



Xebia IT Architects SAS  
10/12 Avenue de l'Arche  
92419 Courbevoie Cedex  
Tél : 01 46 91 76 16  
[www.xebia.fr](http://www.xebia.fr)

# SOA pas à pas

## Une expérience réelle

• • • • • • • • •

*Copyright © Xebia – 2006*



# SOA pas à pas

## *Une expérience réelle*

### Préface

*Nous sommes un certain nombre, quadras et quinquas de l'informatique à assister, pantois, à la prolifération de spécialistes d'un genre nouveau : les « spécialistes SOA ».*

*Nous avons de la chance.*

*Ces gourous auto proclamés que certains amateurs de bon vin qualifieraient d'AONC (Appellation d'Origine Non Contrôlée), ne sont pas avares de leurs conseils et leur nombre grandit aussi rapidement que celui des « spécialistes eCommerce » au moment du gonflement de la bulle Internet des années 2000.*

*A quels saints doit-on se vouer ? L'affaire n'est pas simple et nous n'avons pas la prétention de vous apporter une réponse toute faite.*

*L'humilité est l'une des 10 valeurs en vigueur chez Xebia*

*Ce guide n'a pas d'autre ambition que de partager une expérience concrète de la mise en place de SOA par des consultants de Xebia.*

*Il dresse, pour vous, une liste des chantiers que nous pensons être incontournables pour les entreprises engagées sur la voie des Architectures Orientées Services.*

*Nous espérons que vous prendrez plaisir à le lire.*

**Luc Legardeur**

**Président**

**Xebia**

## Table des matières

<b>Préface</b> .....	<b>2</b>
<b>Table des matières</b> .....	<b>3</b>
<b>Table des illustrations</b> .....	<b>3</b>
<b>1 Introduction</b> .....	<b>4</b>
1.1 Enjeux d'une SOA .....	4
1.2 Les deux approches : Top Down ou Bottom Up .....	4
<b>2 Démarche</b> .....	<b>6</b>
2.1 Conception orientée Services .....	6
2.2 Interconnexion technique .....	8
2.3 L'intermédiation fonctionnelle .....	10
2.4 Socle de développement : industrialisation et solidification .....	12
2.5 Mise en place d'un catalogue de services (Registry) .....	14
2.6 La gestion des paramètres (MDM) .....	15
2.7 La nécessité des formats pivots .....	16
2.8 La contextualisation des services .....	18
2.9 La supervision et la gestion de la sécurité .....	20
2.10 Batch et SOA .....	22
2.11 Positionnement de certaines briques concourant à SOA .....	23
<b>3 Conclusion</b> .....	<b>25</b>
3.1 Appréhender SOA .....	25
<b>Annexes</b> .....	<b>26</b>
3.2 Nomenclature .....	26
3.3 Glossaire .....	26

## Table des illustrations

Figure 1 - Bottom-up & Top-down .....	5
Figure 2 - Echange entre services .....	8
Figure 3 - Intermédiation fonctionnelle .....	10
Figure 4 - Principes d'un MDM .....	15
Figure 5 - Sans format pivot, multiplication des formats .....	17
Figure 6 - Formats pivots par domaine .....	17
Figure 7 - Contextualisation des services .....	19
Figure 8 - BAM .....	21
Figure 9 - Batch et fil de l'eau .....	22
Figure 10 - Fonctions d'un EAI / ESB .....	24

# 1 Introduction

## 1.1 Enjeux d'une SOA

Une entreprise, ou plus communément une organisation est un « système » au sens de la théorie générale des systèmes or, comme l'a montré l'abondante littérature sur le sujet, un système est confronté à quatre défis majeurs :

- **L'interaction** : un système échange avec d'autres voire avec lui-même (feedback), Pour une entreprise cela signifie les partenariats, les relations clients & fournisseurs, etc.
- **La totalité** : le système doit être vu dans son ensemble et ne doit pas être réduit à l'une de ses parties.
- **L'organisation** : les relations entre les composants d'un même système sont tout aussi importantes que les composants eux même. Il ne suffit pas de décrire statiquement la cartographie d'une entreprise, la dynamique des échanges est primordiale à appréhender.
- **La complexité** : elle est liée à l'environnement dans lequel le système évolue, à son organisation, à ses objectifs.

SOA a pour objectif de répondre à ces quatre enjeux :

- Faciliter l'**interaction** par l'exposition de services en s'appuyant sur des concepts forts comme l'interopérabilité et la réutilisation.
- La définition des services dans SOA impose une réelle **réflexion métier** prenant en compte la stratégie de l'entreprise dans sa globalité.
- L'architecture de services doit être en phase avec l'organisation et traduire la dynamique des échanges afin de définir des **services réutilisables**.
- Les S.I. sont de plus en plus complexes, d'une part à cause d'évolutions technologiques incessantes mais aussi à cause de changements permanents dans les organisations, dans la législation, etc. En rendant flexible le S.I., SOA permet de mieux **gérer ces changements** tant technologiques, métiers, organisationnels que réglementaires.

## 1.2 Les deux approches : Top Down ou Bottom Up

La question de la bonne approche a donné naissance à bien des écrits et des théories chacune sous tendant des positions, selon nous, trop dogmatiques.

Le point de vue de Xebia dans ce domaine est simple :

Bien qu'une approche Top-Down soit préférable dans le cadre d'une démarche SOA, elle n'est cependant que rarement possible car elle signifie une refonte de tout ou partie du S.I, jugée bien souvent trop coûteuse et trop risquée. Il est donc bien souvent obligatoire de passer, au moins temporairement (à l'échelle d'un schéma directeur cela signifie quelques années), par une phase de conception ascendante (Bottom-up).

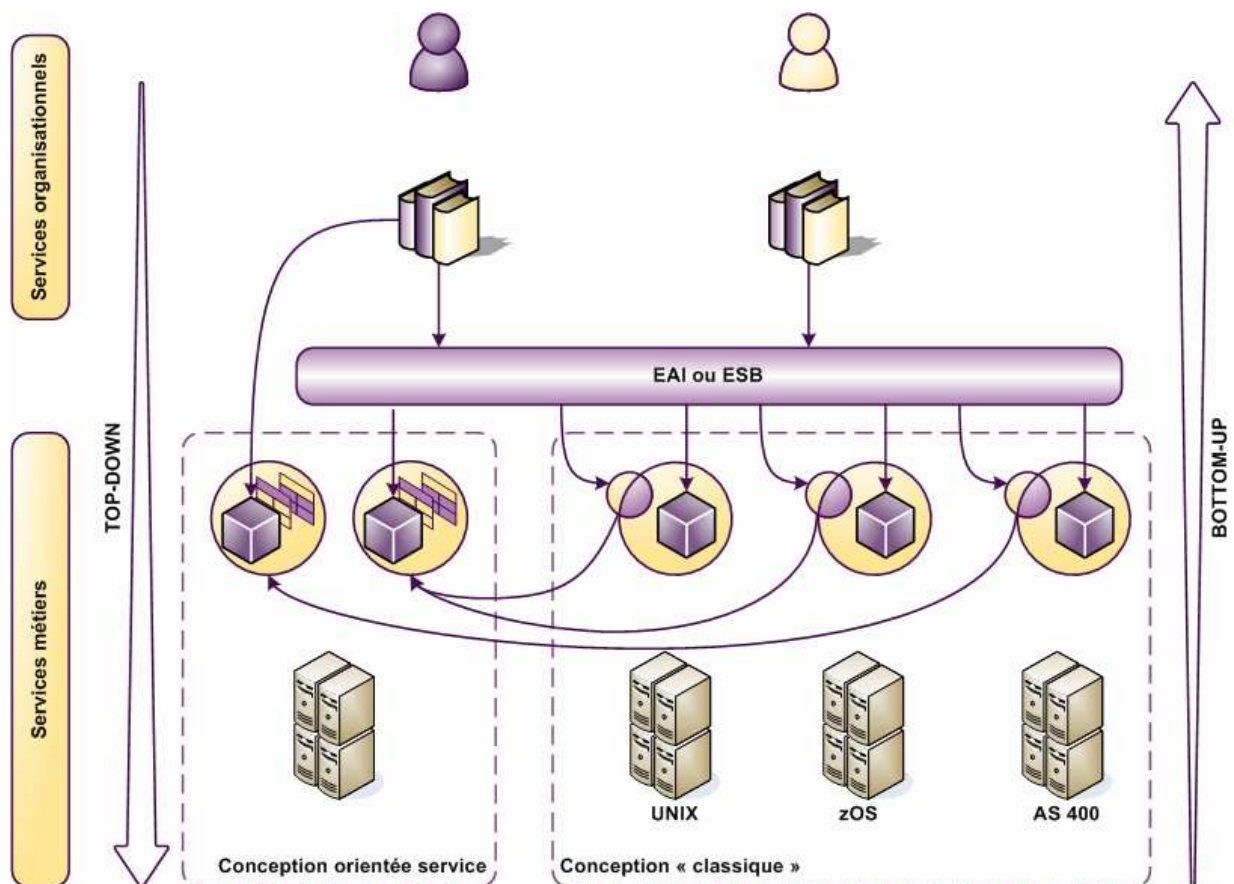
C'est dans les approches Bottom-up que les technologies de types EAI et ESB prennent tout leur sens puisqu'elles permettent dans un premier temps de faire communiquer, entre elles, les différentes briques applicatives tout en garantissant leur totale indépendance et étanchéité les unes vis-à-vis des autres.

Cependant, l'approche Bottom-Up seule ne permet pas de mettre en place une SOA. Il s'agira tout au mieux d'une « mise en mode services ».

Ces services seront vus localement et ne s'inscriront pas dans une **Architecture Orientée Services à l'échelle de l'entreprise**.

La bonne solution est donc de **mixer les deux démarches**, Top-Down pour une portion du S.I. qui est en cours de refonte (ou à refondre) et Bottom-Up pour le reste, ce qui concourt au bon fonctionnement de l'ensemble au quotidien.

**La conclusion de ce chapitre d'introduction est donc bien la suivante : une SOA n'a aucun sens si l'on ne veut pas impacter l'existant. Le mot-clef du trigramme SOA est donc bien Architecture et pas Services. En faisant une métaphore avec le bâtiment, on ne fait pas de l'architecture en repeignant les murs et en changeant la tuyauterie.**



**Figure 1 - Bottom-up & Top-down**

## 2 Démarche

### 2.1 Conception orientée Services

---

#### 2.1.1 Objectifs & démarche

Nous n'allons pas ici justifier la démarche de conception en elle-même qui est une évidence dans la mise en place de SOA.

La démarche de conception des services dans SOA a pour objectifs :

- D'**identifier** les services ;
- De **catégoriser** les services ;
- De **regrouper** les services ;
- De **formaliser** les services.

Le premier chantier servira à **identifier** les services concourant à la cible d'architecture. Ces services peuvent être issus d'une conception « from scratch » (refonte, re-conception) ou d'interfaces masquant l'existant.

Une fois identifiés, les services doivent être « **typés** », on distingue trois grandes catégories :

- **Services organisationnels** : généralement issus des cas d'utilisation ou actes de gestion métiers (par exemple : déclarer un sinistre, ouvrir un compte bancaire, passer une commande, etc.). Les règles contenues dans ces services sont intimement liées à la structure de l'entreprise et donc plus sujettes à changement. Les services organisationnels sont les services de haut niveau. Ce sont eux qui orchestrent d'autres services, services organisationnels ou services métiers. Les services organisationnels sont ceux exposés à un utilisateur via une IHM.
- **Services métier** : contenant les règles métiers ou liés à la législation donc réputés stables. Ces services ne sont normalement pas accessibles directement par les utilisateurs. Ils sont orchestrés par les services organisationnels. Leur complexité est limitée et ils doivent répondre à des besoins élémentaires simples. On trouve dans cette catégorie par exemple « lire contrat » « recherche client », « calculer montant facture ».
- **Services techniques** : purement techniques et dont l'existence est pilotée et décidée par l'informatique et non par le métier.

Pour éviter d'avoir une architecture avec plusieurs centaines voire milliers de services « flottants », il est nécessaire de les **regrouper au sein de composants**. Généralement on considère qu'un composant contient une dizaine à une quinzaine de services maximum. Pour plus de souplesse, ces composants peuvent aussi être regroupés au sein d'autres composants et ainsi de suite, toujours en respectant la métrique de 10 à 15.

Pour adresser tous ces enjeux, une phase de modélisation est donc nécessaire. Cette phase n'est envisageable qu'avec l'aide d'un outil et une méthode de modélisation ad'hoc.

Les enjeux de la modélisation sont :

- **Eviter la redondance dans le système** : Si chaque responsable technique d'un système décide de lui-même ce qu'il veut exposer, de la redondance s'installera à moyen terme. Seule une vue d'ensemble du S.I. permet de juger de la pertinence d'un service.
- **Assurer la réutilisabilité** : Pour être rentable, un service doit être utilisé plus d'une fois. Seuls les experts métiers sont à même de garantir cette réutilisation.
- **Impliquer les utilisateurs** dans le S.I.
- **Définir la cible d'urbanisation** et la trajectoire pour l'atteindre.

### 2.1.2 Critères de vigilance

<b>Méthodologie de conception</b>	Une méthodologie appropriée doit être choisie pour identifier les services.
<b>Choix de l'outil de modélisation</b>	<p>Une étude doit être menée pour vérifier les points suivants :</p> <ul style="list-style-type: none"> <li>▪ Compatibilité UML,</li> <li>▪ Générateurs déjà disponibles,</li> <li>▪ Facilité de création de nouveaux générateurs,</li> <li>▪ Possibilité de faire du reverse engineering.</li> </ul> <p>Ces points permettront d'industrialiser le passage des modèles de conception à la cible technologique (approche MDA).</p>

## 2.2 Interconnexion technique

### 2.2.1 Objectifs & démarche

Nous nous accordons tous pour dire que SOA consiste à exposer en mode service des programmes, des transactions ou des « méthodes / opérations » à l'extérieur de l'application qui les héberge en vue de leur réutilisation. Il est primordial de comprendre que **cette exposition est une vision informatique**, ce qui ne permet pas d'assurer la pérennité à long terme des services exposés.

En effet ceux ci ne seront pas nécessairement alignés sur la stratégie business de l'entreprise. Cependant, cette première étape est utile pour construire le socle technique sur lequel s'appuiera ultérieurement une véritable SOA.

Ce socle technique se découpe en trois composantes majeures :

- Les **services** ;
- La **couche de transport** ;
- Les **connecteurs**.

Ce qui se résume sur le schéma suivant :

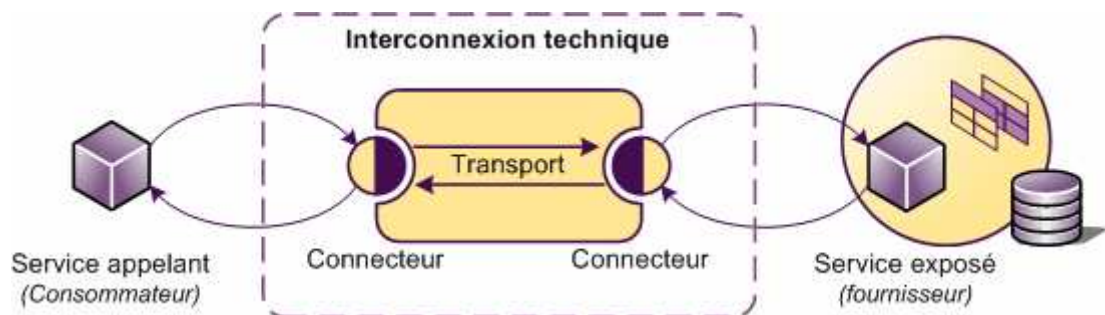


Figure 2 - Echange entre services

Un projet SOA ne se résume pas à une composante EAI / ESB ou WebServices qui ne sont que des moyens de la mise en œuvre.

Il y a lieu d'étudier, en fonction du contexte du S.I., la ou les meilleures solutions à mettre en place. Par conséquent, le transport n'est pas nécessairement un M.O.M. (Middleware Orienté Messages). Il peut parfaitement s'agir d'une connexion http sur laquelle circulent des messages SOAP ou encore d'accès EJB Remote (RMI/IIOP).

Par exemple, la décision de prendre un EAI est liée aux choix stratégiques de l'entreprise en termes d'interconnexion. La plupart des EAI du marché offrent nativement bon nombre de connecteurs pour accéder aux systèmes Legacy (MVS, AS400, ...) mais on trouve aussi des solutions permettant d'exposer par exemple des transactions CICS ou IMS en WebServices.

Les critères de sélection d'un EAI sont nombreux (asynchronisme, routage, existence de connecteurs, outil de BPM, ...) et nous ne nous étendons pas sur le sujet qui a donné lieu à de nombreuses études chez Xebia ou ailleurs.



Quelque soit le choix effectué, il est nécessaire de **masquer aux services les problématiques d'interconnexions techniques**. En effet, le point de vigilance, véritable fil rouge qui guide toute démarche SOA, est la réversibilité technologique.

L'intermédiation technique doit donc pouvoir être changée sans impact sur les services. Pour ce faire il existe de bonnes pratiques de génie logiciel (design patterns) qui s'implémentent par simple ingénierie et ne nécessitent pas l'acquisition de logiciels coûteux.

### 2.2.2 Critères de vigilance

<b>Dissociation code métier / code technique d'exposition</b>	Les services identifiés ne doivent pas connaître par quels mécanismes techniques ils invoquent d'autres services ou sont eux même invoqués.
<b>Connecteurs</b>	Les connecteurs doivent être génériques, donc agnostiques vis-à-vis du métier et surtout réutilisables donc au sein d'un socle technique (framework).
<b>Sécurité des messages</b>	La sécurité (perte de messages, preuves, intégrité, ...) doit être traitée par les connecteurs.

## 2.3 L'intermédiation fonctionnelle

### 2.3.1 Objectifs & démarche

L'intermédiation fonctionnelle est nécessaire pour aborder deux points fondamentaux de toute démarche SOA :

- La **transcodification** :
  - Translation d'identifiants ;
  - Traduction de référentiels ;
  - Localisation des services et du système propriétaire de l'objet métier.
- L'**agrégation** :
  - Exposition de services consolidant des informations de plusieurs domaines fonctionnels.

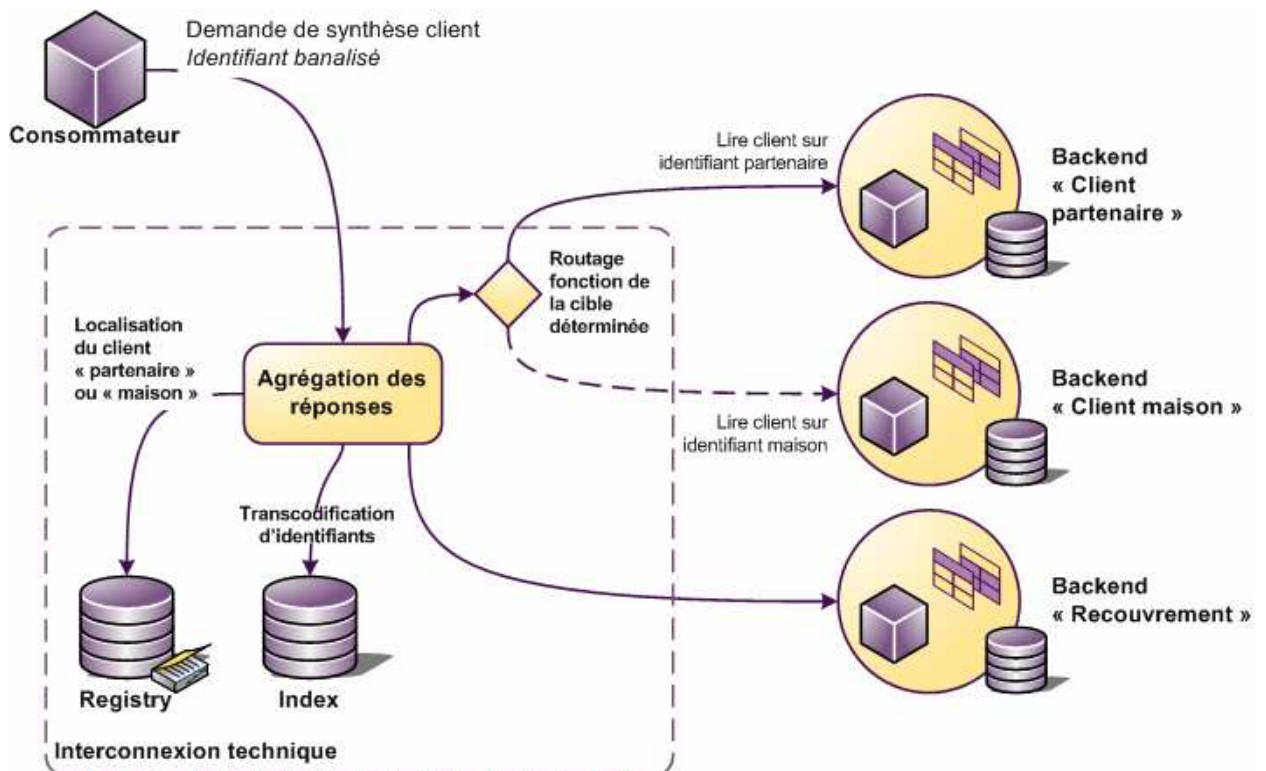


Figure 3 - Intermédiation fonctionnelle



Chaque système travaille avec ses propres identifiants métier, une **transcodification** est donc nécessaire pour permettre une mise en commun de services (on parle de translation d'identifiant). Cette translation consiste à associer pour un identifiant d'un système (un client, un contrat, ...), un identifiant banalisé et unique sur tout le S.I. dans le cadre d'un **index**.

L'**agrégation fonctionnelle** consiste à mettre à disposition des services unifiant les réponses de plusieurs autres services. L'exemple typique est celui de la synthèse client qui consiste à présenter de manière consolidée à la fois les informations du client mais aussi des informations sur son compte ou ses contrats. Cette agrégation prend à sa charge les services transverses couvrant plusieurs domaines.

Ces services trouvent leur place dans des outils d'EAI / ESB ou alors en java natif (La JSR-207 traite tout particulièrement de l'orchestration de service en Java).

### 2.3.2 Critères de vigilance

<b>Architecture &amp; conception fonctionnelle</b>	<p>Un architecte fonctionnel doit être clairement identifié et responsable du chantier de conception fonctionnelle.</p> <p>Les services d'intermédiation fonctionnelle doivent être conçus par les équipes fonctionnelles et donc spécifiés par eux.</p>
--	--

## 2.4 Socle de développement : industrialisation et solidification

---

### 2.4.1 Objectifs & démarche

La clef de la réussite d'une architecture, quelle soit SOA ou pas, mais à plus forte raison si elle est SOA, réside dans la solidité du socle technique. Ce socle doit être maîtrisé tant dans son développement que son exploitation.

Les développements doivent être séparés en deux domaines distincts :

- Les composants techniques ;
- Le métier.

Les composants techniques, généralement regroupés sous la terminologie framework, doivent permettre un **développement rapide** et notamment **masquer au développeur « métier » toutes les problématiques techniques** comme par exemple :

- L'accès à un service distant (WebService, M.O.M. ou EJB) ;
- La gestion des transactions ;
- La sécurité ;
- La gestion des instances d'objets ;
- L'accès aux données ;
- Pré & post traitements de services ;
- L'IHM ;
- Etc.

L'ensemble de ces points doit être traité dans un Document d'Architecture Applicative et Technique (D.A.A.T.). Les principes clefs sont d'obtenir un framework garantissant :

- La **facilité d'extension** (ajout de nouvelles fonctionnalités techniques) ;
- Un **système déclaratif** pour les composants métiers (ajout de nouveaux composants par simple paramétrage) ;
- Un **haut niveau de paramétrage** (transaction déclarative par exemple et donc non gérée au niveau du code) ;
- Une **contextualisation des services** (Cf. paragraphe 2.8, « La contextualisation des services » page 18) ;
- La **réversibilité technologique** (dans la limite du choix du langage d'implémentation).

On comprend aisément qu'une très haute technicité est requise (et c'est là, l'un des métiers de Xebia). Ce framework peut être développé en interne et / ou s'appuyer sur des framework du marché. Quelque soit le choix, la réalisation du D.A.A.T. est essentielle même si un framework du marché est retenu ne serait-ce que pour justifier ce choix. En effet, le D.A.A.T. a pour vocation première la justification des décisions d'architecture.



La mise en œuvre d'un socle technique doit traiter aussi de l'**exploitabilité des applications** notamment la gestion des alertes sur incidents et la supervision.

Une **plate-forme d'intégration** est également nécessaire. Elle comprend au minimum :

- Un outil de gestion de bugs ;
- Un composant d'intégration continue ;
- La documentation projet (manuel du développeur, javadoc du framework, ...)
- Un référentiel commun pour les ressources partagées (bibliothèques – jar, utilitaires, ....) ;
- Etc.

Quelque soit la démarche projet choisie, il n'est jamais mauvais de rappeler que les **tests unitaires** ne sont pas seulement nécessaires mais un passage obligatoire (et que trop souvent réduits à leur plus simple expression).

Parmi ces tests unitaires, dans une architecture SOA, nous retrouvons la notion des **bouchons**.

Pour des contraintes projets, les services fournisseurs ne sont pas nécessairement finis en même temps que les consommateurs. Pour que chaque développement puisse se faire de manière la plus isolée possible – en attendant l'intégration de bout en bout – il est nécessaire dès les phases amonts de définir les bouchons permettant cette isolation.

Bien plus que les bouchons, les **moyens techniques de gestion de ces bouchons** doivent être étudiés en détail. Cela inclut :

- La génération ;
- La gestion du cycle de vie (versionning) ;
- La production a plus ou moins grande échelle d'un nombre représentatif de cas de tests.

La démarche de solidification couvre donc :

- L'industrialisation des **développements** ;
- L'industrialisation des **tests unitaires** ;
- L'industrialisation des **tests d'intégration**.

## 2.5 Mise en place d'un catalogue de services (Registry)

### 2.5.1 Objectifs & demarche

Le catalogue de service est un incontournable pour maîtriser les services. Ce catalogue offre plusieurs fonctionnalités et doit au minimum :

- **Faciliter le fonctionnement de l'infrastructure d'échanges :**
  - Assurer la correspondance entre les différents noms que peut porter chaque service dans les différents sous-systèmes ;
  - Routage ;
  - Transcodification et uniformisation des codes (y compris les codes erreurs).
- **Consolider la connaissance métier :**
  - Être la référence unique portant la connaissance des services de l'entreprise ;
  - Porter la connaissance des formats d'échanges ;
  - Distinguer les services qui nécessitent une réponse de ceux qui n'en attendent pas ;
  - Identifier l'appartenance fonctionnelle de chaque flux ;
  - Gérer les versions.

Le catalogue assure donc la **cohérence entre les différentes nomenclatures** des applications.

Il traite aussi la dimension routage à savoir qu'à partir des informations de contexte il offre la possibilité de déterminer le service cible, le mode d'accès (M.O.M., Web Services, ...), les ressources associées (nom de la file MQ, URL d'accès).

Le catalogue doit être et rester l'**unique point de référencement** des services.

### 2.5.2 Critères de vigilance

<b>Modélisation &amp; conception du catalogue</b>	Le catalogue doit être modélisé par un architecte fonctionnel.
<b>Couverture des besoins</b>	<p>La mise en œuvre d'un catalogue dans sa totalité est un projet d'envergure. Il est possible d'en restreindre le périmètre mais le minimum requis est :</p> <ul style="list-style-type: none"> <li>▪ Transcodification</li> <li>▪ Routage</li> <li>▪ Référencement des schémas XML</li> </ul>

## 2.6 La gestion des paramètres (MDM)

### 2.6.1 Objectifs & démarche

Une architecture SOA manipule une grande quantité de données. Nous avons déjà évoqué le catalogue de services mais le framework lui-même sera composé de plusieurs centaines voire milliers de paramètres. A cela s'ajoutent les référentiels métiers (catalogue produits, tables de scoring, ...) qui généralement bénéficient chacun de leurs propres écrans de gestion.

Le MDM permet d'obtenir une **vue centralisée et unifiée de l'ensemble des paramètres** – techniques et fonctionnels – du Système d'Informations. L'unification des données permet de s'assurer qu'une modification sera correctement propagée dans les systèmes s'appuyant dessus ce qui masque la complexité de la réplication. La vue centralisée permet de n'utiliser qu'un seul outil pour l'ensemble de sa gestion.

Enfin, un outil de MDM définit des profils associés à des données ce qui permet de restreindre certaines modifications à une catégorie de personnes (notion de rôle). La force de ce genre d'outil est de pouvoir être utilisée par les fonctionnels pour l'administration des référentiels, le développement pour le paramétrage applicatif et à l'exploitation pour le paramétrage technique.

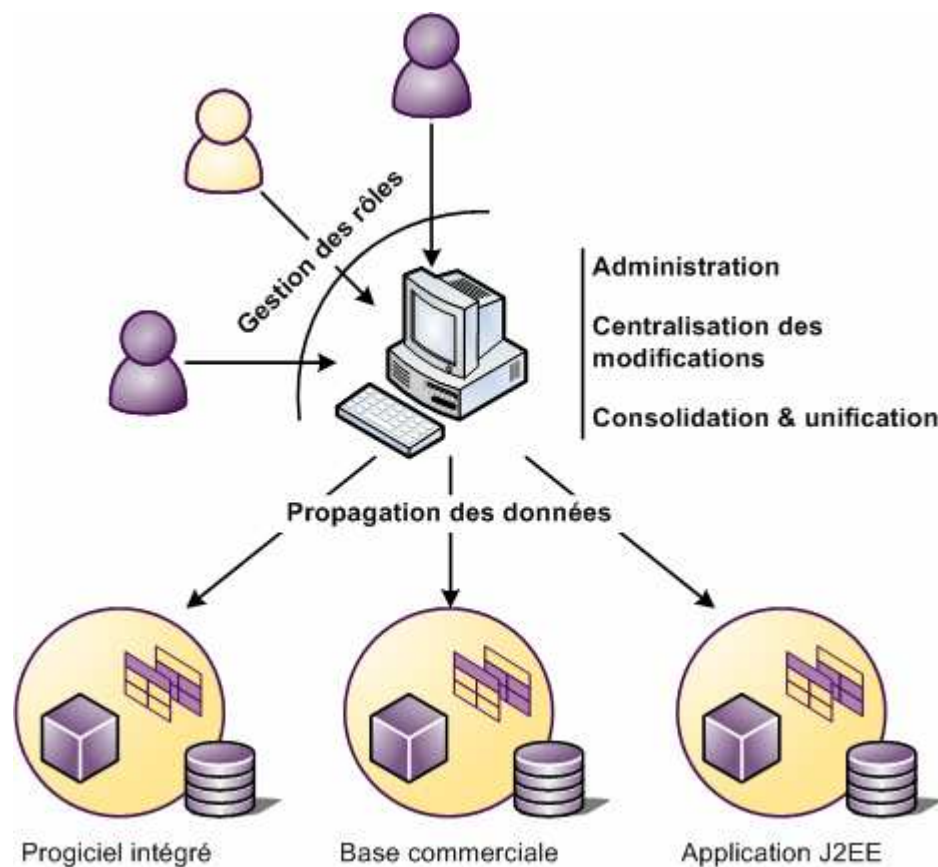


Figure 4 - Principes d'un MDM

## 2.6.2 Critères de vigilance

<b>Processus de travail</b>	La mise en place d'un outil de MDM ne se substitue pas à une réflexion sur le processus global de gestion des données (qui fait quoi et quand).
<b>Connecteurs techniques</b>	Les outils de MDM se focalisent généralement sur la gestion des données. Il est à la charge de l'entreprise de développer les composants permettant de gérer les imports / exports entre le modèle unifié du MDM et les différents systèmes existants.

## 2.7 La nécessité des formats pivots

---

### 2.7.1 Objectifs & démarche

Lorsque l'architecture SOA s'étend sur le S.I., le danger est de voir se multiplier les formats d'échanges.

Leur maîtrise peut s'avérer difficile, notamment en cas de changement de structure de données réutilisées dans plusieurs flux. De même lorsqu'un nouveau consommateur apparaît cela engendre de nouveaux développements au niveau de la couche d'intermédiation ce qui, à la longue, est coûteux en développement et en maintenance.

Une bonne solution consiste à déterminer des formats pivots, généralement par grands domaines fonctionnels, idéalement au niveau de l'entreprise. Ces formats pivots sont des **représentations communes, métiers, partagées et admises par tous**. Cela implique qu'une donnée aura toujours la même signification quelque soit le point de vue fonctionnel.

Un écueil courant dans la détermination des formats pivots est de prendre le modèle déjà en place dans un back-end. A moins que ce modèle n'ait bénéficié d'une étude fonctionnelle avancée et aboutie, on s'aperçoit qu'il est généralement un modèle logiciel c'est-à-dire non partagé et non partageable par définition.

Les formats pivots vont de pair avec une administration de données car la correspondance entre les données dans chaque système et les formats pivots doit bien entendu être clairement identifiée et gérée.

Enfin, ces formats pivots – mais cela est vrai quelque soit le format, pivot ou non – doivent être versionnés et ainsi que leur cycle de vie, géré.

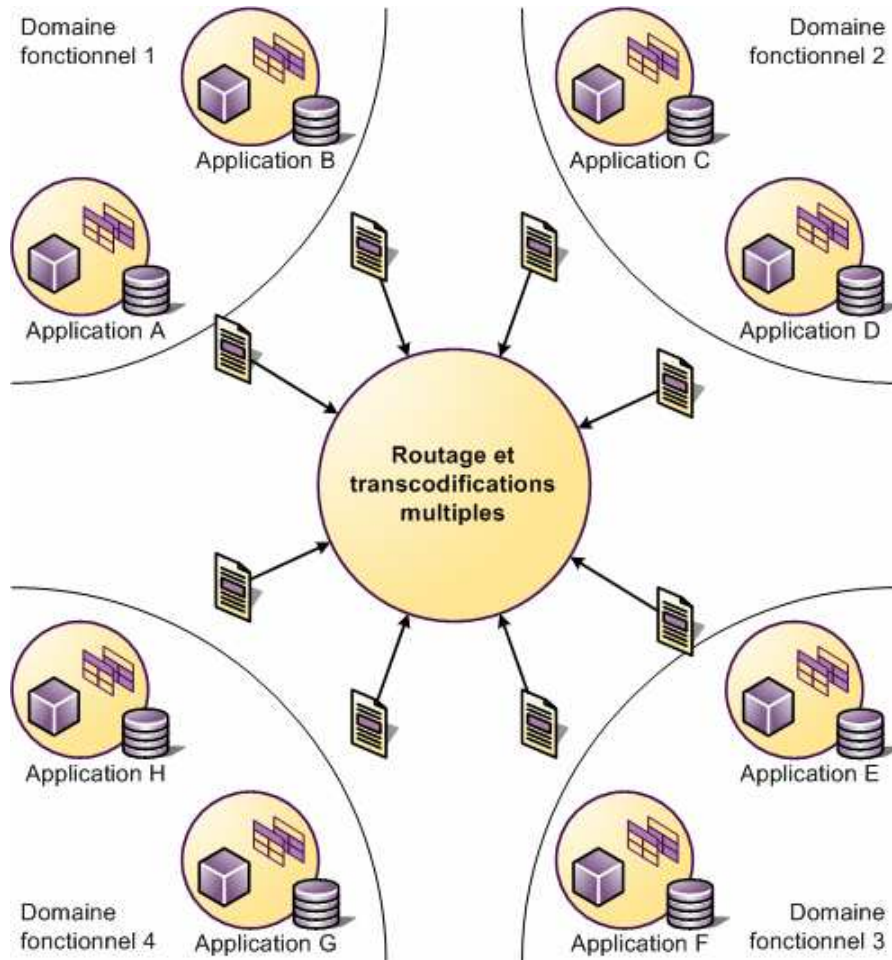


Figure 5 - Sans format pivot, multiplication des formats

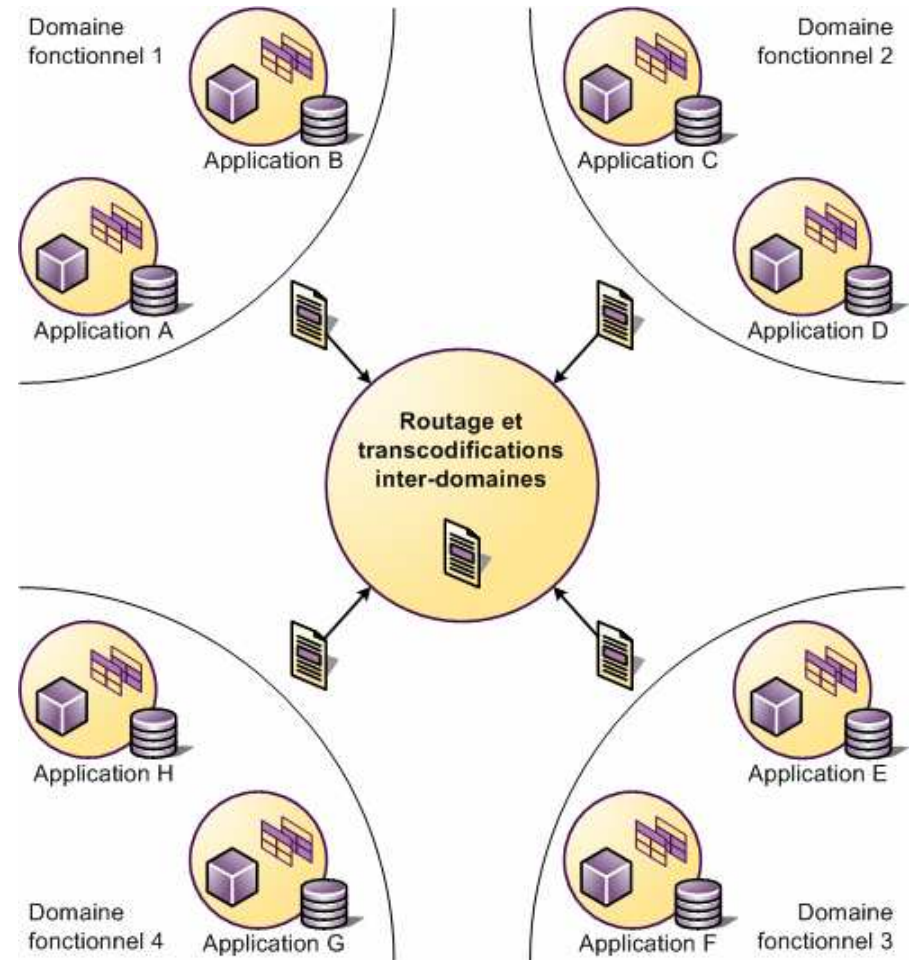


Figure 6 - Formats pivots par domaine

## 2.7.2 Critères de vigilance

<p><b>Etape des négociations</b></p>	<p>La négociation des formats pivots entre différents acteurs métier de l'entreprise est un travail de longue haleine, difficile mais indispensable sur le long terme. Le ROI est par contre très rapide puisque les nouveaux services s'appuieront sur ce format pivot.</p>
<p><b>Décision de ne pas faire de formats pivots</b></p>	<p>Devant l'ampleur du chantier il est tentant de décider de ne pas définir de formats pivots. Les risques encourus sont la multiplicité de connexions EAI pour assurer l'interopérabilité ce qui a pour conséquence immédiate de multiplier les développements et la maintenance.</p>

## 2.8 La contextualisation des services

---

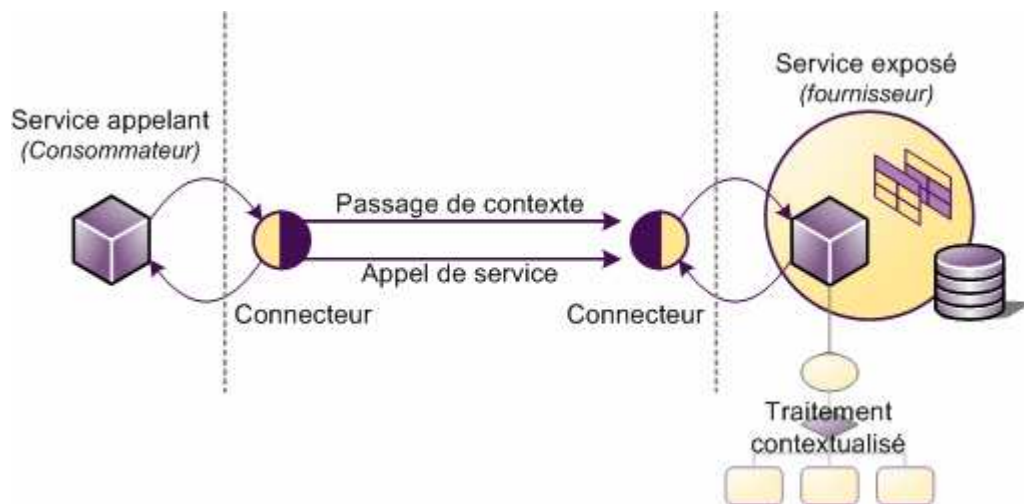
### 2.8.1 Objectifs & démarche

La contextualisation est un point clairement défini sous l'appellation « Execution Context » par l'Oasis, organisme de référence en matière de SOA :

*“The **execution context** of a service interaction is the set of infrastructure elements, process entities, policy assertions and agreements that are identified as part of an instantiated service interaction, and thus forms a path between those with needs and those with capabilities. [...]The execution context is not limited to one side of the interaction; rather it concerns the totality of the interaction – including the service provider, the service consumer and the common infrastructure needed to mediate the interaction. While there may be third parties, for example, government regulators, who set some of the conditions for the execution context, this merely increases the conditions and constraints needing to be coordinated and may require additional information exchange to complete the execution context.”*

Chaque service doit être contextualisé c'est-à-dire que **son comportement varie en fonction du contexte dans lequel il s'exécute.**

Le contexte est considéré comme flottant et toujours présent au cours de l'échange entre le consommateur (du service) et le fournisseur (du service). C'est donc au socle technique (typiquement le framework évoqué au paragraphe 2.4, « Socle de développement : industrialisation et solidification », page 12) de prendre à sa charge ce mécanisme de passage du contexte entre les différentes strates de l'architecture logicielle. Mais ce contexte doit aussi être vu d'un point de vue architecture logique et défini, de même que pour les formats pivots, avec une vue d'ensemble du S.I.



**Figure 7 - Contextualisation des services**

La contextualisation n'est pas exclusivement dédiée aux échanges inter-applications mais aussi lors d'invocations intra-applications et cela tout particulièrement pour la gestion de version du service appelé.

### 2.8.2 Critères de vigilance

<p><b>Transport du contexte</b></p>	<p>Le passage du contexte entre chaque couche d'une application et entre deux applications doit être bien étudié et outillé.</p>
-------------------------------------	--

## 2.9 La supervision et la gestion de la sécurité

---

### 2.9.1 Objectifs & démarche

Lors du déploiement d'une infrastructure SOA se pose nécessairement la problématique de la sécurité d'une part et de la supervision d'autre part.

Traisons tout d'abord l'aspect sécurité ; celui-ci peut être découpé en trois parties :

- Sécurité d'**accès pour un profil** (utilisateur) à un service ;
- Sécurité d'**accès entre les services** (quel service peut accéder à quels autres) ;
- Sécurité des **données** (chiffrement partiel ou total).

Le premier point peut être géré en sécurisant l'accès au niveau front office (authentification « classique », SSO, ...) ou en gérant un niveau d'habilitation avec la notion de carte d'identité XML permettant de valider l'accès à des services pour un profil donné (normes XACML et SAML).

Les accès entre services peuvent être adressés de plusieurs manières : par des contrôles auprès du catalogue de services mais aussi en utilisant les fonctions natives fournies par l'implémentation technique (par exemple la sécurité au niveau des EJB).

Le chiffrement des données, s'il est partiel, est traité soit de manière spécifique, soit en s'appuyant sur les standards du marché au niveau des WebServices avec le standard WS-Security. Quant au chiffrement complet, l'utilisation de SSL, avec https, au niveau du M.O.M. ou des EJB est préconisée mais avec un risque de dégradation des performances.

Concernant la supervision, la notion de **BAM (Business Activity Monitoring)** intervient alors et consiste à :

- Agréger les informations (logs, alertes, messages snmp) de suivis des différentes applications ;
- Consolider et présenter ses données sous une vue unifiée permettant de reproduire l'ensemble d'un acte de gestion ou d'un processus sur le S.I. ;
- Remonter des alertes.

Pour ce faire il est important de définir des formats de logs standards, en s'appuyant par exemple sur CBE (Common Base Event) et donc de mettre en place des **identifiants de corrélation métier** permettant de suivre un processus de bout en bout. L'identifiant de corrélation doit évidemment être propagé par les services y compris lors d'appels en cascade et doit apparaître dans les traces applicatives et les remontées d'alertes.

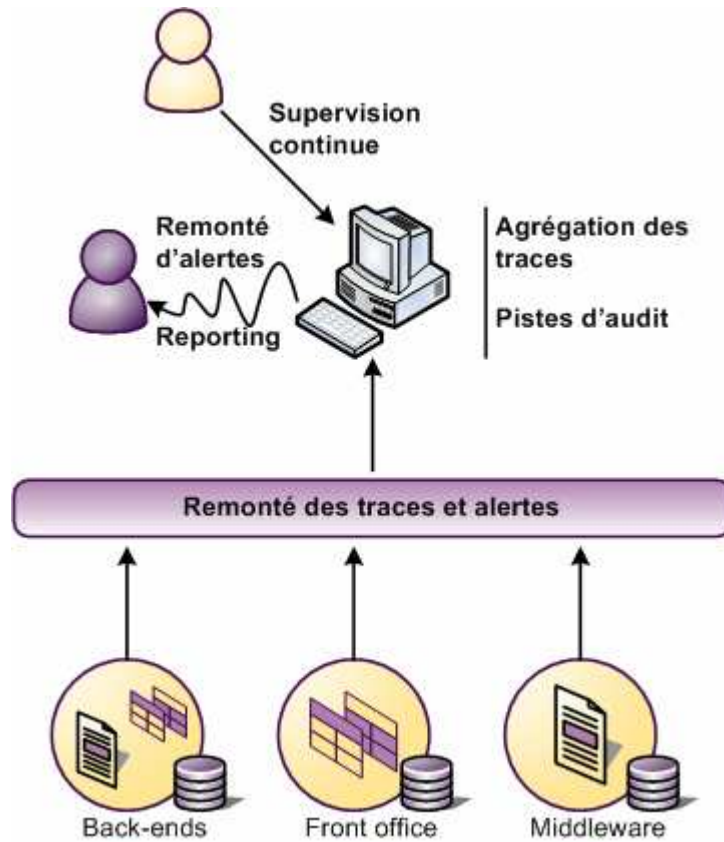


Figure 8 - BAM

Le BAM ne se substitue cependant ni à une supervision technique du socle comme par exemple l'utilisation du réseau ou les espaces disques ni à une supervision des performances (temps de réponse des services).

### 2.9.2 Critères de vigilance

<p><b>Habilitation ou sécurité</b></p>	<p>Il est important de classier correctement les problématiques et de ne pas les mélanger :</p> <ul style="list-style-type: none"> <li>■ Les problématiques d'accès ;</li> <li>■ La notion de pouvoir (ex : contrôle fonctionnel sur des montants) ;</li> <li>■ La segmentation (qui peut voir quoi) ;</li> <li>■ La confidentialité des données ;</li> <li>■ La notion de preuve (piste d'audit).</li> </ul>
--	---

## 2.10 Batch et SOA

### 2.10.1 Objectifs & démarche

La notion de batch nécessite un chapitre à part entière car se pose inévitablement, lorsque l'on parle de réutilisabilité en SOA, la question des batchs.

En SOA, les services sont vus comme unitaires alors que les batchs sont bien souvent considérés sous un angle ensembliste (traitements par lots).

La règle en la matière est simple : il faut repenser l'approche et **considérer les batchs comme des services unitaires** autant que possible.

L'idéal est d'adopter autant que possible une approche « batch fil de l'eau » ce qui revient à une démarche événementielle dans laquelle les informations sont poussées dès que possible. La différence avec un appel de service « classique » est que l'événement se fait en mode asynchrone pur (l'appelant n'attend donc pas de réponse). Dans ce cas précis, un M.O.M. prend toute sa valeur puisque si le service distant n'est pas disponible, l'appelant n'est pas bloqué et l'information sera acheminée dès que faire se peut.

Dans les cas où une approche ensembliste est requise, il est dangereux de vouloir réutiliser les services conçus dans le cadre de l'approche SOA car la tentation sera grande de faire des compromis et donc soit de dénaturer le service soit de pénaliser les performances globales du batch. La conséquence de la non réutilisation de service est bien sûr la duplication inévitable de règles métiers. Cette duplication n'est pas gênante outre-mesure si elle est correctement documentée et acceptée de tout le monde.

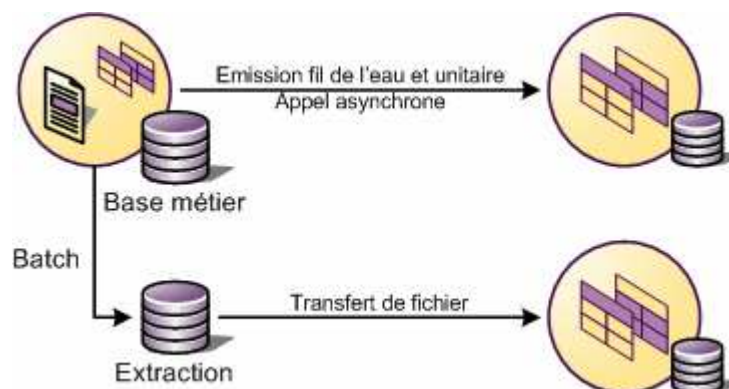


Figure 9 - Batch et fil de l'eau

### 2.10.2 Critères de vigilance

#### Fil de l'eau ou ensembliste

La décision entre les deux modes appartient aux équipes fonctionnelles et peut être prise au cas par cas.

## 2.11 Positionnement de certaines briques concourant à SOA

---

### 2.11.1 Moteur de règles

L'utilisation d'un moteur de règles apparaît très vite comme une évidence dans la mise en place de SOA.

En effet, une architecture SOA donne de l'agilité sur l'interconnexion et l'enchaînement des services entre eux ; **Le moteur de règles donne de l'agilité au sein du service lui-même**. Ce moteur de règles peut être implémenté :

- Soit sous forme d'un service auquel cas une même règle peut être partagée entre plusieurs applications y compris de technologie différentes ;
- Soit en mode natif dans le service ce qui n'est possible que si le moteur de règles partage la technologie d'implémentation de l'application.

Il n'y a pas de réponse précise quant aux critères de choix entre les deux modes il faut trouver le bon compromis entre réutilisation et performance.

De même, il est tentant de tout externaliser dans le moteur de règles, néanmoins si on considère les typologies suivantes :

- Règles métiers ;
- Règles organisationnelles ;
- Pré & post conditions sur les services ;
- Règles d'orchestration (enchaînement).

Seules les trois premières se prêtent à cet usage sachant que seules les règles complexes sont intéressantes à externaliser. Les règles métiers simples, comme des totaux, peuvent demeurer en langage natif.

### 2.11.2 Workflow

Le workflow est vu ici au sens « workflow humain » c'est-à-dire une brique logicielle mettant en œuvre des processus longs incluant à la fois des actions informatisées et des actions non informatisées nécessitant l'intervention d'un humain.

Le workflow dans une architecture SOA peut être vu évidemment comme un consommateur de services mais aussi, et cela est souvent moins anticipé, comme un fournisseur. En effet, un objet métier peut très bien être sous le contrôle d'un workflow pour les démarches administratives liées à la réglementation (délai de prise en compte d'une réclamation) mais aussi sous le contrôle d'un applicatif métier pour les actes de gestion courts. L'applicatif métier interagira donc avec le workflow pour :

- D'une part l'alerter d'actions métier sur l'objet métier ;
- D'autre part vérifier l'adéquation entre l'acte de gestion et l'état de l'objet dans le cas d'actions non-humaines (notion d'habilitation inter-processus).

### 2.11.3 EAI / ESB

L'EAI est un ensemble de technologies que l'on découpe en trois parties :

- Les connecteurs ;
- La couche de transport (M.O.M.) ;
- Le courtier.

Les connecteurs sont les interfaces techniques qui permettent de faire la passerelle entre le M.O.M. et la technologie cible à laquelle on souhaite accéder (une transaction IMS, un programme Java).

Le M.O.M. s'occupe du bon acheminement des messages.

Le courtier à plusieurs rôles, tout d'abord celui de router les messages vers la bonne cible mais peut aussi être un orchestrateur de petit processus.

L'ESB remplit les mêmes fonctions qu'un EAI sauf qu'il s'appuie exclusivement sur les standards du marché (WebServices et J2EE principalement).

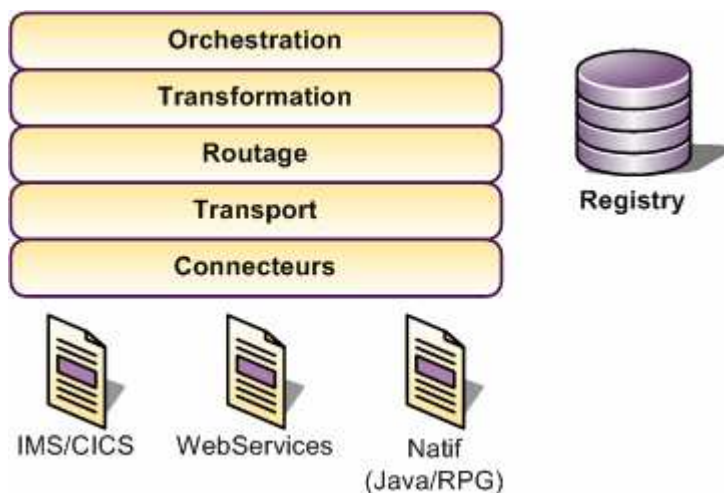


Figure 10 - Fonctions d'un EAI / ESB

## 3 Conclusion

### 3.1 Appréhender SOA

---

SOA est réellement une approche novatrice qui amène à reconsidérer le S.I. dans son ensemble. La conséquence immédiate de cet état de fait est une complexité inhérente à l'approche puisque toutes les composantes du S.I. doivent être prises en considération.

Chaque chantier mentionné dans ce document est un sujet à part entière et nécessite un approfondissement.

La démarche doit s'inscrire dans un schéma directeur à 5 ou 10 ans.

Les investissements humains et financiers sont importants et tout calcul de ROI s'avère être un exercice périlleux voire impossible. L'implication du plus haut niveau de Management de l'entreprise est donc nécessaire.

## Annexes

### 3.2 Nomenclature

SOA brassant des technologies diverses mais aussi des concepts métiers tout aussi variés, il est important d'établir une terminologie, et de positionner une topologie du Système d'Informations.

La définition de ce vocabulaire permet d'échanger autour de concepts compris et partagés par tous ce qui facilite l'échange et le dialogue.

### 3.3 Glossaire

<b>Consommateur (Service)</b>	Un service consommateur est celui qui initie l'échange avec un fournisseur.
<b>Container EJB</b>	Serveur spécifique, rattaché à un serveur d'applications, dans lequel s'exécute les EJB et gérant leur cycle de vie mais aussi la load-balancing.
<b>DAO</b>	Data Access Object, design pattern UML permettant de masquer la technologie de persistance au service appelant.
<b>EJB</b>	Entity Java Bean, API JEE permettant la réalisation de composants fonctionnant en architecture n-tiers & distribués.  Il existe plusieurs types d'EJB : <ul style="list-style-type: none"> <li>▪ Entity (Entité) : pour la persistance des données, il en existe deux sortes : BMP (Bean Managed Persistence) pour lequel il faut produire le code d'accès aux données. CMP (Container Managed Persistence) pour lequel le container gère l'accès aux données.</li> <li>▪ Session : contenant les services métier, deux sortes existent: Stateless (sans état) : ne garde pas la mémoire de la conversation utilisateur. Statefull (avec état) : garde la mémoire de la conversation utilisateur.</li> <li>▪ Message Driven : l'EJB est activé à réception d'un message via un M.O.M.</li> </ul>
<b>Fournisseur (Service)</b>	Un service fournisseur est un service qui répond à une sollicitation.
<b>M.O.M.</b>	Middleware Orienté Message. Technologie permettant l'échange de messages asynchrone entre systèmes.
<b>Orchestration</b>	L'orchestration est une opération algorithmique d'enchaînement de services.
<b>POJO</b>	Plain Old Java Object, classe Java simple, n'héritant d'aucune classe ou n'implémentant aucune interface issues d'un quelconque framework Java.



<b>Post conditions</b>	Partie algorithmique déterministe d'un service, répondant par vrai ou faux en aval du traitement du service. En cas de réponse affirmative, le consommateur à la garantie que le service pourra répondre correctement.
<b>Pré conditions</b>	Partie algorithmique déterministe d'un service, répondant par vrai ou faux en amont du traitement du service. En cas de réponse affirmative, le consommateur à la garantie que le service s'exécutera correctement.
<b>Propagation</b>	Mode d'enchaînement de services dans lequel chacun appelle le suivant.
<b>Service</b>	Un service expose un algorithme spécifié de manière formelle au travers d'une signature de service et potentiellement des pré- et post-conditions.
<b>XML Schema (XSD)</b>	Standard XML pour spécifier les structures de données (types, éléments et attributs).