

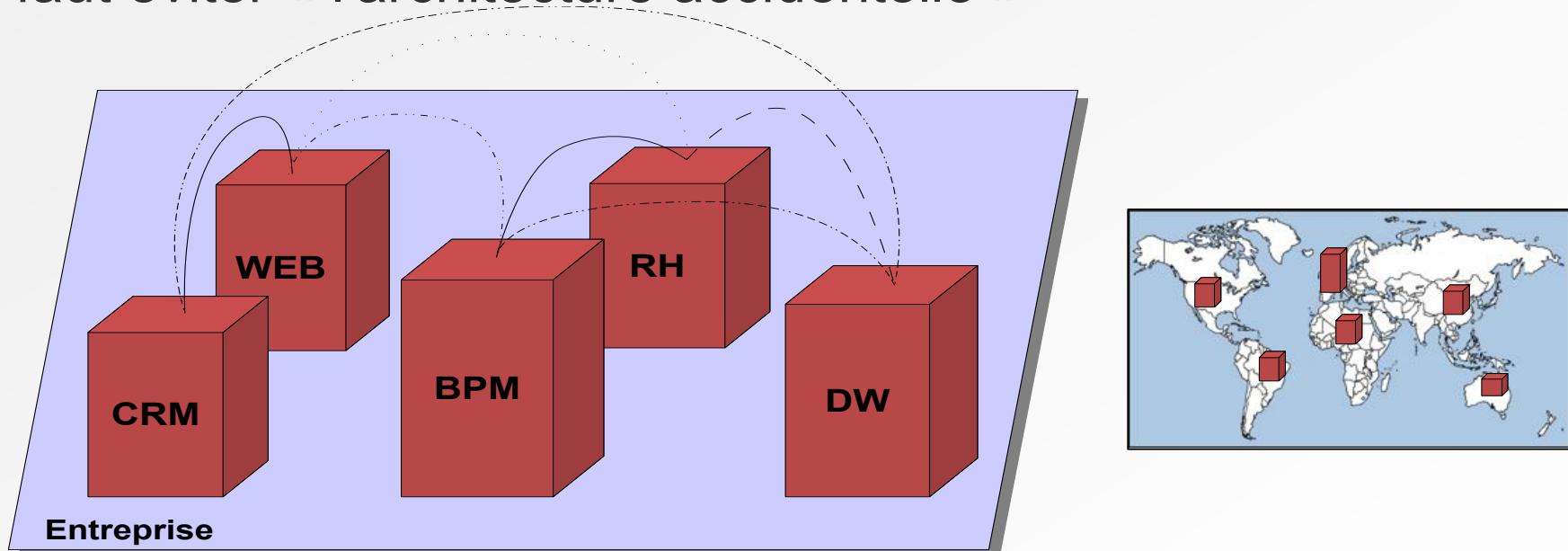
ESB

Enterprise Service Bus

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

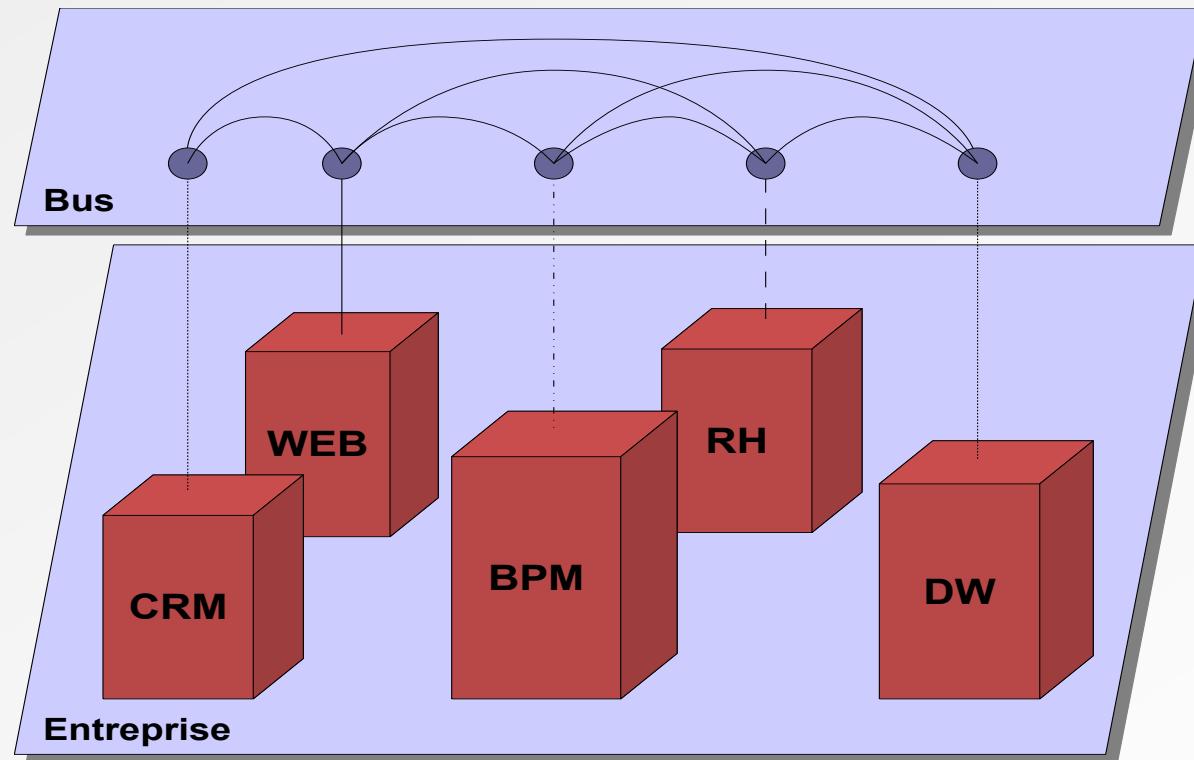
Communication entre des systèmes

- 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

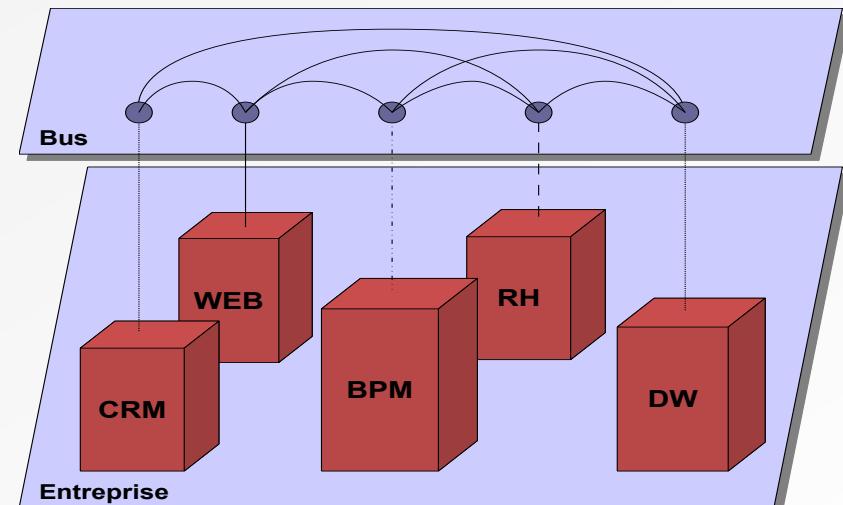
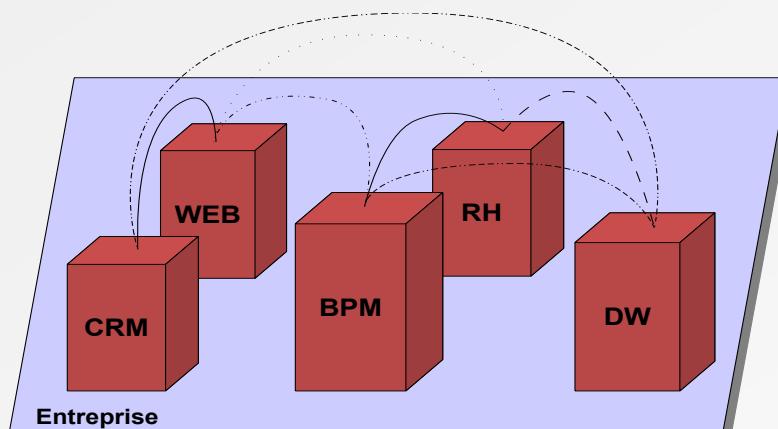


Couplage lâche (interactions)

- 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)

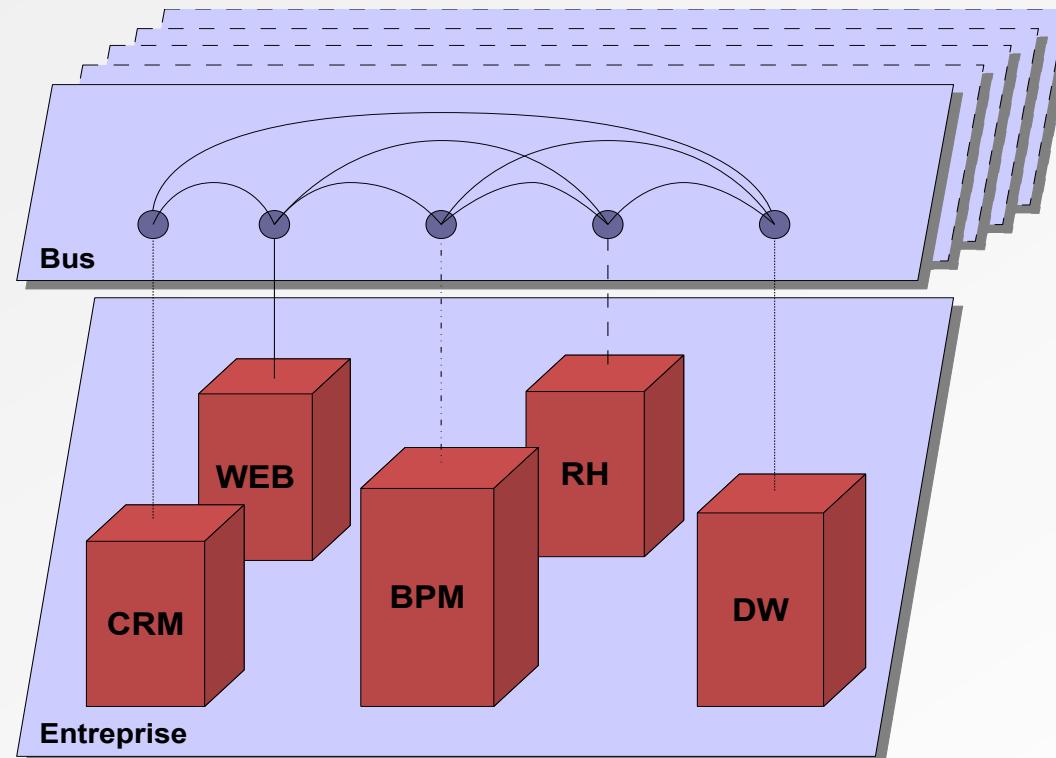
- Les ESB nous permettent de découpler les applications en fournissant une couche d'abstraction
- Les échanges de messages ne se font plus directement entre applications, c'est désormais l'ESB qui fait le médiateur



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

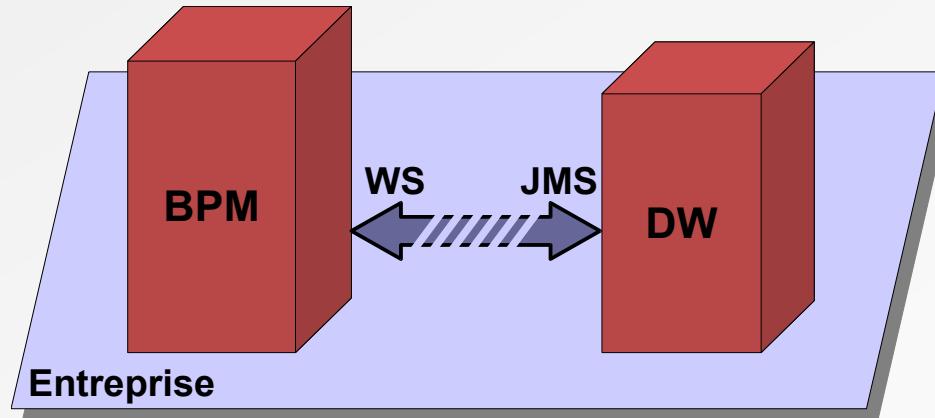
Distributivité

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



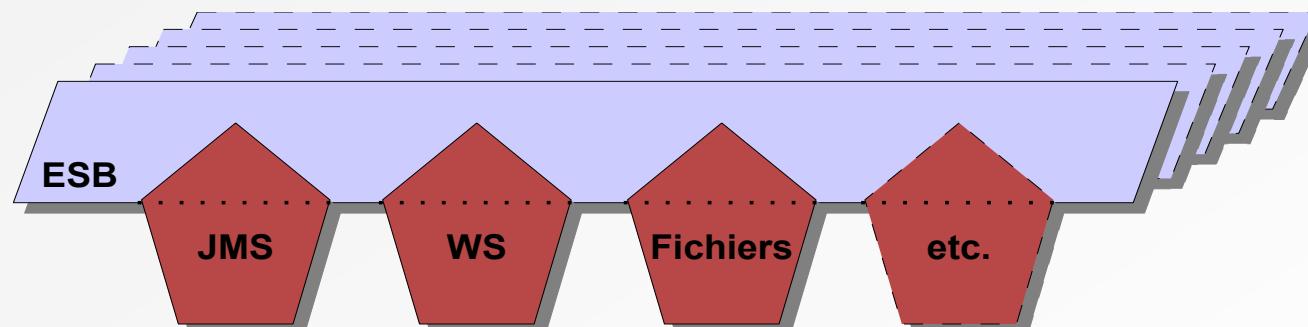
- La distributivité apporte
 - Scalabilité (ajout de nœuds)
 - Fiabilité (failover, load-balancing)
 - Facilité d'intégration (« si vous ne pouvez pas amener l'application au bus, amenez le bus à l'application »)

- 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

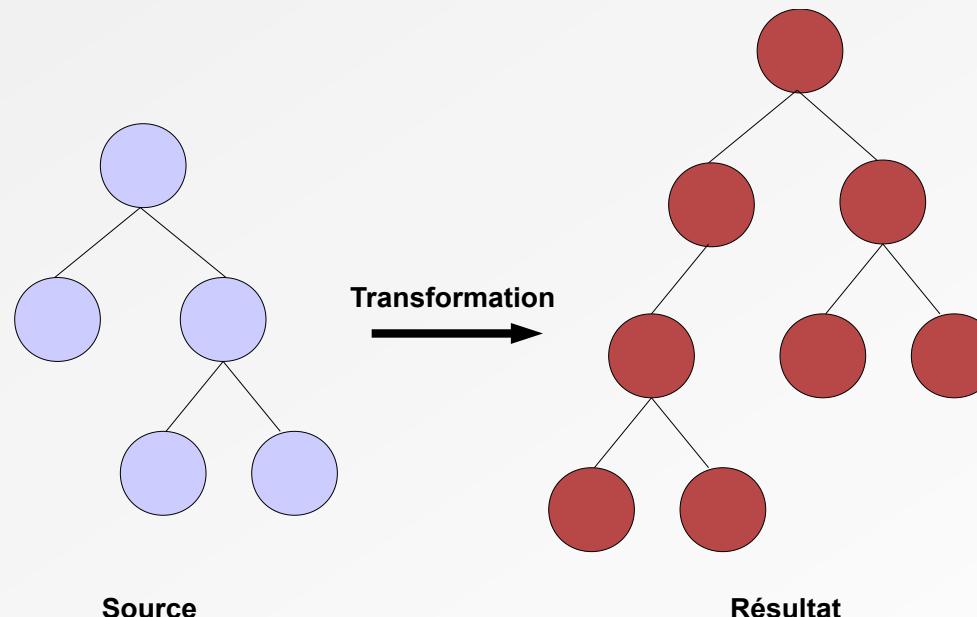
- L'ESB est capable de faire communiquer des applications utilisant des protocoles différents via les adaptateurs (ou connecteurs)
- Extensibilité
 - Installation d'adaptateurs tiers
 - Création d'adaptateurs spécifiques



Transformation et enrichissement

- Transformation

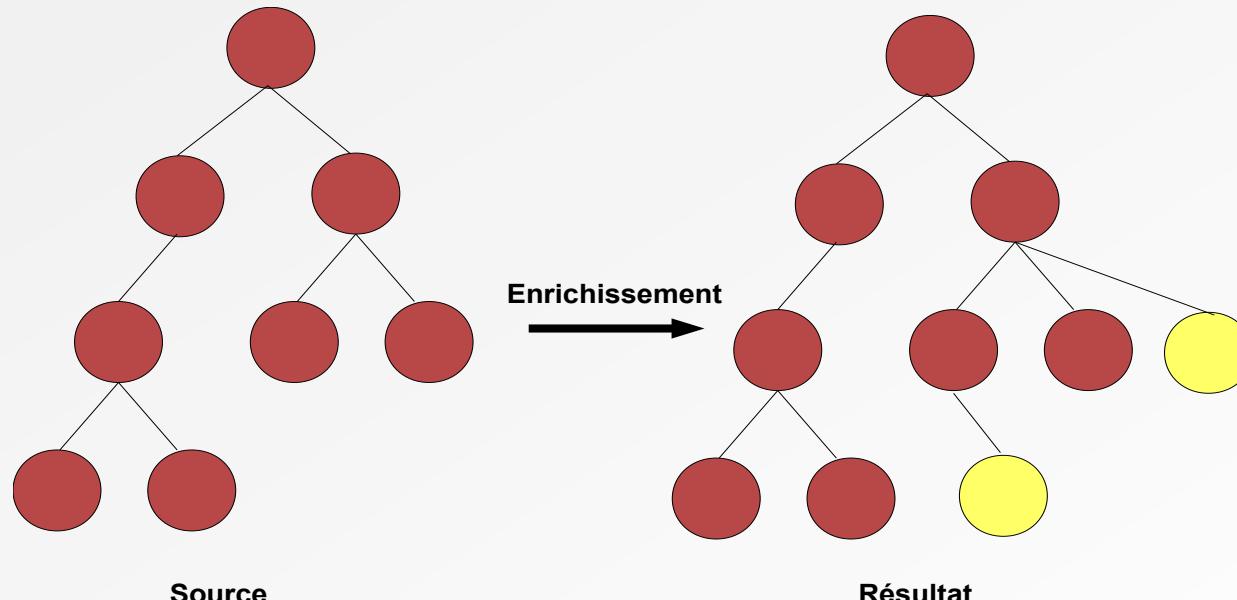
- Les applications ne travaillent pas avec un format commun
- Convertir un message initial vers un format cible
- Transformation de données
- XSLT (eXtensible Stylesheet Language Transformation)



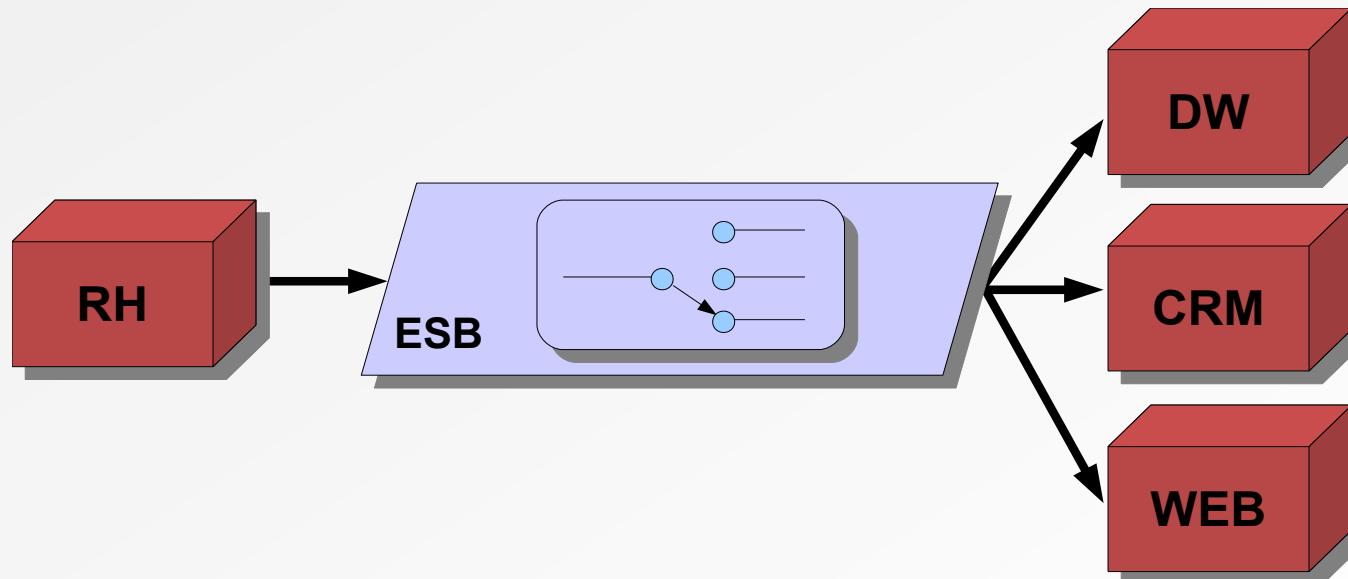
Transformation et enrichissement

- 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



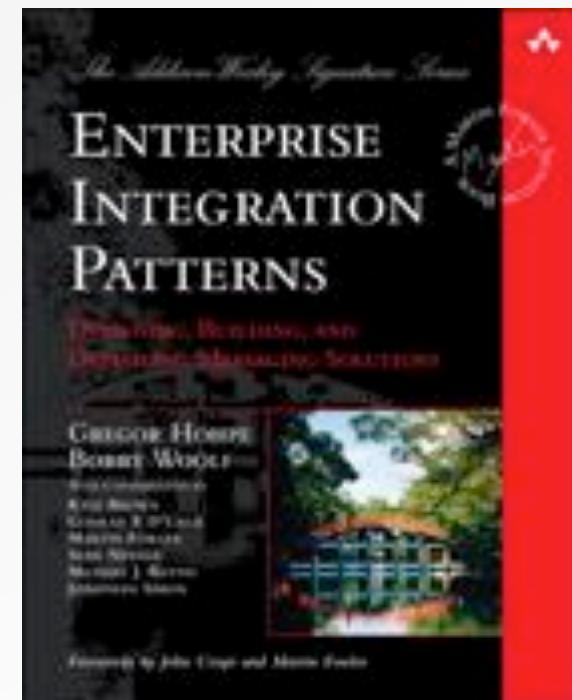
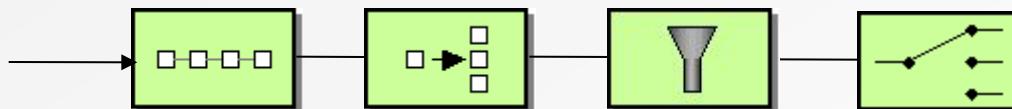
- Intégrité
 - Un ESB doit être fiable
 - Utilisation d'un serveur de messages intrinsèque
 - Application du pattern « *send and forget* »
 - L'ESB garantit que le message arrivera à destination
 - Système de gestion d'erreurs
- Sécurité
 - L'ESB est au cœur du SI, il est donc capital de le protéger
 - 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 :

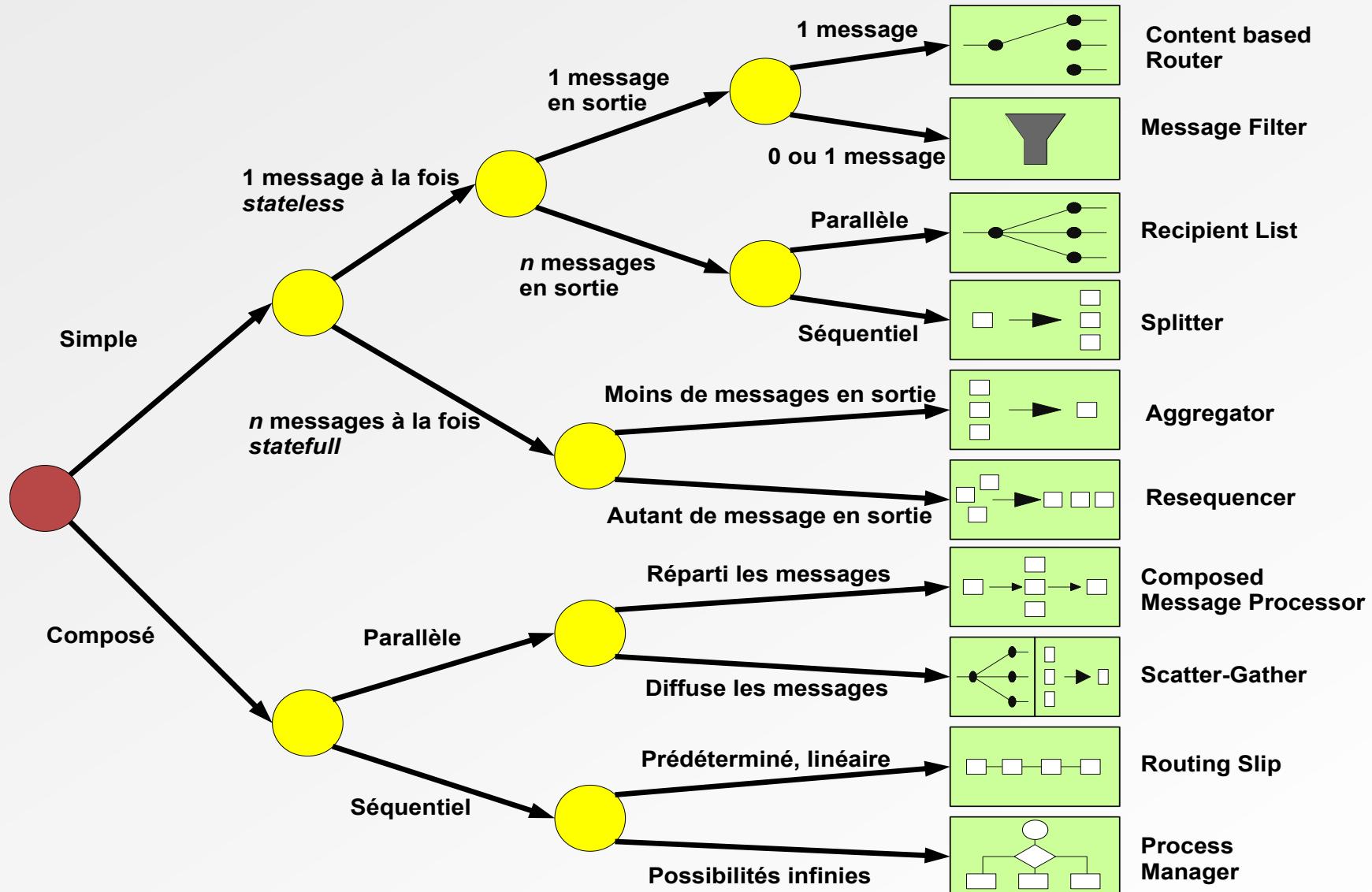
« La succession d'EIP définit une **Route** entre 2 **Endpoints** (un **producteur** et un **consommateur**) »
- La composition des EIP entre eux permet de faire émerger de nouveaux Patterns
- Les 50 EIP permettent de modéliser des solutions à l'ensemble des problématiques d'intégration au travers d'un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

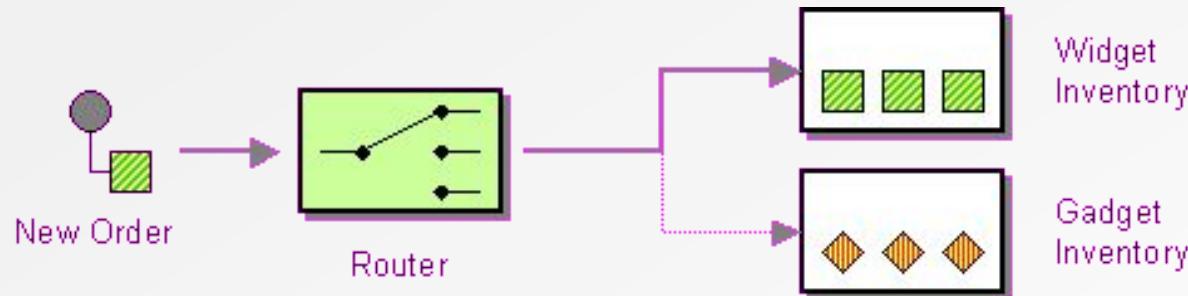
Introduction aux EIP de routage



- 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

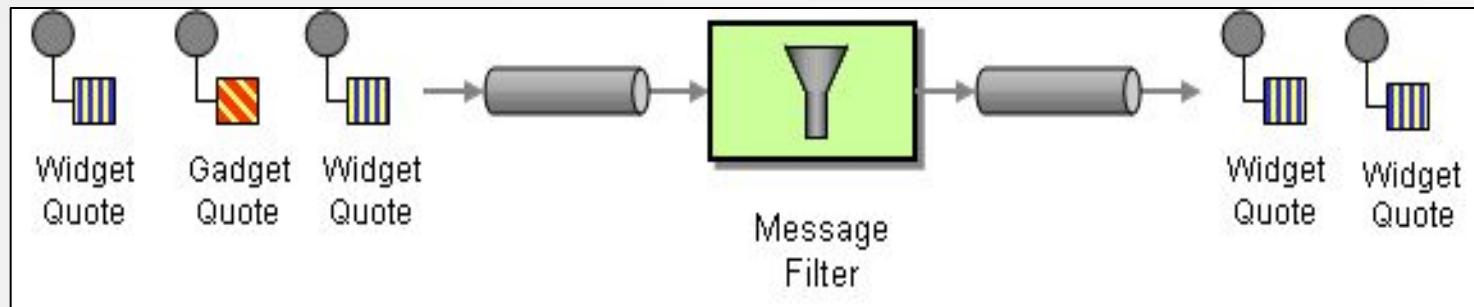
- Régulateur de débit
- Temporisateur
- Répartiteur de charge
- Diffusion vers n récepteurs
- Répétition

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



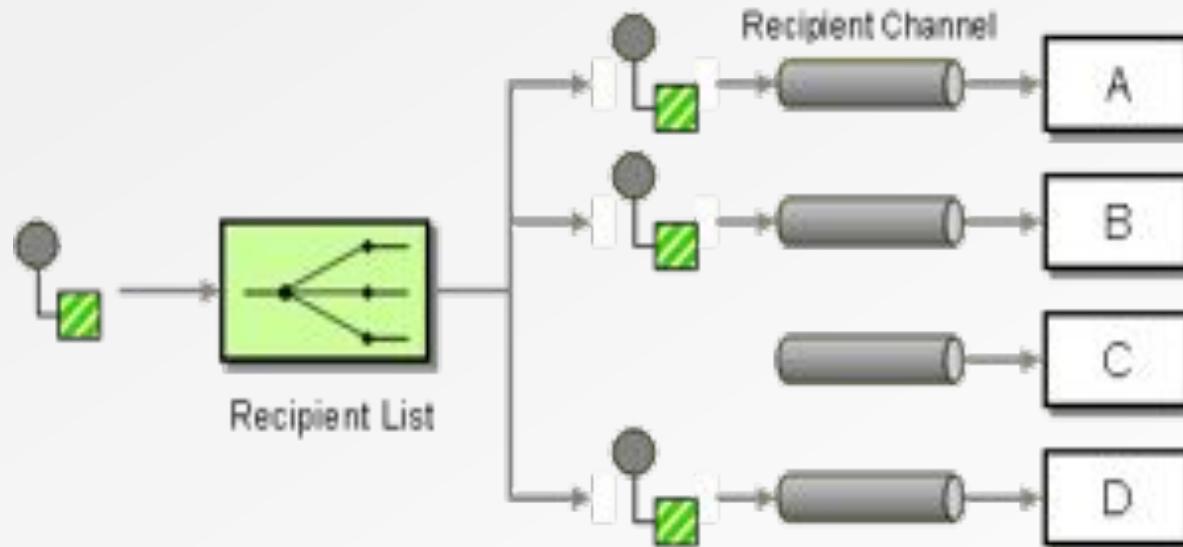
- Routage basé sur le contenu
- Examine le contenu du message (headers ou body)
 - Existence de champs
 - Valeur de champs

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



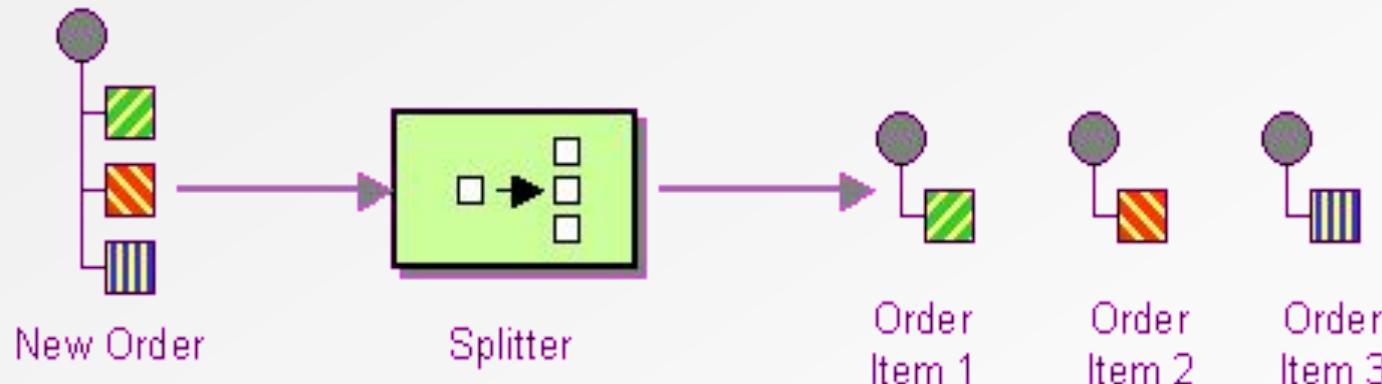
- Filtre les messages pour n'en laisser passer que certains selon un ensemble de critères configurables
- Permet d'éviter de surcharger un système
- Épure les flux de messages

- 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

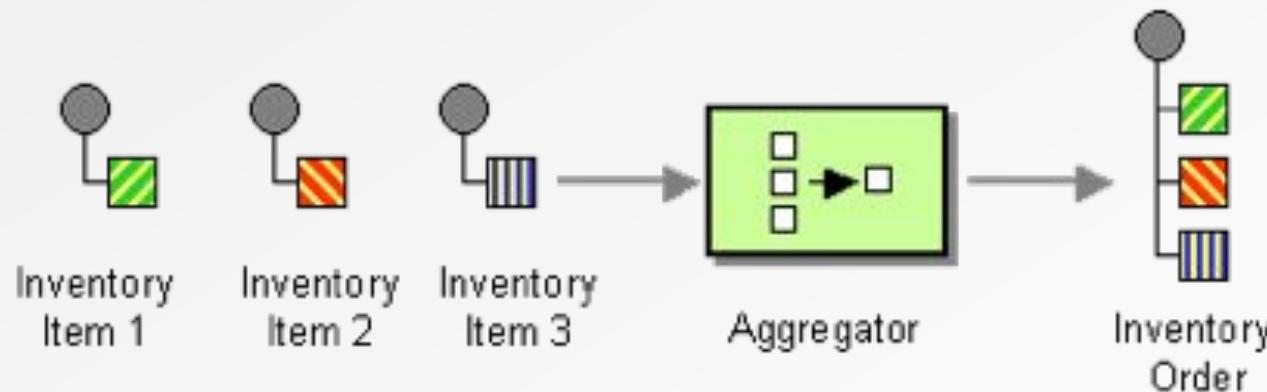
- Splitter
 - Comment traiter un message contenant plusieurs parties, chacune d'entre elles pouvant être traitées différemment ?



- 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

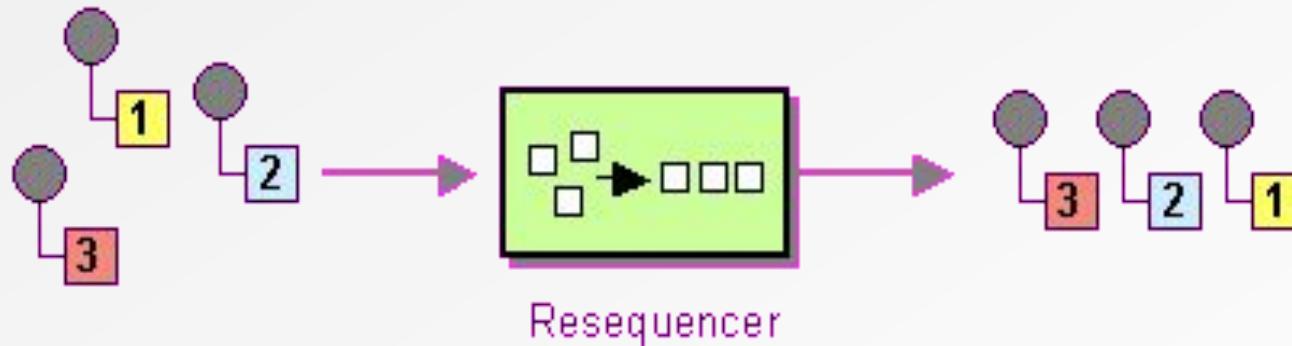
- Aggregator
 - Comment pouvons nous combiner des messages individuels (bien que liés) pour n'en former qu'un seul ?



- 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)

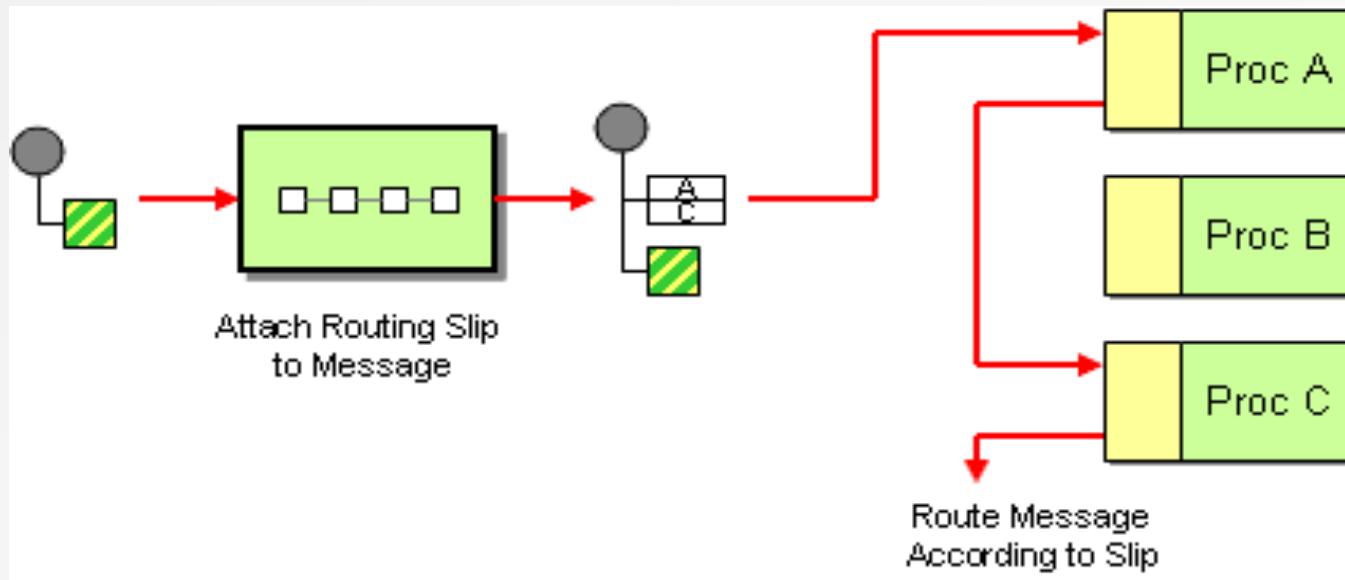
- Aggregator
 - Critères de complétion de la collection
 - *Quand publie-t-on le message résultant ?*
 - *Nombre de messages reçus*
 - *Délai d'attente*
 - Algorithme d'agrégation
 - *Comment réunit-on les messages entrants en un seul message ?*
 - EIP statefull
 - *Stockage des messages individuels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

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



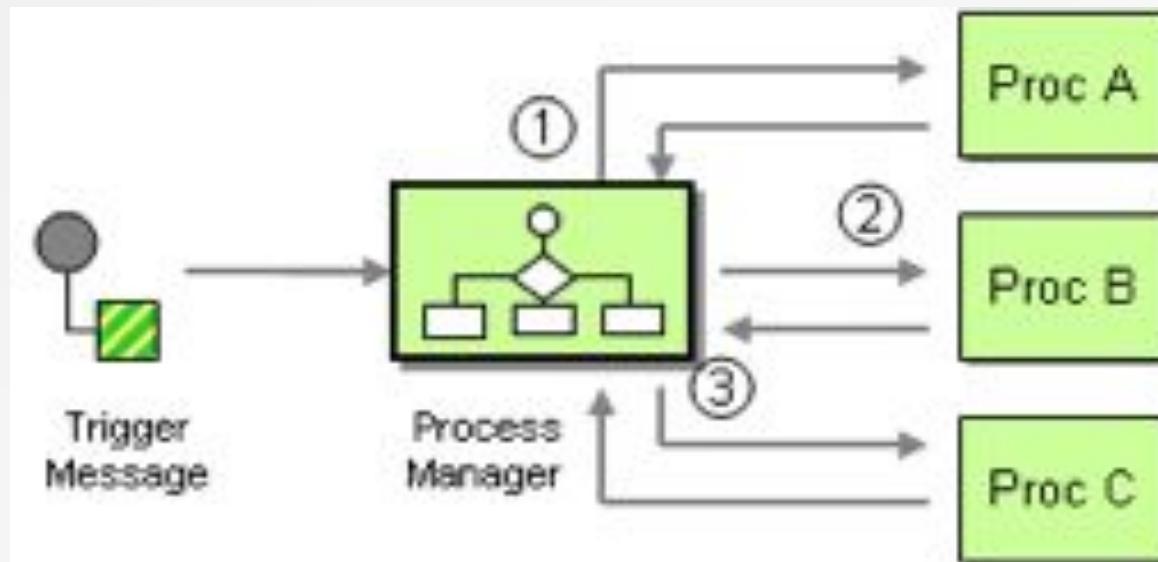
- Resequencer
 - Réception d'un flux de messages désordonnés
 - *Selon un critère métier (ex : numéro de facture)*
 - *Selon un critère technique (ex : id de corrélation)*
 - Rétention des messages dans un buffer interne (*statefull*)
 - Publication ordonnée des messages sur le channel de sortie (ce 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)

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

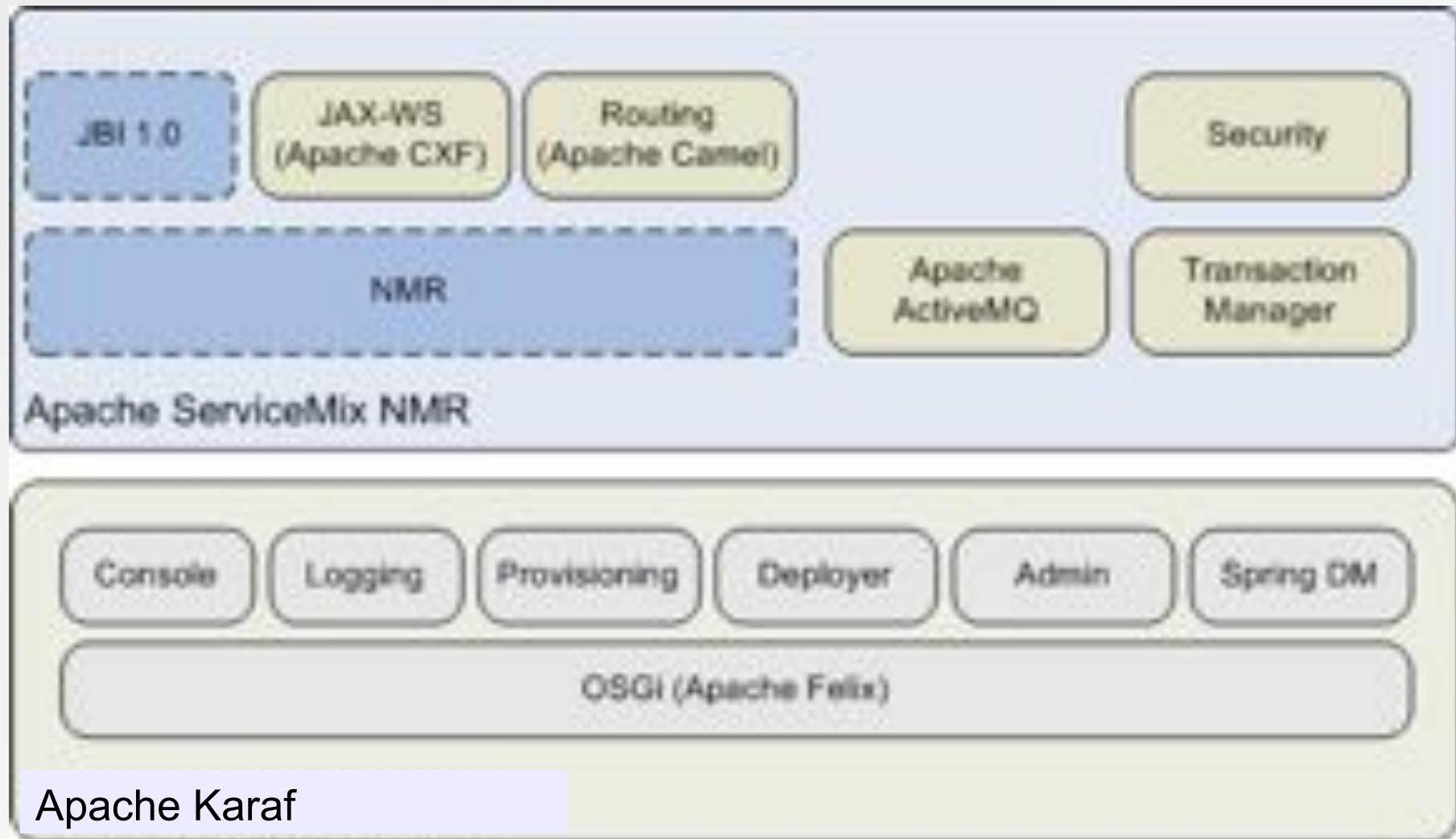


- ServiceMix s'appuie maintenant principalement sur trois technologies
 - OSGi
 - *Modularité*
 - *Classloading dynamique*
 - Camel
 - *Routage*
 - *Connectivité*
 - CXF
 - *Web Services*

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'])..."
```

- 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]
```

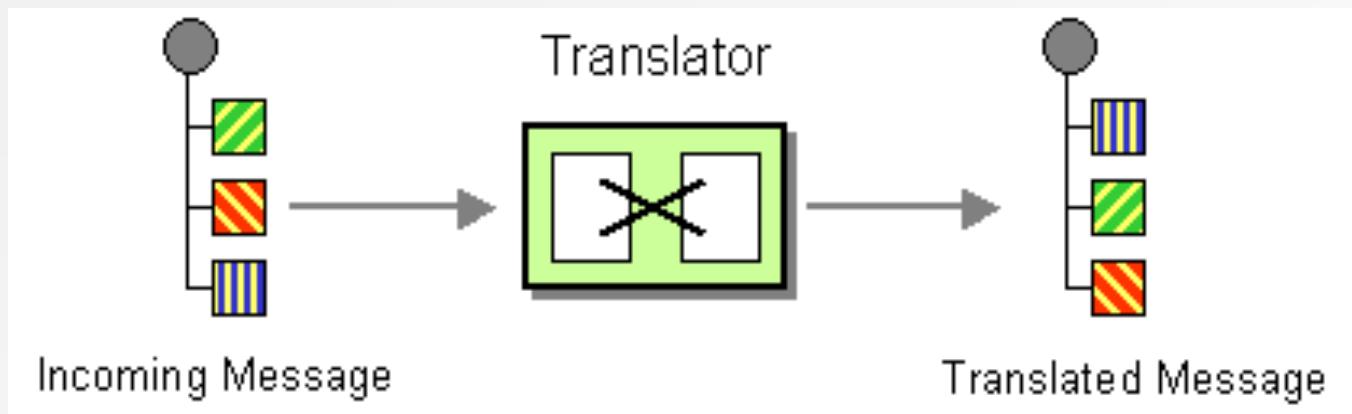
Exemple:

```
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

