop* des composants BPEL
- Toutes les activités BPEL 2.0 sont disponibles



L'IDE BPEL – Édition

- La vue *Properties* permet de spécifier les propriétés des différentes briques (receive, invoke, assign, etc.)



- JDeveloper 10 propose un éditeur BPEL 1.1
 - Indispensable pour Oracle BPEL PM 10.1.3.x
- JDeveloper 11 dispose désormais d'un éditeur BPEL 2.0
 - Indispensable pour Oracle BPEL PM 11.1.1.x
- L'éditeur rajoute des descripteurs spécifiques au moteur Oracle



Les fondements du langage BPEL

- Le développement du processus BPEL est réalisé dans le fichier portant l'extension *.bpel*
- Un nombre fini de briques ou d'activités BPEL constitue la syntaxe du langage
 - Analogie avec les mots clés d'un langage traditionnel
- Ces activités possède leur propre notation XML, avec également leurs propres attributs qui les composent

Activités d'orchestration - *Receive*

- Permet de recevoir un message d'un partnerLink.
L'attente est bloquante jusqu'à réception du message adéquat.

- Création d'une instance d'un processus BPEL possible
- Participation à une instance déjà en cours



receive

- Notation XML

```
<bpel:receive name="receive" partnerLink="client"  
portType="tns:FormationBrique" operation="process"  
variable="input" createInstance="yes"/>
```

Activités d'orchestration - *Reply*

- Envoie un message de réponse ou une exception à la fin d'une transaction synchrone

- Utilisé uniquement sur les opérations synchrones (input / output)
- Ne signifie pas obligatoirement la fin du processus



- Notation XML

```
<bpel:reply name="reply" partnerLink="client"  
portType="tns:FormationBrique" operation="process"  
variable="output" />
```

Activités d'orchestration - *Invoke*

- Envoie un message à un service partenaire (partnerLink), et récupère éventuellement la réponse
 - Peut appeler des opérations synchrones (Request / Response)
 - Peut également appeler des opérations asynchrones (One way)



- Notation XML

```
<bpel:invoke name="invoke" partnerLink="formationBrique"  
operation="process" portType="tns:FormationBrique"  
inputVariable="inputVar" outputVariable="outputVar"/>
```

Manipulation de données - Assign

- Permet de manipuler les variables d'un processus
 - Initialisation de variables / mise à jour de variables
 - Copie de variables
 - Manipulation XML (via le langage XPath)
 - Possibilité de valider la variable sur son schéma XSD
- Notation XML

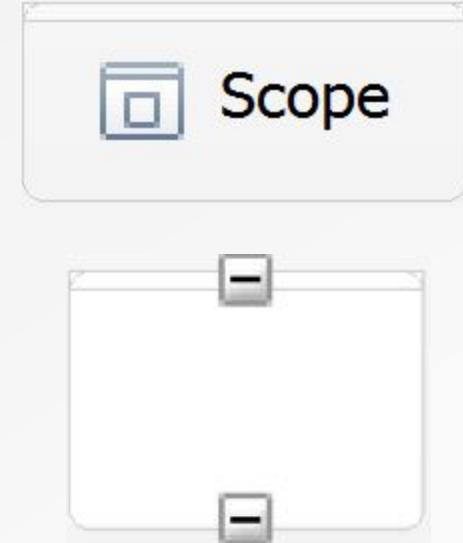


```
<bpel:assign validate="no" name="assign">
  <bpel:copy>
    <bpel:from part="payload" variable="input">
      <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[tns:input]]></bpel:query>
    </bpel:from>
    <bpel:to part="payload" variable="output">
      <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[tns:result]]></bpel:query>
    </bpel:to>
  </bpel:copy>
</bpel:assign>
```

Activités structurantes – Scope

- Découpe en différentes parties logiques un processus BPEL
 - Déclaration de variables locales
 - Récupération d'exceptions, compensations
- Permet d'avoir des activités imbriquées
 - Ne peut contenir qu'une seule activité
- Notation XML

```
<bpel:scope name="scope">  
    [...]  
</bpel:scope>
```



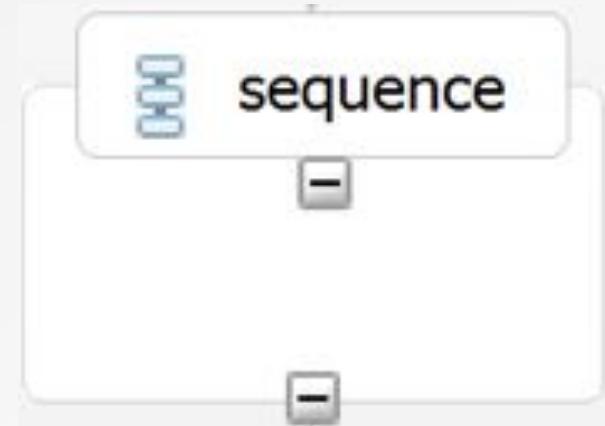
Activités structurantes – Sequence

- Permet d'avoir une série d'activités, exécutées de manière séquentielle

- Permet d'avoir des activités imbriquées
 - Peut contenir n activités

- Notation XML

```
<bpel:sequence name="sequence">  
    [...]  
</bpel:sequence>
```

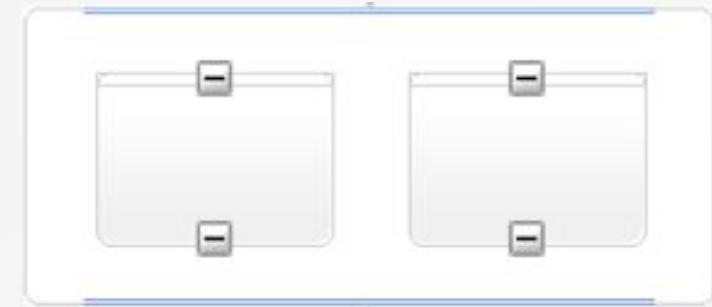


Activités structurantes – *Flow*

- Permet d'avoir une série d'activités, exécutées de manière concurrentes
 - L'exécution simultanée dépend des implémentations des moteurs BPEL
- Permet d'avoir des activités imbriquées
 - Peut contenir n activités
- Notation XML

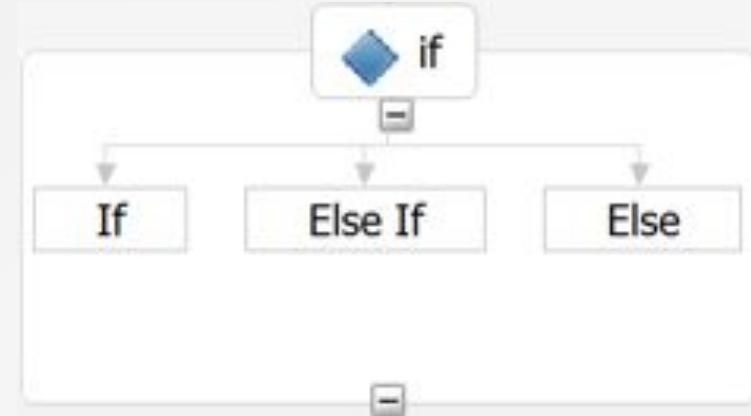
```
<bpel:flow name="flow">  
    [...]  
</bpel:flow>
```

flow



Activités conditionnelles – *If* - *Else if* - *Else*

- Fournit n branches conditionnelles
 - Chacune de ces branches est associée à une expression booléenne
 - La première branche (de gauche à droite) dont la condition est vraie est exécutée
 - Si aucune condition vérifiée, la branche *Else* est finalement exécutée



- Notation XML

```
<bpel:if name="if">
  <bpel:condition><![CDATA[true()]]></bpel:condition>
  [...]
  <bpel:elseif>
    <bpel:condition><![CDATA[true()]]></bpel:condition>
    </bpel:elseif>
    <bpel:else>[...]</bpel:else>
</bpel:if>
```

Activités itératives – *While*

- Effectue le traitement aussi longtemps que la condition booléenne spécifiée est vraie
 - Expression XPath



- Notation XML

```
<bpel:while name="While">
    <bpel:condition><![CDATA[true()]]></bpel:condition>
    [...]
</bpel:while>
```

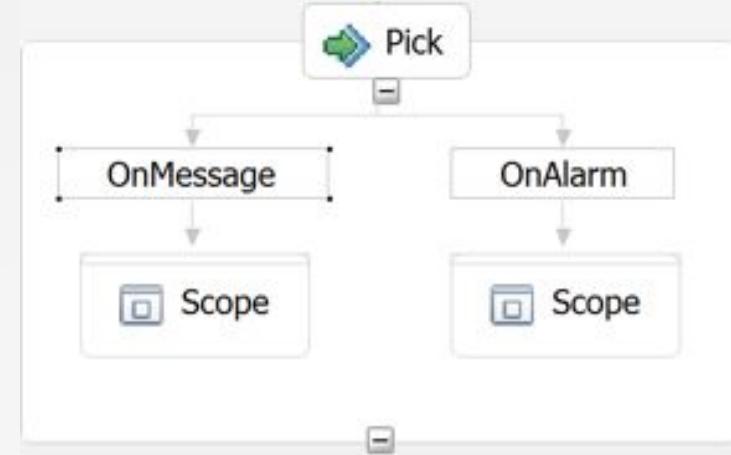
- Attente bloquante
 - Statiquement
 - Dynamiquement avec expression XPath
- La nature de l'attente est soit
 - Une durée *for*
 - Une date déterminée *until*
- Notation XML



```
<bpel:wait name="wait">  
    <bpel:for><![CDATA['PT1H']]></bpel:for>  
</bpel:wait>
```

Activité Pick

- Attente bloquante
 - Jusqu'à ce qu'un des types de messages précisés arrive
 - Ou jusqu'à ce que le timer expire
 - Lorsque un message arrive, l'attente des autres types de message est désactivée
- Situé dans le flux du processus BPEL
- Notation XML



```
<bpel:pick name="Pick">
    <bpel:onMessage partnerLink="client" operation="process" variable="input">
        <bpel:scope name="Scope">[...]</bpel:scope>
    </bpel:onMessage>
    <bpel:onAlarm>
        <bpel:scope name="Scope">[...]</bpel:scope>
        <bpel:for><![CDATA['PT1H']]></bpel:for>
    </bpel:onAlarm>
</bpel:pick>
```



Manipulation variables XML

Manipulation XML en BPEL

- Rappel : Toutes les données manipulées nativement par BPEL sont au format XML
 - Messages reçus, Variables internes
- Les activités *assign* sont les seules activités permettant de manipuler les données XML
 - Manipulation directe
 - Manipulation littérale
 - Manipulation d'expressions
- Utilisée pour copier une source vers
 - Une variable
 - Un partnerLink (considéré comme une variable en BPEL)

Initialisation des variables XML

- Avant de pouvoir manipuler une variable XML, il est indispensable de l'initialiser
 - Avec les données XML souhaitées
 - Pour éviter les *selectionFailure*
- Pour cela, il faut utiliser le mode *literal*

```
<bpel:from>
    <bpel:literal>
        <monFragmentXML>
            <id/>
        </monFragmentXML>
    </bpel:literal>
</bpel:from>
```

Les fonctions XPath génériques

- BPEL utilise constamment les expressions XPath
 - Pour toutes les conditions
 - Pour accéder au données dans les *assign*
- BPEL utilise également les fonctions XPath standard, et étendue
 - concat()
 - substring()
 - count()
 - contains()
 - position()
 - start-with()
 - etc.
- Cette liste dépend généralement des implémentations

Le langage XPath dans BPEL

- Depuis BPEL 2.0, la spécification impose en plus le support de la fonction suivante
 - `bpel:doXSLT` pour réaliser directement les transformation XSLT, sans passer par une fonction propriétaire
 - Support natif des transformations XSLT
- L'ancienne fonction `bpws:getVariableData` qui était utilisé pour accéder aux variables est remplacée par un « \$ »
 - BPEL 1.1 → `bpws:getVariableData("message", "payload", "tns:in")`
 - BPEL 2.0 → `$message.payload/tns:in`
 - Simplifie énormément la lisibilité du code
 - Simplifie également le développement

- Bien que cela ne soit pas indiqué dans la spécification, tous les éditeurs de moteur BPEL offre la possibilité d'utiliser des fonctions custom XPath
- Fonction custom XPath
 - Fonction XPath utilisateur réalisant le traitement désiré
 - Il est donc possible d'étendre les capacités de BPEL à l'infini
- Apache ODE utilise le moteur Saxon 9, ces fonctions doivent donc être écrites en Java
 - `xmlns:fonc="java:com.resanet.xpath.MaCustomXPath"`
 - Avec ce type de namespace, Saxon enregistre directement la XPath et la rend disponible au runtime dans le processus BPEL

Fonction custom XPath pour Apache ODE

- Développer une classe Java ayant une méthode *static*, et la positionner dans le classpath

```
package com.resanet.bpel.xpath;

public class ConcatString {

    /**
     * Concatène deux String.
     *
     * @param str1 string numéro 1
     * @param str2 string numéro 2
     *
     * @return un nouveau String
     */
    public static String concat(String str1, String str2) {
        return str1.concat(str2);
    }
}
```

- Déclarer le namespace dans le processus BPEL

- `xmlns:cnct="java:com.resanet.bpel.xpath.ConcatString"`

Fonction custom XPath pour Apache ODE

- Utiliser sa fonction XPath dans un *assign* par exemple

```
<bpel:copy>
  <bpel:from expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang>xpath2.0">
    <![CDATA[cnct:concat($input.payload/tns.input, ' world')]]>
  </bpel:from>
  <bpel:to part="payload" variable="output">
    <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang>xpath1.0">
      <![CDATA[tns:result]]></bpel:query>
    </bpel:to>
</bpel:copy>
```

- Attention
 - Il faut utiliser le xpath2.0 pour utiliser les XPath avec Saxon

- Grâce à ce procédé, nous pouvons très simplement passer dans le monde Java
 - Possibilité de passer des paramètres
 - Peut retourner un résultat
- Tout le potentiel du langage Java est alors à disposition
 - Spring
 - Java EE
 - etc.

Les transformations XSLT

- XSLT – *eXtensible Stylesheet Language Transformations*
 - Basé sur le langage XSL
- Langage de transformation XML
 - À partir de 2 documents
 - *Document XML*
 - *Feuille XSLT*
- On applique la transformation pour produire un nouveau document XML en sortie
- Langage basé sur XPath pour naviguer dans l'arbre XML

Les transformations XSLT

- Exemple d'une feuille XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:tns="http://bpel.tnsnet.com/xslt">
    <xsl:template match="/">
        <tns:output>
            <xsl:value-of select="tns:input/tns:value"/>
        </tns:output>
    </xsl:template>
</xsl:stylesheet>
```

Les transformations XSLT en BPEL

- BPEL est capable d'utiliser les transformations XSLT
 - Fonction XPath appelant un moteur de transformation
 - En BPEL 2.0
 - Support natif des transformations XSLT
 - XPath `bpel:doXSLTransform` pour réaliser directement les transformation XSLT, sans passer par une fonction propriétaire

```
object bpel:doXsltTransform(string, node-set, (string, object) *)
```

variable retour

uri XSLT

variable d'entrée

paramètres

Les transformations XSLT en BPEL

- Les paramètres fournis à la feuille XSL sont du type
 - String = QName du paramètre
 - Object = Valeur du paramètre (variable, expression Xpath)
- Il est possible de fournir *n* paramètres
- **Les manipulation complexes de données doivent être réalisées en XSLT**
 - **Le langage BPEL n'est pas prévu pour cela**
 - **Très difficile à maintenir en BPEL**
- Exemple

```
bpel:doXslTransform("xslt/feuille.xslt", $var)
```



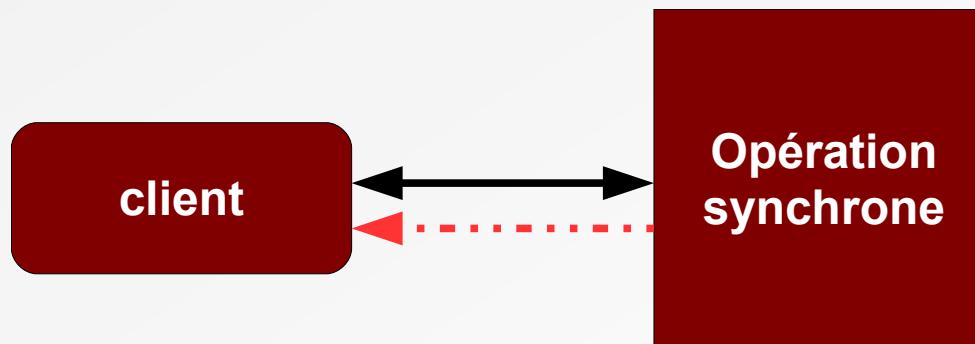
Orchestration de services

Les fondements de l'orchestration

- 2 types de communications possibles dans l'histoire des systèmes d'informations
 - Synchrone
 - Asynchrone
- Il faut réserver les échanges synchrones dans le cadre de requêtes nécessitant une réponse (quasi-) immédiate
- Il est souvent préférable, si possible, de privilégier les échanges asynchrones au sein d'un système d'information pour une question de robustesse
 - Réseau surchargé
 - Base de données HS
 - etc.

Les fondements de l'orchestration

- Synchrone
 - Request / Response
 - Réponse immédiate (quelques secondes)
 - Peut éventuellement retourner une Fault
 - L'attente de la réponse est bloquante



WSDL synchrone

- WSDL synchrone

[...]

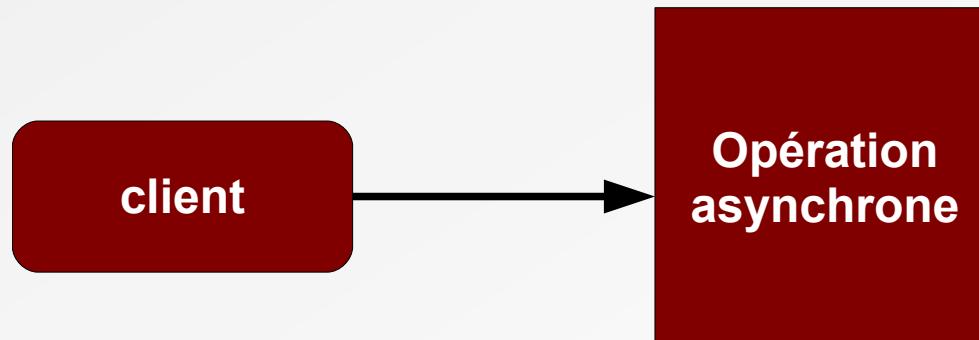
```
<message name="RequestMessage">
    <part name="payload" element="tns:Request"/>
</message>
<message name="ResponseMessage">
    <part name="payload" element="tns:Response"/>
</message>

<portType name="operationSynchrone">
    <operation name="process">
        <input message="tns:RequestMessage"/>
        <output message="tns:ResponseMessage"/>
    </operation>
</portType>
```

[...]

Asynchrone

- Asynchrone
 - One way
 - Pas de réponse, ni de Fault
 - Aucune attente du côté du client
 - Méthode « send and forget »



- Comment avoir un retour ?

WSDL asynchrone

- WSDL asynchrone

[...]

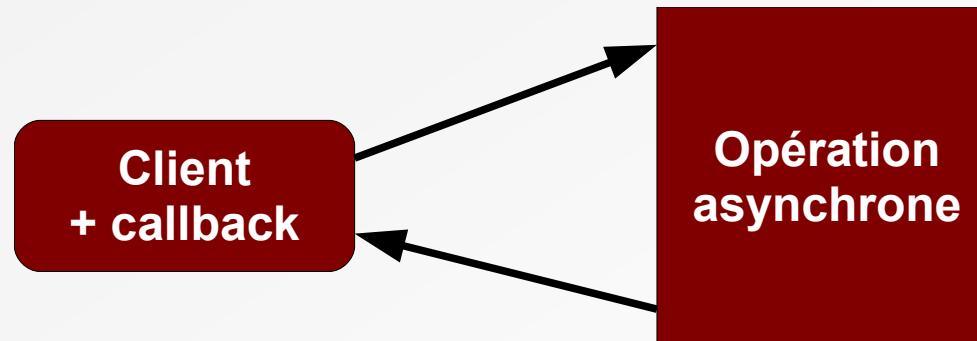
```
<message name="RequestMessage">
    <part name="payload" element="tns:Request"/>
</message>

<portType name="operationAsynchrone">
    <operation name="initiate">
        <input message="tns:RequestMessage"/>
    </operation>
</portType>
```

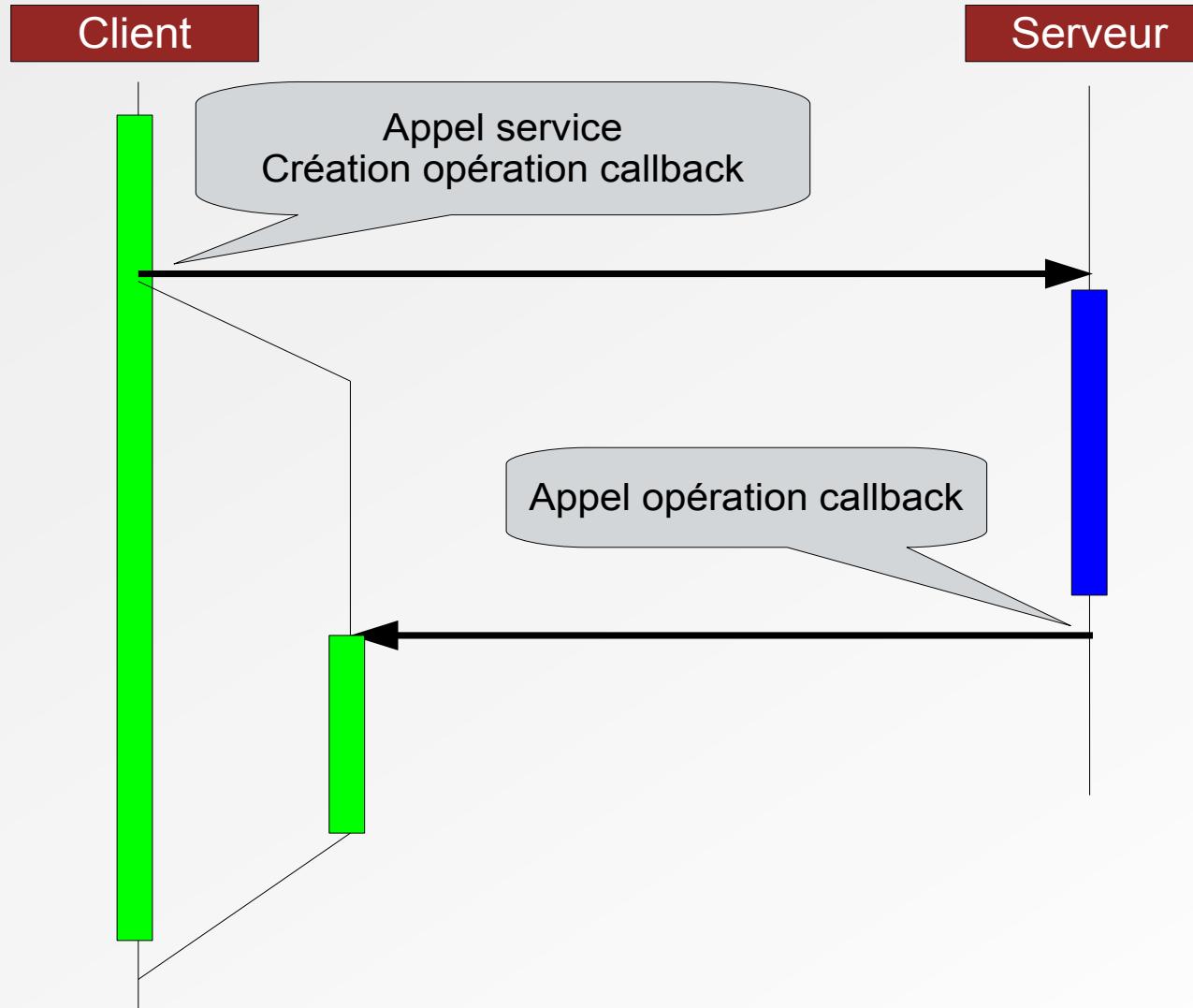
[...]

Asynchrone – Obtenir une réponse

- Asynchrone
 - Pas de réponse car uniquement une requête dans l'opération d'appel
- Il faut une autre opération asynchrone pour envoyer une réponse
 - Cette opération doit être implémentée côté client
 - Opération de callback
 - Serveur → client



Asynchrone – Obtenir une réponse



Asynchrone – Obtenir une réponse

- WSDL asynchrone pour la réponse (callback)

[...]

```
<message name="ResponseMessage">
    <part name="payload" element="tns:Response"/>
</message>
<portType name="operationAsynchroneCallback">
    <operation name="onResult">
        <input message="tns:ResponseMessage"/>
    </operation>
</portType>
```

[...]

Asynchrone et callback – WSDL

- L'intégralité de ces informations sont portés par le WSDL côté serveur
 - L'opération *initiate* est implémentée côté serveur
 - L'opération *onResult* est implémentée côté client

```
<message name="RequestMessage">
    <part name="payload" element="tns:Request"/>
</message>
<message name="ResponseMessage">
    <part name="payload" element="tns:Response"/>
</message>

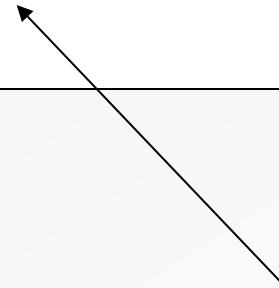
<portType name="operationAsynchrone">
    <operation name="initiate">
        <input message="tns:RequestMessage"/>
    </operation>
</portType>
<portType name="operationAsynchroneCallback">
    <operation name="onResult">
        <input message="tns:ResponseMessage"/>
    </operation>
</portType>
```

Asynchrone et callback – WSDL

- Il y a alors 2 services à définir
 - Celui qui fournit l'opération initiate
 - Celui qui fournit l'opération onResult (à implémenter par le client)

```
<service name="ProcessService">
  <port name="ProcessPort" binding="tns:ProcessBinding">
    <soap:address location="http://localhost:8080/Process"/>
  </port>
</service>

<service name="ProcessServiceCallback">
  <port name="ProcessCallbackPort" binding="tns:ProcessCallbackBinding">
    <soap:address location="http://set.by.caller"/>
  </port>
</service>
```



Adresse dynamique, remplacée par celle du client

Asynchrone – Corrélations

- La corrélation est un moyen pour déterminer si plusieurs messages (au moins 2) ont un lien entre eux
- Exemple
 - Appel d'un service de réservation de billets d'avion asynchrone
 - Fourniture d'un identifiant de commande dans la requête
 - La réponse du serveur retourne ce même identifiant
 - Le client peut corréler la réponse avec la requête
- La corrélation est indispensable pour les communications asynchrones
 - Si pas de corrélation, impossible de rattacher la réponse reçue avec la requête envoyée

- BPEL utilise la corrélation entre les messages qui transitent sur les différentes interfaces (partnerLinks)
 - Lors d'un callback asynchrone, un message est envoyé via un Web Service sur un portType déterminé
 - Problème
 - L'exécution d'un processus BPEL est nommée instance BPEL
 - Plusieurs instances d'un même processus BPEL peuvent être actives simultanément
 - Comment savoir à quelle instance le message reçu doit être envoyée ?
- **c'est le rôle de la corrélation**

- Il existe deux types de corrélation
 - Corrélation native (implicit correlation)
 - Custom corrélation (explicit correlation)
- La corrélation native est mise en œuvre automatiquement par BPEL pour les communications entre processus BPEL
- La custom corrélation est à implémenter manuellement afin de pouvoir recevoir des messages externes

- Chaque instance BPEL qui veut recevoir un callback doit initier des paramètres de corrélation
- Ainsi, le moteur BPEL sait que le message reçu qui satisfait les paramètres de corrélation de telle instance doit lui être envoyé
- Attention
 - Les corrélations sont uniquement utilisées lorsque l'on souhaite attacher un message à une instance donnée
 - Lors de création d'instance BPEL, la corrélation n'est pas nécessaire puisque l'instance n'existe pas encore

- La spécification ne traite pas le sujet des interactions entre les processus BPEL
 - Utilisation de la corrélation native
- Cependant, la majorité des implémentations implémente la corrélation native
 - La communication asynchrone entre processus BPEL est totalement transparente pour le concepteur
 - Pas de custom corrélation à mettre en place
 - Tout est gérer automatiquement
 - Basée sur le WS-Addressing sur Oracle
 - Basée sur un session id sur ODE
- Supportée chez Oracle BPEL, Apache ODE

- Spécification WS-*
- WS-Addressing permet aux Web Services de communiquer des informations d'adressage
- Les données de routage de messages incluses au header SOAP

```
<Envelope>
  <Header>
    <Action>http://resanet/hotelPortType/reserver</Action>
    <MessageID>urn:uuid:0123456789</MessageID>
    <To>http://localhost:8080/services/hotel</To>
    <ReplyTo>
      <Address>http://localhost:9990/decoupled_endpoint</Address>
    </ReplyTo>
  </Header>
  <Body>
    [ . . . ]
  </Body>
</Envelope>
```

BPEL – Le WS-Addressing

- Au retour

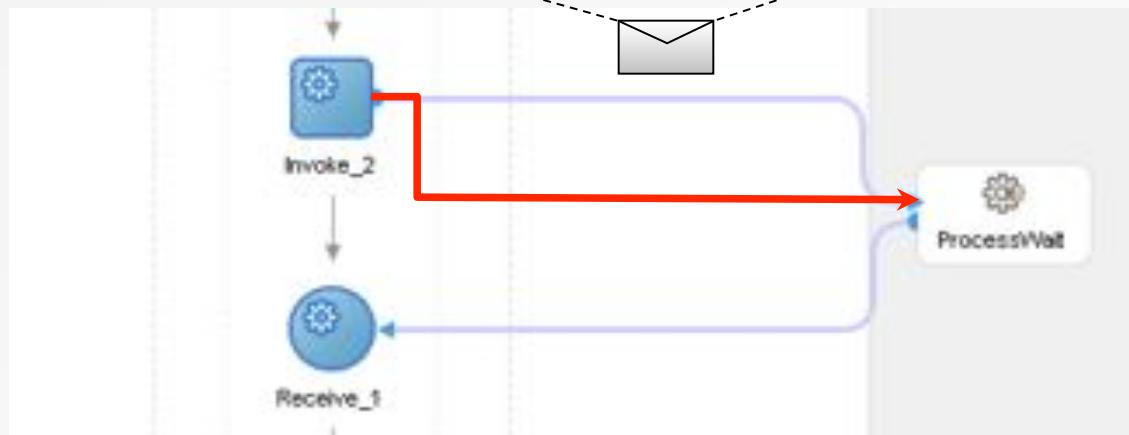
```
<Envelope>
  <Header>
    <Action>http://resanet/hotelPortType/reserverResponse</Action>
    <MessageID>urn:uuid:345123</MessageID>
    <To>http://localhost:9990/decoupled_endpoint</To>
    <RelatesTo>urn:uuid:0123456789</RelatesTo>
  </Header>
  <Body>
    [...]
```

- Le principe de la corrélation native
 - La corrélation s'effectue simplement de façon transparente
 - Les données ne sont pas des données utilisateurs, mais des données techniques ajoutées dans le Header SOAP.
- Le WSA est la norme provenant du W3C adoptée pour toutes les problématiques de corrélations entre les Web Services asynchrones

BPEL – Exemple corrélation native

Processus	Id	Contexte

```
<soap:Envelope ...>
  <soap:Header>
    <wsa:MessageID>123</wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>
        http://localhost:9700/process/callback
      </wsa:Address>
    </wsa:ReplyTo>
  <soap:Header>
  <soap:Body>
  ...
  </soap:Body>
</soap:Envelope>
```

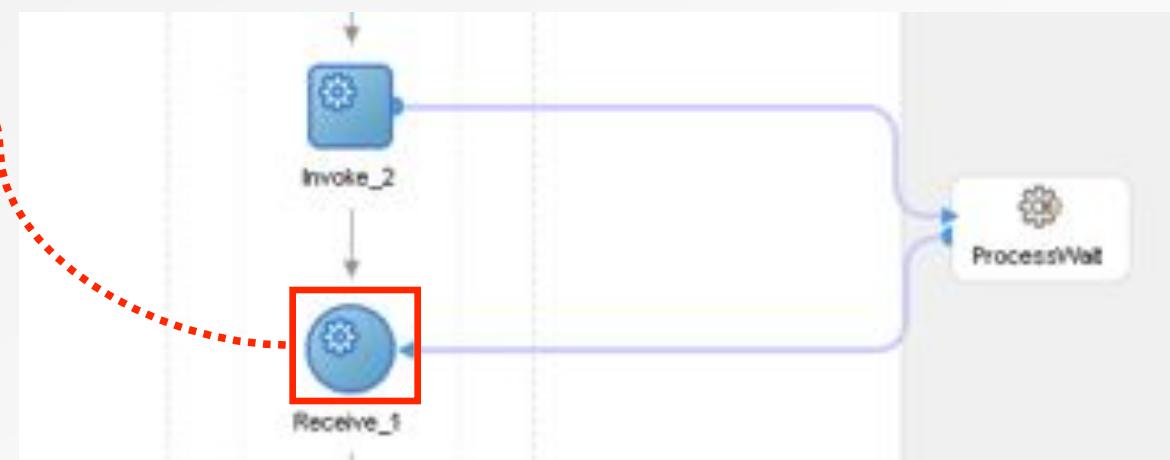


ProcessAppel

ProcessWait

BPEL – Exemple corrélation native

Processus	Id	Contexte
ProcessAppel	123	<ProcessAppel> ... <Receive_1> en cours </Receive_1> </ProcessAppel>

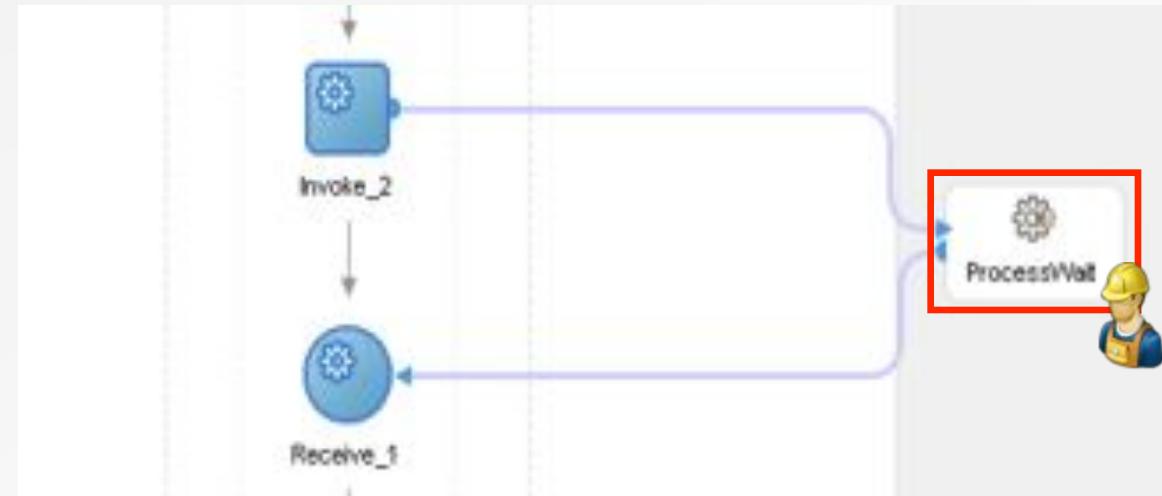


ProcessAppel

ProcessWait

BPEL – Exemple corrélation native

Processus	Id	Contexte
ProcessAppel	123	<ProcessAppel> ... <Receive_1> en cours </Receive_1> </ProcessAppel>



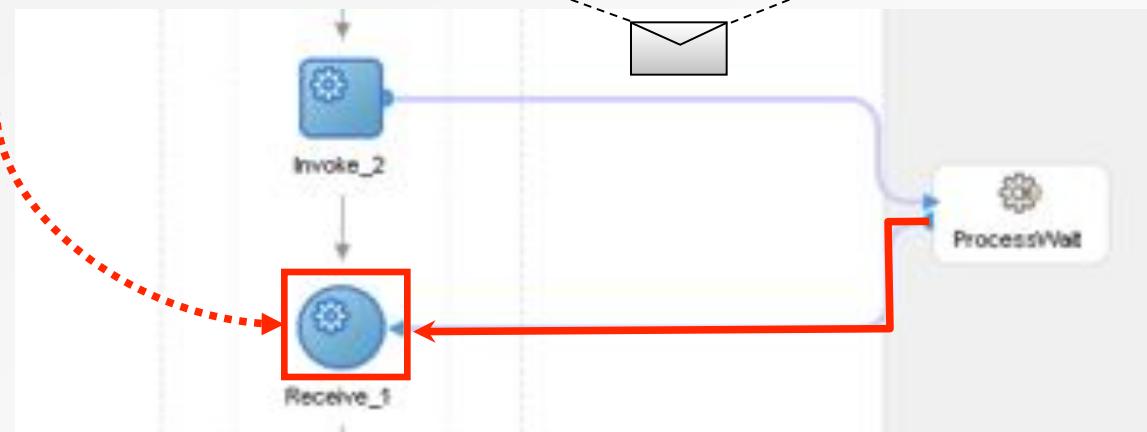
ProcessAppel

ProcessWait

BPEL – Exemple corrélation native

Processus	Id	Contexte
ProcessAppel	123	<ProcessAppel> ... <Receive_1> en cours </Receive_1> </ProcessAppel>

```
<soap:Envelope . . .>
<soap:Header>
  <wsa:RelatesTo>123</wsa:RelatesTo>
<soap:Header>
<soap:Body>
...
</soap:Body>
</soap:Envelope>
```



ProcessAppel

ProcessWait

- Il est possible d'envoyer des messages externes à BPEL
 - Messages qui ne proviennent pas directement d'autres processus
 - Messages envoyés soit
 - *d'un autre système*
 - *d'un moteur de workflow*
 - *d'une IHM*
 - etc.
- Il faut alors mettre les mécanisme de corrélation custom en place
 - Impossible d'utiliser la corrélation native étant donné que le moteur ne gère pas le parternlink
 - Obligation d'utiliser la custom corrélation

→ **Le principe et le fonctionnement sont strictement identiques à une corrélation native**



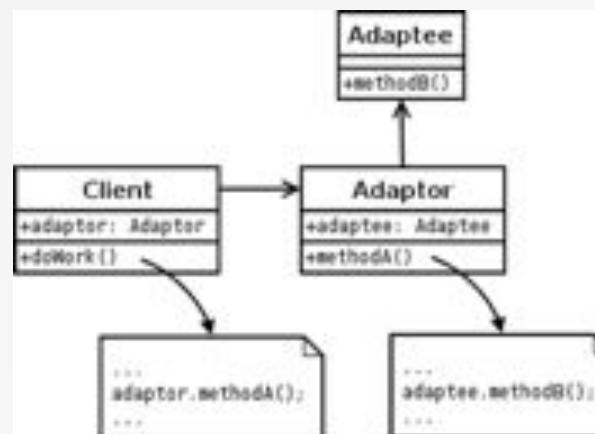
Intégrer un monde hétérogène

Comment intégrer un SI existant ?

- Le langage BPEL est conçu et implémenter pour s'interfacer avec des interfaces Web Services uniquement
 - Attention, il s'agit uniquement d'une interface
- Problème
 - Comment intégrer le reste du monde ?
 - Base de données ?
 - JMS ?
 - Fichiers ?
 - HTTP ?
 - Web Services différents (REST, RPC) ?
 - etc.

Les adaptateurs

- Introduction de la notion d'adaptateurs
 - Également nommés
 - *Adapters*
 - *Connecteurs*
- Un adaptateur en BPEL reprend la même notion que le très célèbre design pattern Adapter

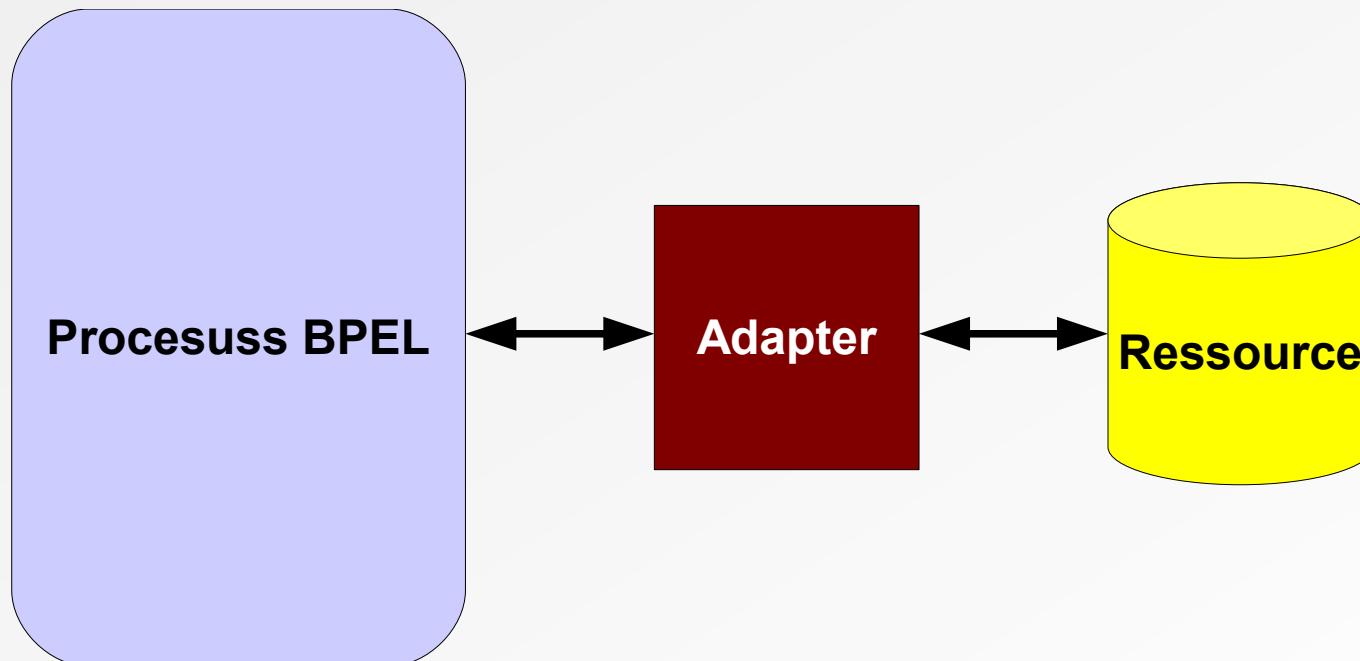


Utilisation des adaptateurs

- Les adaptateurs permettent de faire le pont entre le monde des Web Services et
 - Une autre technologie
 - Un autre protocole
- Il est alors possible d'intégrer toutes les applications possibles et existantes, même si elles communiquent avec un format propriétaire
- Les adaptateurs les plus courants
 - JMS
 - Base de données
 - Fichiers (via un tamponnage en base de données parfois)

Utilisation des adaptateurs

- De façon générale, un adaptateur permet d'accéder et de communiquer avec une ressource externe

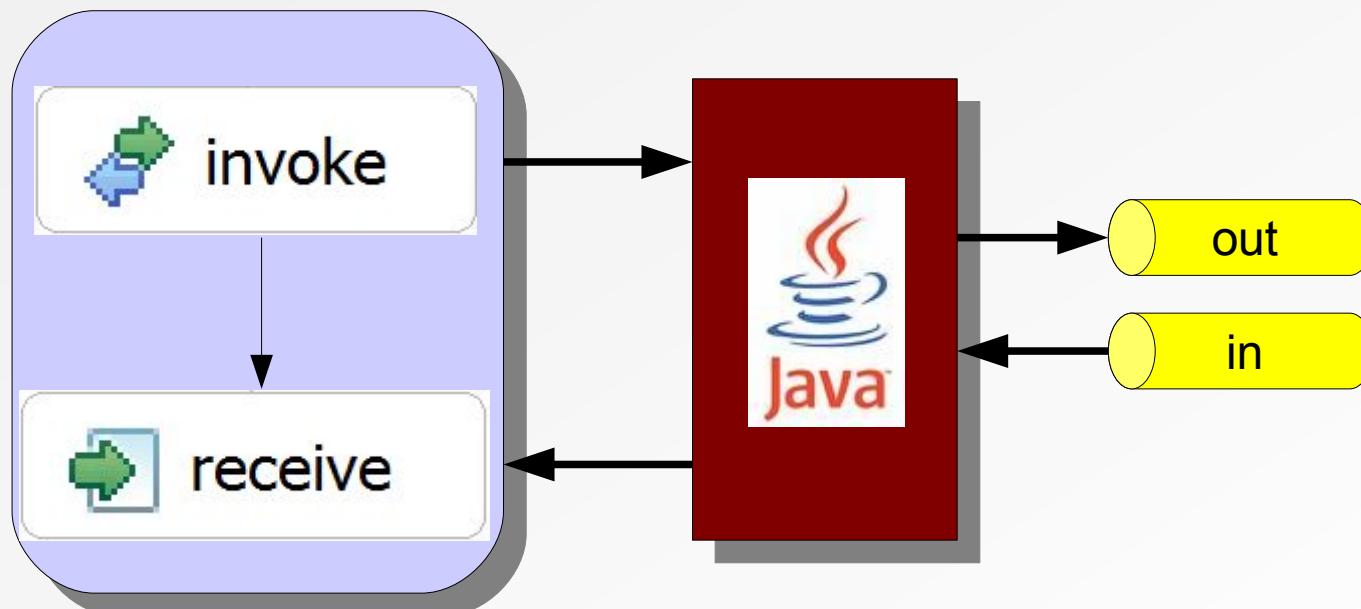


Développer un adaptateur en Java

- La première des chose à faire → le WSDL
 - Définir les types de données qui transitent
 - Définir les opérations à implémenter
- Génération du code Java (mode contract-first) JAXWS
 - CXF
 - JAXWS-RI
 - JBossWS
- Implémentation du Web Service
- Déploiement

Un adaptateur JMS

- Exemple – un adaptateur JMS
 - Web Service JAXWS-RI pour réception de la requête
 - Envoi sur file JMS
 - Réception avec un MDB
 - Callback de l'instance BPEL via un message externe



Exemple d'adaptateurs

- Exemple – un adaptateur JMS
 - Dans ce cas, l'adaptateur est asynchrone
 - Ne pas oublier de gérer la corrélation (via les custom corrélation)
 - Le message JMS doit porter l'information de corrélation (JMSCorrelationID)
 - Si impossible de faire transiter un même identifiant de BPEL jusqu'au message JMS, l'adaptateur doit gérer son propre mécanisme de corrélation
- Exemple – un adaptateur base de données
 - Cet adaptateur serait plutôt du type synchrone
 - Très simple à mettre en œuvre
 - Appel du Web Service
 - Exécution de la requête SQL
 - Retour des résultats dans la réponse synchrone

Les adaptateurs des moteurs BPEL

- Certaines implémentations de moteur BPEL fournissent par défaut des adaptateurs
- L'implémentation Oracle fournit
 - Base de données
 - JMS, MQ, AQ
 - Fichier
 - FTP
 - SAP, etc.
- Il faut être prudent avec les implémentations proposées
 - Pas toujours adapté au besoin très spécifique des projets
 - La performance n'est pas toujours au rendez-vous du fait de la généricité de ces composants

Bonnes pratiques

Les bonnes pratiques

- Créer des scopes
 - Avoir des variables locales
 - Lisibilité du code
 - Performance
- Éviter les manipulations trop complexes dans les *assigns*
 - Privilégier les feuille XSL
 - Performance
- Attention aux activités *empty*
 - Performance sur certaines implémentations

- Créer des processus asynchrones
 - Pas trop de niveau hiérarchique
 - Performance
- Privilégier les processus synchrones pour processus simples
 - Performance
 - Moins de contraintes
- Utiliser BPEL comme middleware uniquement
 - Logique de routage
 - Suite d'échanges entre différents services
 - Ne pas utiliser comme un langage de programmation