MANAGEMENT SYSTÈMES D'INFORMATION

# SOA

## Le guide de l'architecte du SI

Xavier Fournier-Morel, Pascal Grojean
Guillaume Plouin, Cyril Rognon

Préface de Luc Fayard

2e édition

DUNOD

# SOA

## Le guide de l'architecte

# SOA

## Le guide de l'architecte du SI

**Xavier Fournier-Morel**
*Responsable du pôle architecture de SQLI Suisse*

**Pascal Grojean**
*Responsable de SQLI Consulting*

**Guillaume Plouin**
*Responsable de la veille technologique du groupe SQLI*

**Cyril Rognon**
*Responsable "capitalisation et R&D" de SQLI Consulting*

2e édition

DUNOD sqli

Toutes les marques citées dans cet ouvrage sont des marques déposées par leurs propriétaires respectifs.

Source : digitalvision®





ISBN 978-2-10-053549-1

# Préface

Éloge de l'ouverture.

« Règle numéro un : le client a toujours raison. Règle numéro deux : si jamais le client a tort, relire la règle numéro un. » Le fameux slogan des magasins américains Stew Leonard pourrait s'appliquer au système d'information, en prenant le mot « client » dans toutes ses acceptions. Car ce qui a vraiment changé ces dernières années en informatique, c'est peut-être cela : un regard différent sur son environnement, la compréhension qu'un beau projet est forcément un travail en commun, non seulement des hommes mais aussi des ressources et des outils.

C'est sans doute cette force de l'ouverture qui pousse à la « webisation » des applications d'entreprise d'une part et aux SOA (*Services Oriented Architecture*) d'autre part. Cette « architecture orientée services » fut d'abord lancée comme un concept à la mode et elle aurait pu terminer comme nombre d'entre eux, dans l'arrière-salle des fausses bonnes idées. Mais elle fut rapidement relayée par des vraies méthodes et des outils. Au point de devenir peut-être le levier le plus important aujourd'hui dans la modernisation de l'informatique.

Un autre des grands avantages des SOA et des services web est de renforcer le lien entre fournisseurs et utilisateurs : difficile en effet de relier des composants métiers sans se pencher sur ces métiers ! On a vu alors cette chose impensable quelques années auparavant, la constitution de groupes de projets multidisciplinaires ou les spécialistes de la technologie et ceux de l'entreprise commençaient enfin à se parler d'égal à égal !

Cette idée que l'innovation réside plus dans l'usage que dans la technologie elle-même est probablement la vraie révolution. Elle change le regard des créateurs de produits high-tech qui ne les conçoivent plus en fonction de spécifications purement techniques mais comme une réponse aux besoins du marché. Elle implique les utilisateurs beaucoup plus tôt dans le processus de la conception. En intégrant la notion de SOA, l'informatique ne fait que se mettre au diapason du business moderne où le client est roi, le marketing « *one to one* » et le travail collaboratif.

Mais nous n'en sommes qu'au début de cette nouvelle ère, le jargon est encore là, les explications ne sont pas toujours des plus claires. Cet ouvrage est donc le bienvenu. D'autant que les SOA ne sont qu'une solution nouvelle à un vieux problème : ce n'est pas d'hier en effet que datent les réflexions sur les composants et la séparation des applications en couches. Ni SOA ni web services ne sont la panacée : l'architecture homogène et fluide c'est bien, les données standardisées c'est mieux. La multiplicité des normes et des standards, la décomposition trop accentuée peut-être des différentes briques, le difficile passage du concept à la réalité, autant d'obstacles à franchir. Pour obtenir cette informatique moderne, fluide, agile dont tout le monde rêve, il reste encore beaucoup de travail à faire. Mais les SOA sont sans doute une étape incontournable.

Luc Fayard

Directeur de la rédaction de 01 Informatique

# Avant-propos

Les problématiques de communication et d'intégration entre les applications hétérogènes au sein d'une entreprise sont un vieux défi de l'informatique. On a tenté par le passé de les résoudre par diverses méthodes et technologies : scripts pilotés par batch, échanges de fichiers par FTP, middleware orientés message, EAI, etc. Avec Corba, on a essayé d'apporter une réponse standardisée à la problématique, mais sans rencontrer de réel succès décisif.

Or l'empilement des applications au fil du temps a conduit à une situation intenable de « silos étanches ». L'absence de solution architecturale efficace pour résoudre ce problème a plongé les systèmes d'information dans une situation de blocage vis-à-vis des exigences des métiers.

Les architectures orientées services (SOA) offrent aujourd'hui un nouveau modèle et une formidable opportunité pour résoudre ces problématiques. Elles vont même **beaucoup plus loin** puisqu'elles permettent de mettre le système d'information au service des métiers, en procédant à son alignement avec les processus d'entreprise.

Cet ouvrage a pour objectif de présenter et de recadrer la définition et les usages de SOA. Il s'agit d'exposer les concepts de service dans l'architecture du système d'information, sans tomber dans le piège de la description de l'implémentation technique. En effet, trop de personnes confondent aujourd'hui SOA et Web Services.

Les auteurs souhaitent se démarquer des divers livres blancs disponibles sur le sujet en faisant la part entre l'effet de mode, le marketing des éditeurs et l'apport véritable de ces types d'architecture par rapport à ce qu'ils appellent le « cahier des charges des SI agiles ».

Ils proposent de lister les questions à se poser pour déployer SOA à bon escient, en mettant l'accent sur une méthodologie pertinente qui recouvre les aspects architecturaux et organisationnels.

Un modèle d'implémentation technique est alors proposé au travers des Web Services et d'autres approches basées sur JEE.

Un éclairage est donné sur la vision des architectures de services proposée par les acteurs du Web 2.0, et sur le caractère complémentaire plutôt que contradictoire de cette vision avec l'approche SOA.

Enfin, les auteurs présentent l'ensemble des outils nécessaires à la mise en œuvre et la maintenance d'une architecture SOA, et font le point sur les réponses des éditeurs de logiciels à ces besoins.

### *Première partie – Le cahier des charges des SI agiles*

L'objectif de cette première partie est d'introduire les problématiques de rationalisation des systèmes d'information, puis de présenter un cahier des charges pour disposer d'un SI en phase avec les attentes actuelles.

Elle présente tout d'abord un historique de l'évolution des systèmes d'information et montre comment ces derniers sont souvent complexes et peu cohérents. Elle évoque ensuite le besoin d'« agilité » des SI dans le contexte actuel d'accélération de l'économie mondiale. Elle explique, de plus, pourquoi les outils dont disposaient jusqu'à présent les DSI répondaient mal à leur problématique de bonne cohérence.

Enfin, elle aborde les exigences métiers et techniques que l'on peut attendre d'une nouvelle démarche d'organisation et d'intégration du SI. La suite de l'ouvrage montrera comment SOA répond à ces exigences.

> Cette partie introductive est accessible à tous les profils : maîtrises d'œuvre comme maîtrises d'ouvrage.

### *Seconde partie – Présentation de l'approche SOA*

L'objectif de cette deuxième partie est de clarifier les concepts utilisés, en commençant par la notion de services et d'architecture de services, puis d'introduire la composition de services. Elle présente une architecture de référence positionnant les différents concepts. Elle clarifie également pourquoi les Web Services sont utiles, mais pas indispensables.

Elle s'attache à décrire les fondements du concept de service et notamment la notion de contrat et de messages.

Enfin, elle introduit les outils à mettre en place de façon progressive en regard de l'architecture SOA de référence.

> Cette partie est conceptuelle. Elle est accessible à tous les profils de maîtrises d'œuvre, mais aussi aux maîtrises d'ouvrage qui désirent appréhender en profondeur les architectures fonctionnelles réalisables avec SOA.

### *Troisième partie – SOA : Tout repose sur la méthode*

La partie 3 présente les principaux aspects méthodologiques de l'approche SOA, avec deux points de vue complémentaires : la modélisation de l'architecture et le cycle de vie d'un projet SOA.

Cette partie traite d'abord de la modélisation des services. Est abordée ensuite la modélisation des processus métier, l'adoption de SOA par les entreprises étant en grande partie liée à la volonté de mettre en place de nouveaux processus métier.

Enfin, le chapitre sur la modélisation des applications interactives propose d'étudier l'impact de SOA sur le classique modèle MVC.

Le chapitre sur le cycle de vie d'un projet met en évidence les impacts de SOA sur les différentes étapes d'un cycle de vie logiciel. L'outillage de ce cycle de vie conduit à aborder les aspects productivité et à examiner la relation entre SOA et MDA.

SOA n'est pas une mode mais une évolution profonde de la façon de penser le système d'information : les équipes doivent donc mûrir, et la partie 3 se clôt sur la présentation d'un modèle de maturité architecturale, complémentaire d'une démarche d'industrialisation de développements de Type CMM-I.

Cette partie méthodologique vise les architectes et chefs de projets.

### *Quatrième partie – La boite à outils Web Services*

On a précisé dans cet avant propos qu'il ne faut pas faire d'amalgame entre SOA et Web Services. Cependant, la conception des spécifications Web Services a été menée dans l'objectif de répondre au mieux aux enjeux des architectures SOA. Ainsi, les Web Services sont une technologie très pertinente pour mener une démarche SOA.

L'objectif de cette partie est de présenter les différentes spécifications qui pourront être mises en œuvre dans le cadre d'une architecture SOA basée sur les Web Services : spécifications de base des Web Services (SOAP, WSDL, UDDI), spécifications permettant de gérer la sécurité, la garantie d'acheminement, la garantie transactionnelle, le monitoring des services, les aspects composition, orchestration et présentation.

Cette partie présente des normes techniques : elle intéressera les architectes, chefs de projets et développeurs.

### *Cinquième partie – SOA : une mise en œuvre concrète*

Une mise en œuvre réussie d'une SOA et de services doit faire correspondre les enjeux métiers, les contraintes et avantages techniques, au sein du SI local ou entre SI. La partie 5 de ce livre s'adresse aux architectes techniques et veut démystifier une utilisation pragmatique et pertinente des technologies pour bénéficier de SOA. Si ce type de solutions est réalisable avec ou sans les Web Services, il est très important de bénéficier des formalismes comme WSDL qui vont structurer la démarche de production de services, en s'appuyant sur des normes comme WSI-Basic Profile.

Cette partie détaille certains aspects techniques permettant d'utiliser plus finement et en connaissance de cause les assistants et ateliers logiciels qui fournissent ou accompagnent la production de Web Services.

Enfin, on y présente des solutions permettant d'accéder à différentes implémentations d'un même service. C'est l'annonce des solutions ESB et SCA, où la conception s'abstrait encore plus des contraintes et des évolutions techniques.

Cette partie est très technique : elle s'adresse aux chefs de projets techniques et développeurs qui mettront en place SOA sur le terrain.

### *Sixième partie. SOA et Web 2.0*

La « blogosphère » oppose souvent les approches SOA et Web 2.0. Une thèse régulièrement avancée suggère que SOA serait une démarche lourde et complexe tandis que le Web 2.0 proposerait la simplicité et l'agilité.

Cette partie montre comment les deux approches ne s'opposent pas, mais sont au contraire complémentaires. Elle explique en détail les concepts et principes d'architecture qui sous-tendent le Web 2.0 : Mashups, Ajax et REST. Enfin, elle analyse cette approche Web 2.0 dans la perspective de construire une SOA d'entreprise, en mettant en avant ses avantages et ses inconvénients – en évitant un engouement aveugle comme un dénigrement systématique.

Cette partie traite d'alternatives en termes d'architecture technique. Elle intéressera les architectes, chefs de projets techniques et développeurs.

### *Septième partie – Décrypter l'offre du marché*

Cette dernière partie dresse un panorama des outils disponibles pour bâtir, héberger, exécuter et administrer les solutions métier SOA.

Elle précise le périmètre de l'ESB (*Enterprise Service Bus*), en liste les principales fonctions techniques. Elle présente les outils complémentaires de l'ESB, comme le registre des services, l'orchestrateur, le distributeur de requêtes CRUD ou le cache de données. Elle introduit de ce fait, le concept de plate-forme SOA, encore appelée APS, *Application Platform Suite*.

Elle se termine par une présentation des éditeurs qui proposent un outillage SOA. Si cette présentation n'est pas exhaustive, elle donne les critères permettant d'aboutir à un choix d'outils raisonné.

Cette partie intéressera les architectes et équipes d'exploitation, responsables de choisir les bons outils pour construire et maintenir une SOA.

### *Remerciements*

Les auteurs tiennent tout d'abord à remercier leurs épouses pour leur patience et leur soutien pendant les longs mois de la rédaction de ce livre.

Leur reconnaissance va aussi à Bruno LEYSSENE et Yahya EL MIR, directeurs du groupe SQLI, qui ont sponsorisé et rendu possible ce projet. Le soutien d'Emmanuel BOUCHET, directeur de SQLI Suisse, a aussi été d'un grand secours.

Enfin, ils remercient des collègues et amis qui ont bien voulu relire l'ouvrage et les faire bénéficier de leurs remarques pertinentes : Jean-Christophe BROQUAIRE, Zakaria EL MOUJAHID, Jacques KHA, Xavier LINAIS, Pascal CADET, Gabriel KASTENBAUM et Hugues MARGUET.

# Table des matières

## Troisième partie – SOA : tout repose sur la méthode

## Quatrième partie – La boîte à outils Web Services

## Cinquième partie – SOA : Une mise en œuvre concrète

## Sixième partie – SOA et Web 2.0

## Septième partie – Décrypter l'offre du marché

# PREMIÈRE PARTIE

# Le cahier des charges des SI agiles

L'objectif de cette première partie est d'introduire les problématiques de rationalisation et d'agilité des Systèmes d'Information, puis de présenter le cahier des charges d'un SI en phase avec les attentes actuelles. Ce dernier est envisagé sous un angle métier, puis sous un angle technique.

- Le premier chapitre présente l'évolution de l'informatique et les problématiques d'entropie des SI ; il montre que l'accélération globale du monde actuel risque d'accroître fortement ces problèmes si rien n'est fait.
- Le deuxième chapitre montre que, jusqu'à présent, l'outillage (architectures, méthodes, techniques) mis à disposition des DSI résout mal ou partiellement les problématiques de rationalisation du SI.
- Le troisième chapitre présente les exigences métier et techniques que l'on peut attendre d'une nouvelle démarche d'organisation et d'intégration du SI.

Les parties suivantes montreront comment SOA répond à ces différentes exigences.

# 1

# De l'entropie des Systèmes d'Information

**Objectif**

Ce chapitre rappelle brièvement l'évolution des technologies informatiques et ses conséquences en terme d'entropie pour le Système d'Information. Il fait ainsi le constat de la difficulté à maintenir un SI homogène et rationnel.

Il présente ensuite les évolutions des marchés mondialisés, avec comme conséquence des exigences croissantes des métiers vis-à-vis du SI.

## 1.1 UNE BRÈVE HISTOIRE DE L'INFORMATIQUE

### 1.1.1 Le mainframe

Dans l'histoire de l'informatique, on a tout d'abord conçu des applications monolithiques, les sites centraux, où toute la logique de persistance, de traitement, et de présentation était agrégée dans un ensemble indissociable. Toute l'informatique des entreprises était alors concentrée dans un serveur unique : le mainframe.

Le mainframe supporte en général des applications de gestion essentielles pour l'entreprise. Sa principale contrainte technique est d'assurer la haute disponibilité et l'intégrité des données. Ses coûts d'acquisition et d'exploitation sont élevés, mais il offre à l'entreprise un système totalement cohérent et fiable.

### 1.1.2 Les applications client/serveur

Avec l'invention du PC, sont apparues des applications plus légères, en architecture client/serveur. Ces architectures ont proposé de déporter les composants d'interface sur les postes de travail en conservant des serveurs pour la persistance. Elles ont ainsi apporté plus de graphisme et plus de sophistication en terme d'ergonomie.

Le faible coût de ces nouvelles applications a permis leur multiplication dans les entreprises. Ces dernières se sont alors dotées de logiciels transverses de messagerie, de gestion financière, d'aide à la décision, etc. De plus, l'informatique a commencé à être utilisée par les métiers et progressivement par tous les profils de l'entreprise.

Ainsi, les Systèmes d'Information ont grandi rapidement, à la suite d'initiatives éparses des différents départements et métiers de l'entreprise. L'absence de gouvernance de cette croissance a souvent abouti à des duplications d'informations, saisies et gérées de manière autonome par ces différents services.

Par ailleurs, l'enthousiasme autour des applications client/serveur a engorgé les postes de travail de multiples exécutables difficiles à maintenir et dotés d'interfaces incohérentes. Cette multiplication des interfaces s'est révélée complexe à maîtriser et donc improductive pour l'utilisateur.

### 1.1.3 Les applications Web

À la fin des années 1990, les problématiques de déploiement et de gestion des parcs de PC ont montré les limites des applications client/serveur. De plus, un nouveau besoin s'est fait sentir : celui de proposer l'accès aux applications non plus seulement aux employés de l'entreprise, mais aussi à ses clients et partenaires.

Dès lors sont apparues les applications Web, auxquelles on accédait à l'aide d'un simple navigateur. Elles ont permis d'intégrer les clients et partenaires aux processus métiers de l'entreprise et de développer le commerce électronique. De plus, elles ont amené une réduction des coûts de gestion et de licence du parc informatique.

La montée en puissance de la programmation orientée objet a accompagné le développement des applications Web avec tous les bénéfices qu'on connaît : usage de composants et frameworks, réutilisabilité, modélisation UML, etc. Cependant, les architectures n-Tiers, associées à ces technologies, ont augmenté le morcellement du SI.

### 1.1.4 Un premier bilan

Chacune des technologies citées dans les paragraphes précédents a eu sa pertinence à un moment donné dans l'histoire de l'informatique. De plus, chacune a ses avantages et ses inconvénients.

Par conséquent, aucune entreprise n'a fait le choix de la table rase à chaque changement de génération technologique. La conservation de ce biotope hétérogène s'explique aussi par les contraintes organisationnelles et financières qui résulteraient d'un tel changement.

Ainsi, les directions informatiques ont dû faire face à des parcs informatiques très morcelés et surtout à des applications peu à même de communiquer ou de partager des informations.

Comme évoqué précédemment, l'informatique s'immisce dans tous les métiers et le PC devient omniprésent sur les bureaux de tous collaborateurs de l'entreprise. On parle pour cela d'informatique « pervasive » : la multiplication des applications devrait donc s'accentuer.

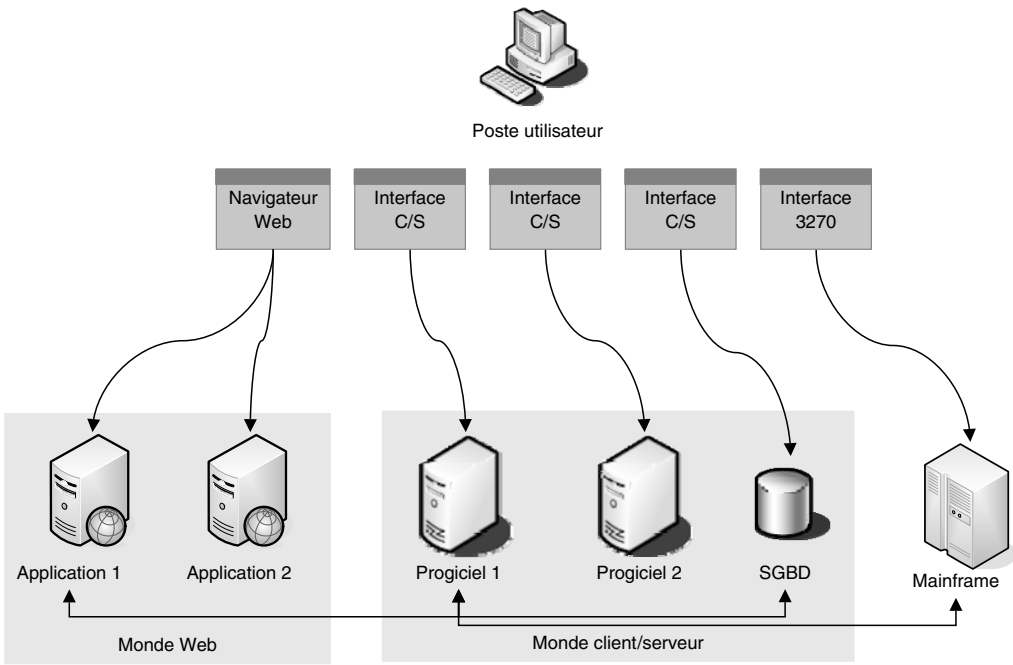



**Figure 1.1** — L'entropie du SI

Le constat est donc le suivant : les DSI doivent trouver un moyen d'organiser et de maîtriser cette hétérogénéité. Ils doivent aussi rationaliser la mise en commun d'informations transverses à l'entreprise : les fameux référentiels.

## 1.2 L'ACCÉLÉRATION DES MARCHÉS ET SES CONSÉQUENCES SUR LE SI

### 1.2.1 Le besoin d'agilité

Chacun sait aujourd'hui que la mondialisation des échanges a rendu la concurrence plus âpre entre les entreprises de tous types. Pour rester compétitives, ces dernières doivent ainsi être capables de livrer de nouveaux produits de plus en plus vite, en réduisant au maximum les coûts de production.

Le système d'information est impacté par cette accélération; dans le cadre de processus de plus en plus informatisés, il est au cœur de ces nouvelles exigences de productivité. Les métiers lui demandent d'être extrêmement réactif, pour aider à mettre sur le marché de nouvelles offres. On parle pour cela **d'agilité** : ce terme désigne la capacité du SI à supporter des évolutions rapides.

Un corollaire de l'accélération des marchés est l'importance des fusions/acquisitions qui permettent aux entreprises d'asseoir leur position et d'élargir leur offre. Cette tendance a aussi des conséquences sur le SI. Il doit en effet être capable d'absorber rapidement d'autres systèmes informatiques, souvent hétérogènes sur le plan technologique. On voit là un autre aspect de **l'agilité**, réclamée cette fois par la direction de l'entreprise.

Enfin, l'entreprise peut être amenée à développer une offre sur la base d'une nouvelle technologie. Par exemple, un opérateur télécom va proposer une nouvelle offre haut débit en s'appuyant sur la technologie WiMax. Dans ce cadre encore, on exigera du SI qu'il s'adapte rapidement aux impacts de cette nouvelle technologie pour supporter la nouvelle offre. C'est encore un exemple **d'agilité**.

En résumé, on dit généralement que les besoins métiers et stratégiques de l'entreprise dessinent une cible à atteindre pour le SI. Pour bien servir son entreprise, on dit que ce dernier doit **s'aligner** avec les besoins exprimés et suivre une **trajectoire** cohérente avec la cible.

### 1.2.2 L'intégration des acteurs externes

Les produits de consommation sont de plus en plus construits sur mesure selon les désidératas des clients. Ainsi, on voit apparaître dans diverses industries des « configurateurs » qui permettent au client de façonner son produit : par exemple, les configurateurs automobiles permettent aux internautes de choisir, à partir d'un vaste catalogue, la cylindrée, la couleur, les accessoires de leur automobile. Les constructeurs de matériel informatique offrent le même genre de souplesse.

On a vu précédemment que les applications Web avaient permis la naissance du commerce électronique en offrant aux consommateurs une interface vers le SI, afin de mener des actes d'achat. Avec les configurateurs, on va plus loin puisque le consommateur intervient dans la phase de conception de son article. Il devient donc architecte de son produit.

L'entreprise s'appuie donc sur le SI pour produire en flux tendu, offrir des produits personnalisés, livrer plus vite, et cela sans augmenter ses effectifs.

Par conséquent le SI fait face à de nouvelles exigences :

- Il doit offrir de plus en plus de portes d'entrées aux consommateurs et sous-traitants.
- Il doit supporter des processus de plus en plus automatisés et faisant intervenir une grande variété d'acteurs.
- Il doit offrir une disponibilité à toute épreuve.

On verra dans la suite que la démarche SOA propose des solutions pour répondre à cette exigence d'intégration des acteurs externes dans les processus métiers.

## En résumé

On a vu que l'évolution de l'informatique a conduit à des SI hétérogènes et difficiles à maintenir.

Par ailleurs, la généralisation de l'outil informatique dans les directions métiers fait que les utilisateurs attendent toujours plus du SI.

Il devient donc vital de trouver des moyens de mieux gérer et homogénéiser le SI.

# 2

# Les limites des réponses usuelles

**Objectif**

Le chapitre précédent a montré les problématiques d'entropie du SI.

Nous allons maintenant présenter les solutions proposées jusqu'à ce jour pour tenter sa rationalisation, et en brosser les principales limites.

## 2.1 LA DÉMARCHE D'URBANISATION[1]

### 2.1.1 Introduction à la métaphore de la cité

La première réponse historique à la rationalisation du SI est la démarche dite d'urbanisation. Cette démarche propose de comparer le SI à une ville comprenant des zones, des quartiers et des îlots[2], puis d'appliquer une méthodologie proche à celle de l'urbanisation traditionnelle pour gérer son organisation.

L'urbanisation traditionnelle porte sur la rationalisation des réseaux de circulation, d'égouts, d'électricité, de gaz, etc.; tandis que l'urbanisation des SI travaille

1. Pour en savoir plus sur la démarche d'urbanisation : *Le projet d'urbanisation du système d'information*, Christophe Longépé, Dunod, 2001.
2. Dans la cité physique, un îlot est un pâté de maison, tandis qu'en informatique, c'est une application obtenue par développement spécifique ou achat de progiciel.

essentiellement sur les processus métiers, la communication inter-applicative et sur la mise en place de référentiels transverses.

Le travail sur les référentiels est une démarche organisationnelle de longue haleine. Il consiste à lister les informations dupliquées au sein de l'entreprise, puis à essayer de les rapprocher sur la base de clefs d'identification uniques : numéro de commande, numéro de produit, identifiant client, etc. Les outils techniques ne jouent pas un rôle important dans cette démarche. Cependant, elle peut s'appuyer sur des outils de dédoublonnage des données, comme les MDM (cf. partie 7).

Par contre des outils sont proposés dans la démarche d'urbanisation dans le cadre des échanges inter-applicatifs, de l'automatisation des processus métiers, de l'homogénéisation des interfaces : ils sont présentés dans la suite de ce chapitre.

### 2.1.2 Le modèle de référence de l'urbanisme

De même que leurs homologues travaillant sur la cité physique, les urbanistes de SI utilisent des cartes et des plans à différents niveaux de lecture. Dans la cité physique, on parle de plan d'occupation des sols, de plan du réseau des transports en commun, de plan des réseaux d'égouts, etc. De même, les cartographies du SI sont multiples et abordent différents niveaux de technicité.

Les urbanistes de SI utilisent généralement une convention de représentation avec 4 niveaux de cartographies que nous appellerons dans la suite **le modèle de référence de l'urbanisme**. Ces quatre niveaux, présentés dans la figure 2.1, sont les suivants :

- **La vue métier :** elle recense les événements métier que l'entreprise doit traiter, les processus métier répondants à ces événements, les documents utilisés dans les processus, et les acteurs exécutant les processus en consultant/élaborant les documents. L'expression « Processus Métier » désigne ici les procédures en vigueur dans l'entreprise pour traiter un événement métier : à ce niveau, on ne se soucie pas encore d'automatiser ces processus via un outil.
- **La vue fonctionnelle :** elle décrit les fonctions du Système d'Information, telles qu'on les décrit dans un cahier des charges en vue de la mise en œuvre d'une nouvelle application. Les fonctions peuvent être par exemple : la gestion d'un contrat client, la gestion d'une expédition de produit, la messagerie électronique, les agendas partagés, etc. C'est au niveau de la vue fonctionnelle que les urbanistes parlent de zones et de quartiers, afin de regrouper les fonctions connexes.
- **La vue applicative :** elle recense l'ensemble des applications informatiques utilisées par les acteurs pour remplir des fonctions et outiller des processus.
- **La vue physique :** elle recense l'ensemble des composants d'infrastructure (matériels, infrastructure réseau, outillage de sécurité, archivage, outillage d'administration et de monitoring, etc.) qui supportent le SI.

Les vues processus et fonctionnelles représentent les besoins des métiers : elles sont abstraites. A contrario, les vues applicatives et physiques représentent des implémentations bien réelles.

Toute la difficulté de la démarche d'urbanisation réside dans l'intersection entre une vision abstraite idéale et une réalité concrète que doivent intégrer les problématiques terre à terre de maintenance, mise à jour, déploiement, intégration avec l'existant, pannes, etc.

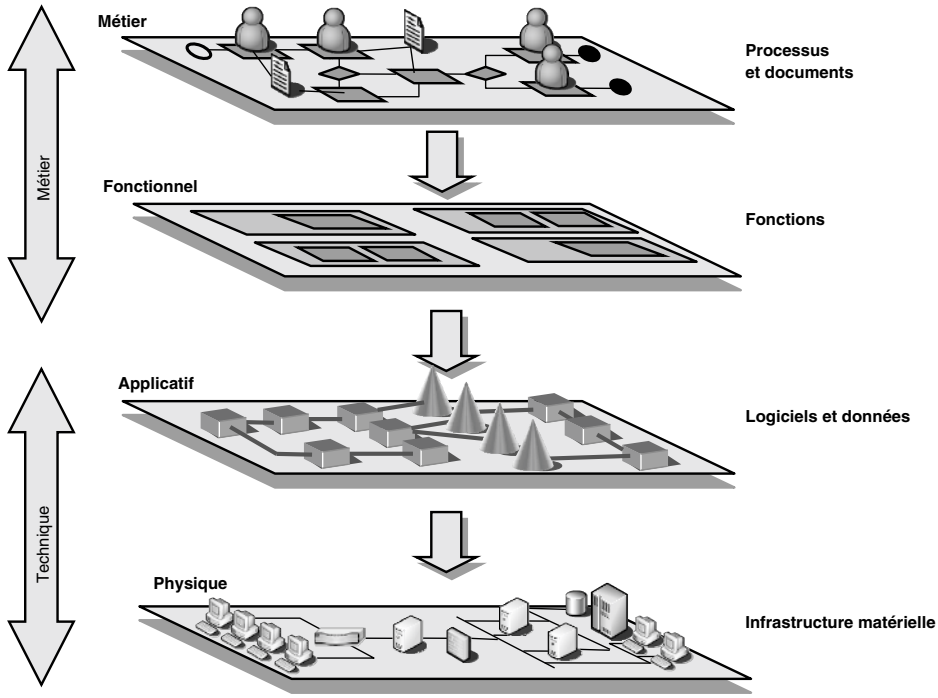



**Figure 2.1** — Le modèle de référence de l'urbanisme

En particulier, le risque qu'encourent les urbanistes est de produire des quantités importantes de documents abstraits qui ne mènent dans la pratique à aucune action concrète sur le terrain. Ce travers est comparable à celui des architectes qui conçoivent des tours d'un kilomètre de haut et ne les construisent jamais.

De fait, dans des entreprises de taille importante, la démarche d'urbanisation a parfois pour seul objectif de connaître l'existant, de disposer d'une vue d'ensemble du SI.

> On verra dans la suite que SOA offre une solution innovante pour gérer l'interface entre les besoins des métiers et l'implémentation technique. La démarche SOA facilite ainsi le travail des urbanistes.

### 2.1.3 La réalité du SI

Comme on l'a vu dans le chapitre précédent, le SI doit faire face à une grande hétérogénéité technologique. De plus les applications sont généralement mal équipées pour communiquer avec les autres blocs du SI. On parle pour cela de fonctionnement en « **silo** ».

Chaque silo est généralement autonome et isolé en terme de processus, d'interface homme/machine, de socle technique. Ainsi la vision par couche des urbanistes se heurte aujourd'hui à une réalité technique qui entrave leur démarche globale de rationalisation (cf. figure 2.2).

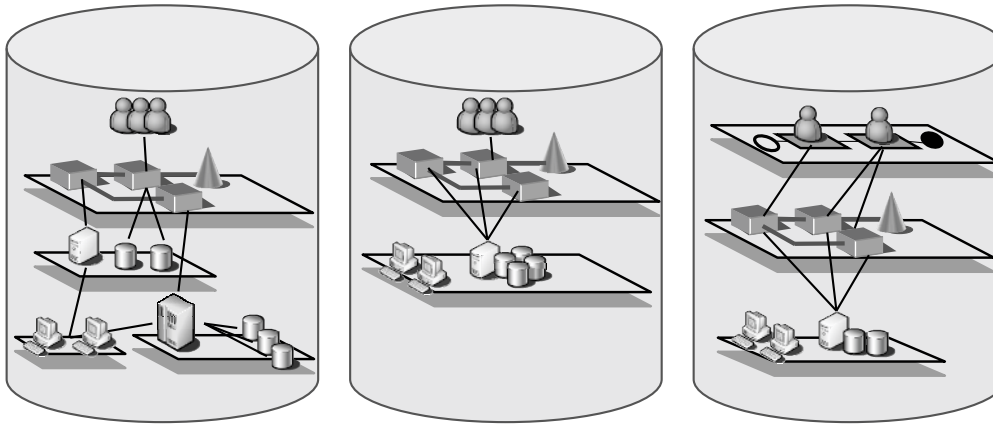


**Figure 2.2** — La problématique des silos applicatifs

Il en résulte de nombreux désagréments, tels que :

- L'impossibilité de mettre en place des processus métier réellement transverses, car ces processus devraient accéder à des fonctions enfouies dans des technologies diverses.
- La duplication des objets et des règles métier dans les référentiels, donc l'obligation des doubles saisies, la difficulté d'avoir une vision « client » unifiée, etc.
- La difficulté d'évolution des applications due à la complexité engendrée par la variété des technologies qui résulte de l'histoire du SI.
- La difficulté à profiter des opportunités technologiques.
- L'impossibilité de réutiliser des composants logiciels.

Dans ce contexte les communications inter-applicatives sont souvent gérées au cas par cas, par des liens un à un en fonction des besoins : c'est le fameux **syndrome du « plat de spaghettis »**.

# 2.2 UN OUTILLAGE AU SERVICE DES URBANISTES

Ce paragraphe décrit un certain nombre d'outils, conçus dans un objectif d'urbanisation : il présente leur mode de fonctionnement et leurs limites.

## 2.2.1 Les outils EAI

Les outils EAI (*Enterprise Application Integration*) sont des solutions dédiées à la gestion des échanges d'informations entre applications hétérogènes sur le plan technique. Leur objectif premier est donc de rationaliser le « plat de spaghetti » évoqué plus haut.

Ils répondent généralement aux besoins suivants :

- Offrir une interface (un connecteur) vers toutes les typologies d'applications du SI.
- Acheminer les messages/données entre applications.
- Savoir mener des transformations intermédiaires des messages.
- Orchestrer les processus d'intégration.

La figure 2.3 présente schématiquement le fonctionnement d'un outil d'EAI.

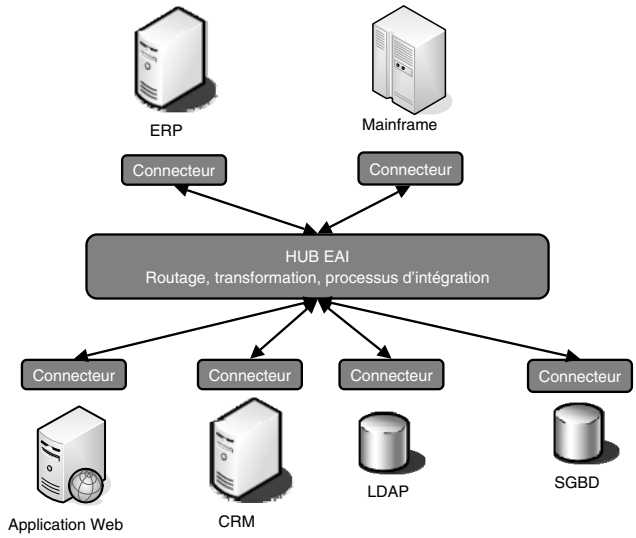



**Figure 2.3** — Représentation schématique d'un outil EAI

Les outils EAI pâtissent malheureusement d'un certain nombre de défauts :

- L'acquisition et le déploiement d'un outil EAI sont **très onéreux**. De plus, ces outils nécessitent un **paramétrage long et fastidieux** avant les premiers échanges de messages entre applications.

- Les solutions EAI sont **propriétaires** : chacune utilise ses mécanismes propres pour offrir les services présentés précédemment. Ainsi, le SI devient très dépendant de l'éditeur de la solution.
- Elles ne disposent pas toujours de l'exhaustivité des connecteurs nécessaires à l'entreprise.
- Tout repose sur l'outil EAI : il est ainsi le passage obligé, le **bouc émissaire** et le « *Single Point of Failure* » de tous les processus d'intégration.

Enfin les outils EAI ne résolvent pas la problématique du fonctionnement en silo : ils proposent un principe de rectification après coup, un système palliatif (cf. figure 2.4).

On verra dans la suite que, a contrario, la démarche SOA résout les problématiques de fonctionnement en silo de manière proactive.

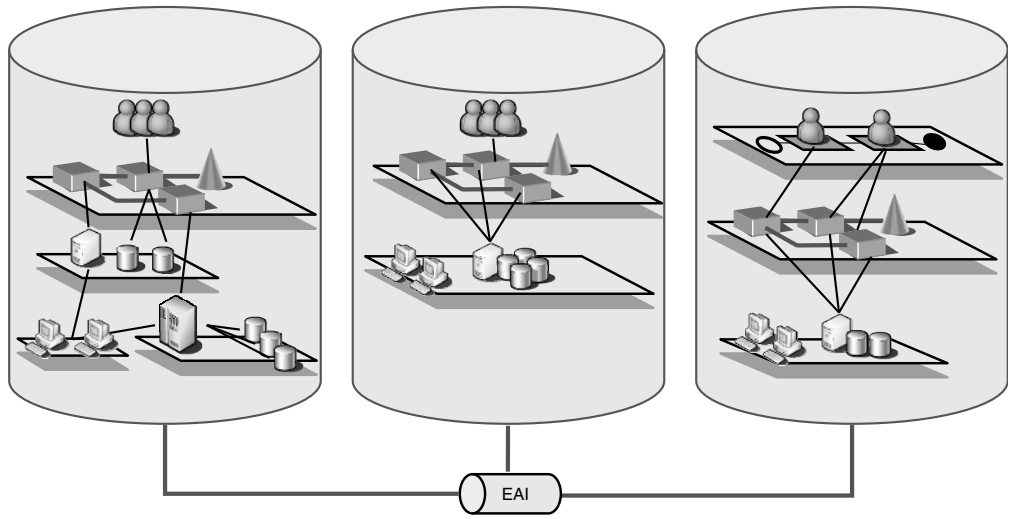



**Figure 2.4** — L'EAI favorise les silos

### 2.2.2 Les outils de workflow

Les outils de workflow ont été conçus pour permettre d'automatiser rapidement un processus métier à partir de sa modélisation. Ils proposent donc une interface de modélisation, utilisable par des non-informaticiens et un moteur permettant l'exécution du processus modélisé.

Ces outils ne couvrent pas tous les types de processus : il s'agit généralement de processus dit *human-driven*, car ils sont basés sur des séries de tâches effectuées par des utilisateurs finaux : demandes, validations, rejets, etc. (cf. partie 3).

Tout comme les outils d'EAI, les solutions de workflow souffrent de limitations :

- Elles sont **propriétaires** : chacune utilise ses mécanismes propres pour modéliser et déployer les processus métiers. Ainsi, le SI devient très dépendant de l'éditeur de la solution.
- Elles sont **limitées** en terme d'intégration avec des processus supportés par d'autres applications.
- Elles sont **limitées** à des typologies de processus bien spécifiques : elles ne sont donc pas généralisables à l'ensemble de l'entreprise.
- Elles nécessitent une **adaptation** des applications existantes si celles-ci doivent s'intégrer au processus métier.

Enfin les outils de workflow sont généralement utilisés au niveau d'un département de l'entreprise, donc d'un silo : ils ne prennent pas en compte les processus globaux et donc encouragent le fonctionnement en silo (cf. figure 2.5).

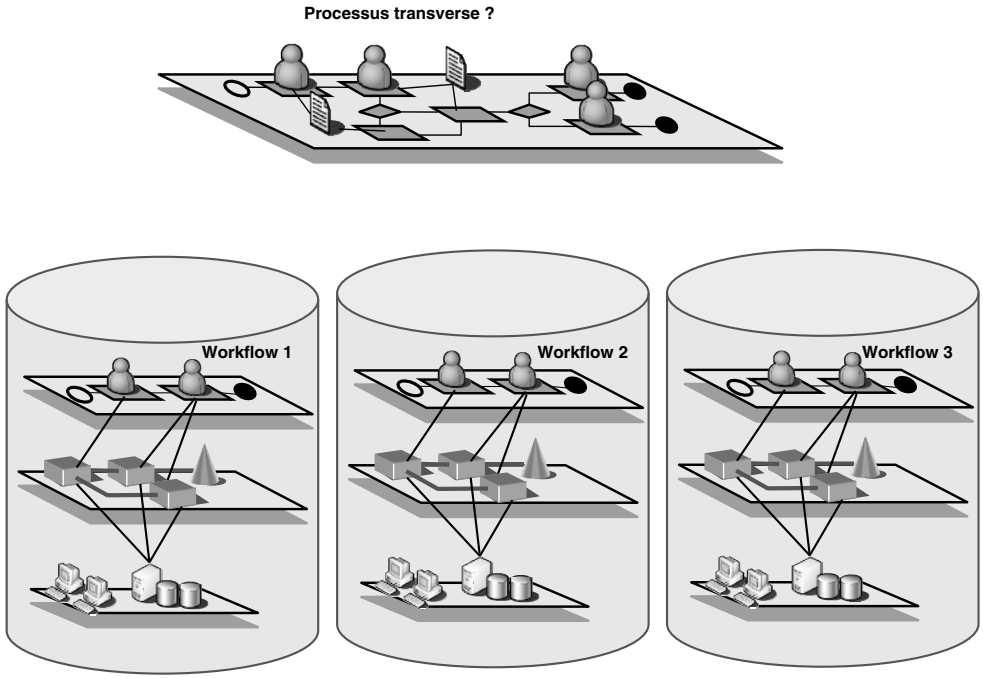



**Figure 2.5** — L'outil de Workflow encourage les silos

## 2.2.3 Les portails Web

Les outils de portail ont été inventés pour offrir un point d'entrée unique, sous la forme d'une interface unifiée et cohérente aux utilisateurs du SI. Leur objectif était donc de mettre fin à la prolifération de multiples exécutables sur le poste utilisateur,

et d'agréger les interfaces Web si celles-ci sont multiples (cf. les grandes entreprises qui comptent jusqu'à 50 intranets).

Les portails de première génération proposaient pour cela des batteries de connecteurs capables d'intégrer des interfaces d'application client/serveur, mainframe ou autre. Ces connecteurs permettaient de transformer des interfaces technologiquement hétérogènes en interfaces Web à l'aide d'applets Java, d'Active X ou d'opération de « *revamping* »[1] (cf. figure 2.6).

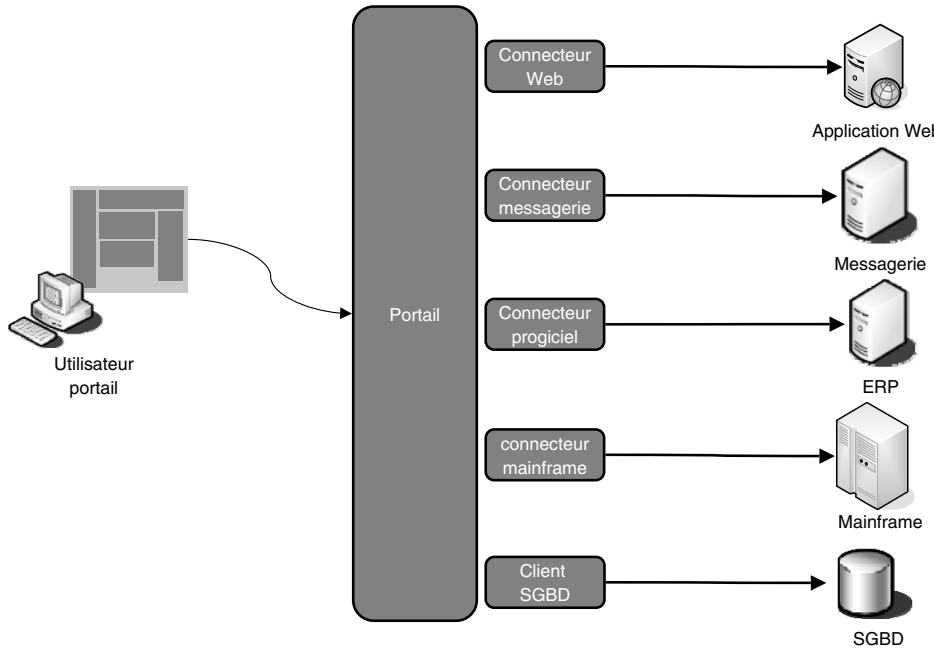



**Figure 2.6** — Fonctionnement des portails de première génération

Ces outils portails de première génération souffraient comme les outils précédents de défauts :

- Ils nécessitaient un **paramétrage long et fastidieux**.
- Ils étaient **propriétaires** : chacune utilise ses mécanismes propres pour agréger les interfaces. Ainsi, le SI devient très dépendant de l'éditeur de la solution.
- Ils ne disposaient pas toujours de l'exhaustivité des connecteurs nécessaires à l'entreprise.
- Ils proposaient une interface non personnalisable et ne prenaient pas en compte la diversité des accédants potentiels : collaborateurs, clients, fournisseurs, mais aussi processus B2B automatisés.

1. On désigne par ce terme les systèmes de transformation d'interface lourde en interface Web.

Enfin les portails de première génération ne résolvaient pas la problématique du fonctionnement en silo : au contraire, ils l'encourageaient en proposant de contourner le problème par un système palliatif (cf. figure 2.7).

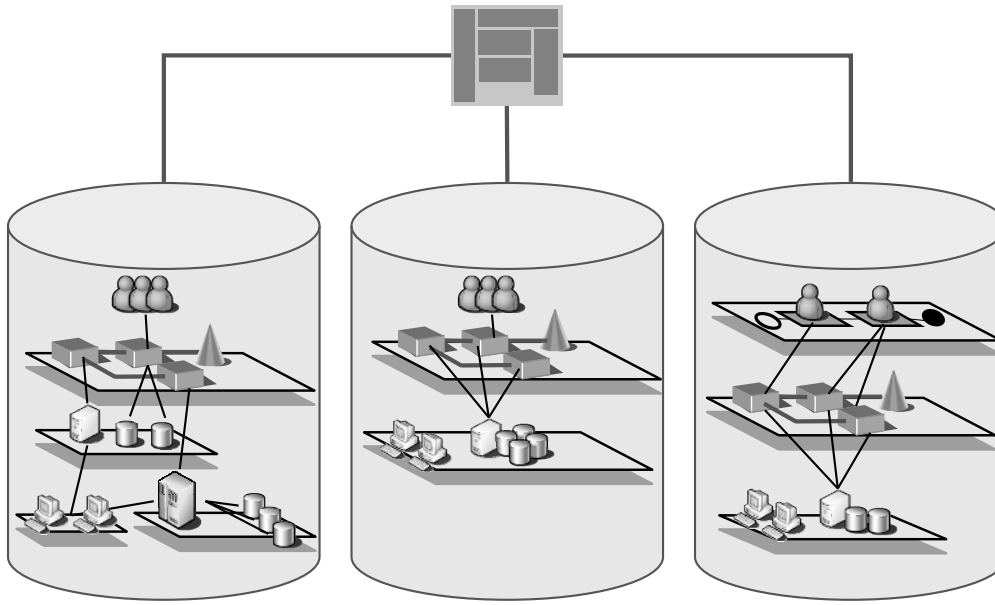


**Figure 2.7** — Le portail de première génération encourage les silos

On verra dans la suite que les portails de dernière génération utilisent des protocoles standard pour accéder aux applications (cf. partie 4), et qu'ils leur imposent une structuration plus évolutive, conforme aux vœux des urbanistes.

### 2.2.4 Les langages objets

Les langages objets ont offert des potentiels intéressants de réutilisation. En effet, le concept d'interface permet de créer des composants invocables par d'autres.

Il est ainsi possible de composer des éléments pour construire des applications de plus en plus complexes, ou bien de s'appuyer sur des briques existantes pour personnaliser leurs fonctions. Ce procédé est au cœur des concepts de frameworks.

Cependant, si les composants objets sont réutilisables au sein d'un ensemble applicatif monolithique, ils le sont moins à l'échelle du SI. En effet, ils offrent une trop forte adhérence à leurs technologies d'implémentation (Java, .NET, etc.) pour être utilisable à grande échelle.

Ainsi les capacités de réutilisation des langages objets ne sont pas exploitables dans une démarche d'urbanisation. Et l'objet, de même que les outils évoqués précédemment, favorise les silos applicatifs.

### 2.2.5 Un constat de frustration

Les paragraphes précédents ont montré que la démarche d'urbanisation est louable et même indispensable pour tout SI de taille conséquente. Cependant, les solutions dont on disposait jusqu'à présent ne donnent pas satisfaction. De fait, la démarche aboutissait souvent à une documentation « littéraire » de la cible du SI urbanisée, qui ne débouchait sur aucune mise en œuvre concrète.

Les outils traditionnels de type portail, EAI et Workflow disposent d'une souplesse limitée et fonctionnent selon un mode propriétaire. Ils sont un pis-aller, une solution incomplète qui n'apporte pas entière satisfaction, car ils n'offrent pas de vraie alternative au fonctionnement par silo, qui bride l'évolutivité du SI.

> On verra dans la suite de cet ouvrage que la démarche SOA offre une solution beaucoup plus complète à ces problématiques d'urbanisation : elle se base sur des principes indépendants des technologies ou des solutions d'éditeurs. Elle répond ainsi aux problématiques de connectivité et d'interopérabilité. Elle offre une vision plus large et plus satisfaisante pour les urbanistes.

## En résumé

Les outils d'EAI, portail et de Workflow ont proposé des solutions parcellaires pour mener à bien l'urbanisation du SI.

Il apparaît clairement que, dans le cadre de cette démarche, l'indépendance vis-à-vis d'une technologie ou d'un éditeur donné est essentielle. On verra dans la suite comment SOA permet cette indépendance.

# 3

# Le cahier des charges du SI

**Objectif**

Le chapitre précédent a montré que la démarche d'urbanisation telle qu'elle a été menée jusqu'à présent n'offrait pas entière satisfaction pour rationaliser le SI.

L'objectif de ce chapitre est maintenant de définir :

- Quelles fonctions devraient offrir un SI pour mieux répondre aux besoins des métiers de l'entreprise.
- Quelles fonctions devraient offrir une plate-forme d'intégration pour satisfaire les urbanistes.

## 3.1 LES EXIGENCES DES MÉTIERS

On a vu dans ce qui précède que l'informatique devient omniprésente et donc de plus en plus critique dans l'entreprise, et qu'elle doit s'aligner avec les besoins métiers et stratégiques. L'évolution des modes d'usage des SI fait apparaître d'autres exigences, décrites dans ce paragraphe.

### 3.1.1 Déployer rapidement des processus métiers

Comme évoqué dans le paragraphe sur les outils de Workflow, l'entreprise a aujourd'hui besoin de déployer rapidement des processus métiers à partir d'une expression de besoins.

Le Système d'Information doit donc proposer des outils permettant de modéliser puis de déployer les processus dans un temps court et en limitant les développements spécifiques. Ces outils doivent être plus souples et plus généralistes que les outils de Workflow. On verra dans la suite que **l'approche SOA propose des solutions pour répondre au besoin de déployer rapidement des processus métiers**.

### 3.1.2 Les métiers ont besoin d'une vision temps réel sur le business

Pour faire face à l'accélération des marchés évoquée précédemment, les métiers doivent aujourd'hui disposer d'indicateurs en temps réel sur l'activité de l'entreprise :

- Ils doivent pouvoir accéder aux informations consolidées sur un client donné : coordonnées, contrats, commandes en cours, fiches de satisfactions, etc.
- Ils doivent de même disposer d'indicateurs sur le cycle de vie d'un produit donné : commandes en cours, facturation, stocks, capacité d'approvisionnement, etc.
- Ils doivent enfin disposer d'indicateurs sur le comportement et l'efficacité des processus métier automatisés.

Pour satisfaire ce type d'exigence, le Système d'Information doit être capable de donner une visibilité sur les processus et les référentiels d'entreprise, et ceci quel que soit le point d'entrée. Il doit pouvoir décloisonner les applications monolithiques et faciliter la circulation d'informations en son sein. Il doit aussi être agnostique sur le plan technologique, c'est-à-dire qu'il ne doit pas rencontrer de point de blocage lié à une hétérogénéité technologique.On verra dans la suite que **la démarche SOA propose des solutions pour répondre à ce besoin d'indicateurs en temps réel**.

### 3.1.3 La DSI doit justifier son budget

Le temps où l'informatique était perçue par les directions et les métiers comme une mécanique mystérieuse et impénétrable est révolu. Les DSI ne peuvent plus, comme dans le passé, disposer de budgets pour investir dans une nouvelle technologie sans justification, ni calcul de retour sur investissement.

L'informatique est aujourd'hui une source d'énergie pour l'entreprise au même titre que l'électricité. Ses dépenses doivent donc être justifiées et rationalisées; et les DSI doivent travailler sur la pérennisation de leurs investissements. De plus, les nouveaux investissements doivent être pensés en terme de réutilisabilité.

Ainsi, il n'est plus d'actualité de déclencher des « big bangs », c'est-à-dire de redévelopper entièrement un existant dans le seul objectif d'utiliser la technologie la plus actuelle. On privilégie plutôt l'intégration des anciens systèmes avec les nouveaux grâce à des middlewares adaptés.

Par ailleurs, on favorise le développement de composants ou frameworks réutilisables et capables de s'assembler de diverses manières, selon les besoins des métiers. On verra dans la suite que **la démarche SOA répond à cette exigence d'investissement informatique en phase avec les besoins métiers**.

## 3.2 LES EXIGENCES TECHNIQUES

Toute nouvelle démarche d'harmonisation globale reposera nécessairement sur une infrastructure d'intégration. Cette infrastructure ne sera pas seulement un système d'acheminement de messages : elle devra offrir des fonctionnalités de gestion de processus, de données, d'application et de suivi. De plus elle proposera des outils de mise en œuvre et d'aide au déploiement. Dans la suite, on désignera cette infrastructure par le vocable « **plate-forme d'intégration** ».

### 3.2.1 Inciter à la réutilisation

#### *Valoriser l'existant du SI*

Cette exigence est la traduction technique de la pérennisation des briques existantes du SI (cf. chapitre précédent). La plate-forme d'intégration du SI doit en tout premier lieu intégrer des interfaces vers les diverses typologies d'applications.

Dans les Systèmes d'Informations un peu conséquents, on trouve un certain nombre de familles applicatives :

- Applications métiers, très variables selon le secteur d'activité de l'entreprise.
- Applications de collaboration : progiciels de messagerie, agendas partagés, etc.
- Applications de support : progiciels de gestion des ressources humaines, de la paye, de la comptabilité, etc.
- Applications décisionnelles, permettant la remontée d'indicateurs de pilotage.
- Bases référentielles.

La plate-forme d'intégration devra donc proposer une connectivité exhaustive afin de couvrir toutes les typologies d'applications héritées au sein du Système d'Information.

> On verra dans la suite de cet ouvrage que SOA privilégie la notion de « *best effort* », c'est-à-dire que chaque application devra faire de son mieux pour présenter des interfaces facilement intégrables par les autres. La plate-forme d'intégration doit donc offrir des interfaces vers les applications incapables de fournir ce « *best effort* », notamment les applications patrimoniales.

#### *Composer les nouvelles applications*

On a vu précédemment que le SI doit proposer des applications réutilisables et capables de s'assembler de diverses manières, selon les besoins des métiers.

La plate-forme d'intégration devra donc offrir des mécanismes techniques permettant de composer les applications qu'elles soient interactives, semi-automatisées, ou automatisées. Ces aspects seront détaillés dans la partie 2.

#### *Ouvrir les interfaces*

On a vu dans le chapitre 1 que le SI doit offrir des interfaces pour toutes les typologies d'utilisateurs, en particulier les clients et les fournisseurs.

La plate-forme d'intégration devra donc offrir des mécanismes techniques permettant de mettre à disposition des Interfaces Homme-Machine sur la base de la composition d'applications évoquée dans le paragraphe précédent. Ces interfaces pourront être adaptées par les utilisateurs externes au SI selon leur environnement et leur charte graphique. Elles pourront aussi être accédées par des processus automatiques.

### 3.2.2 Faciliter les échanges à tous les niveaux du SI

#### *Savoir gérer différentes typologies de flux*

La plate-forme d'intégration doit être capable de véhiculer des informations entre les applications selon différents modes d'échange, suivant différents formats, et à différents niveaux d'intégration.

Les modes d'échanges à adresser sont les suivants :

- **Échange synchrone (architecture couplée) :** dans ce cas, l'application appelante attend la réponse à sa requête avant de continuer ses traitements. HTTP est un mode classique d'échange synchrone.
- **Échange asynchrone (architecture découplée) :** dans ce cas, la demande est placée dans une queue d'attente et le traitement en cours n'est pas interrompu. Il prendra en compte la réponse à sa requête lorsque celle-ci sera disponible. Parmi les systèmes d'échange asynchrones les plus classiques, on peut citer les middlewares orientés message, ou tout simplement les protocoles de messagerie.
- **Mode *Publish and Suscribe* :** dans ce contexte, la plate-forme d'intégration met à disposition des abonnements à des flux d'informations. Les applications désireuses d'être informées de changements au sein du SI peuvent s'abonner à ces flux. Un cas classique est une souscription aux informations d'évolution d'un référentiel métier.
- **Information par déclenchement :** dans ce cas, c'est un événement applicatif qui provoque l'échange d'information. La plate-forme d'intégration détecte un changement au sein d'une des applications du SI et notifie les îlots applicatifs concernés. On peut citer l'exemple de l'arrivée d'un nouveau collaborateur dans l'entreprise : son insertion dans le logiciel de gestion des ressources humaines déclenche la création d'une boîte mail.

#### *Assurer la sécurité des échanges inter-applicatifs*

**Rappels généraux sur la sécurité[a]**

Les spécialistes analysent généralement les besoins de sécurité d'une plate-forme suivant cinq fondamentaux :

- **L'authentification** : elle consiste à s'assurer de l'identité d'un utilisateur avant de lui donner l'accès à un système ou à une application. ➢

a. Pour en savoir plus sur les notions de sécurité, voir *Sécurité des Architectures Web*, Guillaume Plouin, Julien Soyer, Marc-Éric Trioullier, Dunod, 2004.

➢
- **La confidentialité :** elle consiste à empêcher la lecture d'informations par des personnes non habilitées ou malintentionnées.
- **L'intégrité :** elle désigne la capacité à s'assurer de la non-altération des informations d'origine, qu'elle soit accidentelle ou malveillante.
- **La disponibilité :** elle concerne la garantie sur le bon fonctionnement d'une application, sa résistance vis-à-vis des pannes accidentelles et des attaques incapacitantes.
- **La traçabilité :** elle consiste à stocker des traces de toutes les interactions des utilisateurs avec les applications afin de pouvoir détecter des attaques ou des dysfonctionnements.

La disponibilité et la traçabilité sont des problématiques d'exploitation : elles relèvent des aspects SLA, évoqués dans la partie 2 de cet ouvrage.

L'authentification peut être gérée de diverses manières : identifiant/mot de passe, générateur de mot de passe à usage unique, biométrie, etc. cependant, la manière la plus élégante d'authentifier un accédant est d'utiliser un **certificat numérique** au format X509. Le certificat est l'équivalent d'une carte d'identité dans le monde numérique : il permet de s'authentifier, mais aussi de **signer numériquement** des documents. Des certificats sont, par exemple, utilisés dans le cadre de la télé-déclaration des impôts sur Internet.

La signature électronique, la confidentialité et l'intégrité sont généralement assurées par des algorithmes cryptographiques. On en distingue trois grandes familles :
- **Les algorithmes à clef secrète** comme triple DES ou AES : ils sont basés sur un secret partagé, la valeur de la clef.
- **Les algorithmes à clef publique** comme RSA ou DSA : ils utilisent des paires des clefs, appelées clef publique et clef privée.
- **Les algorithmes de calcul d'empreinte** comme MD5 ou SHA1 : ils associent à un document une sorte d'identifiant unique qui permet de vérifier son intégrité.

SSL est, par exemple, un cas classique d'usage de ces algorithmes pour assurer la confidentialité des échanges.

La plate-forme d'intégration doit proposer une réponse aux problématiques de l'encart précédent. En particulier :

- Authentifier les accédants au SI, en particulier si ceux-ci sont des personnes externes à l'entreprise (clients, fournisseurs).
- Assurer la confidentialité et l'intégrité des échanges, en particulier si ceux-ci mettent en jeu des tiers externes à l'entreprise.
- Assurer la disponibilité des services (cf. exigences du chapitre 2).

### *Assurer une garantie d'acheminement*

**Rappels sur la garantie d'acheminement**

La plupart des protocoles de transport utilisés en informatique n'offrent pas de garantie de livraison suffisante pour des transactions applicatives. Des protocoles comme HTTP et SMTP offrent bien un contrôle technique sur les erreurs de transmission,

➢

➢
mais ce contrôle est de très bas niveau : il ne permet pas de notifier un groupe de services qu'un message est bien arrivé ou qu'une séquence de messages est arrivée dans le bon ordre.
L'usage d'un système de garantie d'acheminement est essentiel en architecture distribuée, d'autant plus qu'on ne traite pas des échanges point à point mais des échanges entre de nombreux services. Il est particulièrement recommandé dans le cadre d'échanges asynchrones et de gestion de queues de messages, où la perte de contrôle sur l'acheminement des données se révèle assez fréquente sans infrastructure ad-hoc.

La plate-forme d'intégration devra offrir les garanties présentées dans l'encart précédent :

- Notifier qu'un message est bien arrivé.
- Notifier qu'une séquence de messages est arrivée dans le bon ordre.

*Gérer les transactions distribuées*

**Rappel sur les transactions distribuées**

Le cadre des transactions distribuées est celui où l'on doit écrire des données dans un référentiel à partir de traitements concurrents et distribués. Ce type de transaction nécessite un middleware assurant la cohérence des accès en écriture : le moniteur transactionnel.
Les transactions distribuées fonctionnement généralement selon le principe *two-phase commit* :
- **Première phase :** le moniteur transactionnel demande à chaque ressource d'être prête soit à valider toutes les modifications, soit à les révoquer.
- **Seconde phase :** si toutes les préparations sont OK le moniteur demande à toutes les ressources de valider les modifications (*commit*), sinon il demande à toutes les ressources de tout révoquer (*roll back*).

La plate-forme d'intégration devra offrir les garanties présentées dans l'encart précédent.

De plus, lorsque les processus métiers mettent en œuvre de nombreuses étapes intermédiaires avant écriture des transactions dans les référentiels, on parle de « transactions longues » (cf. partie 3). La plate-forme d'intégration devra aussi savoir gérer ces typologies de transactions.

### 3.2.3 Reposer sur des référentiels de méta-données

Pour offrir au SI une vision de ses capacités d'intégration, il est important que la plate-forme dispose d'un référentiel des applications prêtes à l'intégration. Ce dernier doit référencer les modes d'accès aux îlots applicatifs et pointer sur les dictionnaires décrivant leur langage d'échange. Il doit aussi savoir localiser les différentes

versions disponibles des applications et cela en environnement de production, de recette, de développement, etc.

Pour assurer l'authentification et la gestion des droits des accédants, la plate-forme devra, en outre, disposer d'un référentiel des utilisateurs internes et externes au SI.

De plus, il peut être nécessaire de déployer un référentiel consolidé de description des données : il s'agit notamment d'équivalence sémantique entre concepts métier, de tables de références qui pointent vers les référentiels d'entreprise grâce à des clefs d'identification uniques, etc. Ce référentiel permet en particulier de dupliquer des données métiers en conservant une maîtrise sur leur intégrité.

Enfin, au cours de processus mettant en jeu plusieurs applications, il peut être utile que la plate-forme joue un rôle de traducteur. La traduction à effectuer par la plate-forme d'intégration consistera à transposer des messages d'une syntaxe donnée vers une autre. Elle devra pour cela disposer des dictionnaires propres à chaque application, c'est-à-dire d'un référentiel des grammaires utilisées par chacune d'entre elles. Le besoin de traduction requiert donc un référentiel des dictionnaires de traduction.

### 3.2.4 Piloter la plate-forme

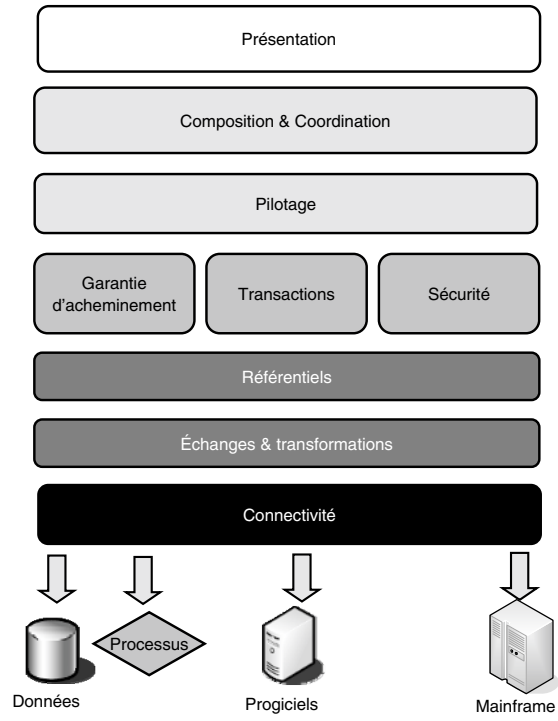



**Figure 3.1** — L'ensemble du cahier des charges technique d'une plate-forme d'intégration

La plate-forme d'intégration doit être capable de fournir un reporting aux directions métiers et à la DSI. Il s'agit ici de produire des vues et des états de synthèse sur le déroulement des processus métiers sous-tendus par les outils d'intégration. La console de suivi doit fournir une grande variété de mécanismes d'alertes, et s'intégrer aux outils décisionnels et de reporting, voire directement aux interfaces d'utilisateurs externes.

Il s'agit aussi, à terme, d'être capable d'ajuster les ressources selon la demande, de consolider l'expérience acquise par des pratiques de résolution d'incidents.

En complément, il est nécessaire d'assurer un suivi technique des échanges. Ce suivi bas niveau permet de remonter des alertes aux équipes d'exploitation sur des dysfonctionnements techniques. Ces dernières seront intégrées dans des outils classiques de monitoring.

## En résumé

Ce chapitre récapitule les exigences des métiers et de la DSI vis-à-vis d'un SI agile.

Les métiers expriment aujourd'hui leurs exigences en ces termes :

- Le SI doit être capable de s'adapter rapidement à une fusion ou à une réorganisation des équipes.
- Le SI doit être capable de fusionner ou faire coexister des socles techniques hétérogènes.
- Le SI doit être capable de déployer rapidement des processus sur la base d'un modèle fourni par les métiers.
- Le SI doit offrir des interfaces pour les clients et les fournisseurs.
- Le SI doit offrir une disponibilité à toute épreuve.
- Le SI doit offrir des indicateurs en temps réel sur l'activité de l'entreprise.
- Le SI doit pérenniser ses briques existantes et favoriser la réutilisation.

La figure 3.1 récapitule les exigences techniques.

On verra dans les parties 2 et 3, que SOA apporte des réponses méthodologiques et architecturales à l'ensemble de ces exigences.

Par ailleurs, la partie 4 présentera en détail, dans le cadre des Web Services, les technologies qui répondent aux exigences techniques.

Enfin, les parties 5 à 7 proposeront une vision concrète, technique et industrielle, de SOA.

# SECONDE PARTIE

# Expliquer les concepts SOA

SOA[1], Services et Web Services[2] sont des terminologies très à la mode. La plupart des publications, présentations ou conférences sur ces thèmes s'accompagnent quasi systématiquement de définitions le plus souvent différentes les unes des autres. En recherchant le terme SOA sur Google, il est possible d'atteindre plus de 40 millions de liens Internet, et près de 400 millions pour « Web Services » !

L'objectif de cette deuxième partie est donc de clarifier les concepts utilisés, en premier lieu la notion de services et d'architecture de services puis celle de composition de services. Il introduit une architecture de référence positionnant les différents concepts. Il clarifie également pourquoi les Web Services sont utiles, mais pas indispensables.

Le second chapitre s'attache à décrire les fondements du concept d'orientation service et notamment la notion de contrat et de messages.

Enfin le dernier chapitre introduit les outils à mettre en place de façon progressive en regard de l'architecture SOA de référence.

1. SOA (*Service Oriented Architecture*, ou architecture orientée services) étant l'acronyme anglo-saxon le plus couramment répandu, il sera utilisé tout au long de ce livre.
2. Web Services (WS) est la terminologie anglo-saxonne équivalente à Services Web. La notation abrégée WS-* est utilisée pour désigner les standards s'y rapportant.

# 4

# Urbanisation et architecture SOA

## Objectif

Le sujet SOA est très vaste et nécessite de définir précisément de quoi on parle ! En effet, ainsi qu'en témoigne une consultation rapide sur Internet, de nombreuses définitions sont proposées, depuis celles proposées par l'encyclopédie WikiPedia[a] jusqu'à celles des groupes de normalisation dédiés de l'OASIS[b]. Toutes ces définitions cherchent à établir un modèle de référence « standard » permettant une compréhension commune des éléments clés se rattachant à SOA.

L'objectif premier de ce chapitre est de proposer une définition synthétique et claire des notions cachées derrière les trois lettres « SOA ». À partir d'un modèle d'architecture, il présente les concepts clefs tels que le concept de services et le concept de composition de service, en mettant en évidence la relation qui doit s'établir entre les clients des services et les fournisseurs de ces services.

Le chapitre introduit ensuite le concept d'application composite, en s'attachant aux différences par rapport aux applications traditionnelles.

Une dernière partie se consacre à démystifier certaines idées préconçues sur ce qu'est ou n'est pas une Architecture Orientée Service.

a. Wikipedia consultable sur http://www.wikipedia.org.

b. OASIS – *Organization for the Advancement of Structured Information Standards*. Les membres du comité dédié et le modèle de référence SOA amorcé dès 2005 sont mentionnés sur http://www.oasis-open.org/.

# 4.1 SOA CONCRÉTISE LE MODÈLE D'URBANISATION

SOA, ou Architecture Orientée Service : le concept met d'emblée en avant l'importance de l'architecture du SI. La démarche SOA a en effet pour objectif général de faire évoluer cette architecture afin de répondre aux problèmes décrits dans la partie 1. Il est donc important de montrer comment le modèle général d'urbanisation vu en partie 1 évolue avec SOA. C'est l'objet de ce sous-chapitre.

La réponse SOA au syndrome des silos, propose une stratégie **d'anticipation** capable de se généraliser à l'échelle de l'entreprise. Cette stratégie repose et prolonge l'habitude prise par les urbanistes de modéliser des couches d'abstraction entre le monde réel des processus et les technologies sous-jacentes. Cette approche concrétise la **vue logique**, ou **vue des Services** (le terme étant progressivement défini à partir du prochain paragraphe).

La vue des services répertorie les fonctions métier sous-jacentes, pour beaucoup déjà existantes et établies, et les présente sous forme de services.

## 4.1.1 L'émergence de « pourvoyeurs de services » ?

**Qu'est-ce qu'un service ?** Un service, au sens SOA, met à disposition d'acteurs (humains ou logiciels) intervenants dans des processus métiers, un accès vers une ou plusieurs fonctions métiers.
Un service concrétise le lien entre la couche métier (constituant le côté **consommateur**) et les implémentations dans le SI (constituant le côté **fournisseur**) en prenant à sa charge un contrat (réalisé par le côté **pourvoyeur**).

Le regroupement de fonctions doit avoir un sens sur le plan métier : le consommateur du service n'a pas à se préoccuper de la façon dont ces fonctions sont implémentées et a fortiori des technologies sous-jacentes.

Le terme de service est introduit par analogie avec le monde réel : lorsqu'il utilise un service de distribution de billets de banque via un distributeur automatique de billets, le consommateur de ce service n'a pas à connaître le fonctionnement technique du distributeur et encore moins le détail de ses connexions avec les serveurs des différentes banques impliquées. Cette simplicité, liée à un effort de standardisation du dialogue homme-machine, permet à n'importe quel consommateur de (ré)utiliser le service avec un minimum d'effort cognitif.

Un service vise donc à être **simple d'emploi** et **réutilisable**.

Par exemple, un service de calcul de montant d'une commande permet à un utilisateur de ce service d'obtenir un montant toutes taxes comprises. Ce calcul peut faire appel à un moteur de tarification et à un calculateur de taxe, mais l'utilisateur n'a pas à s'en préoccuper – sauf éventuellement pour préciser l'unité monétaire à

utiliser. De même, l'utilisateur n'a pas ou plus à savoir que le moteur de tarification doit récupérer des informations du catalogue produit (tarif de base et promotions en vigueur), et accéder au progiciel de gestion de contrats clients (pour récupérer les avantages tarifaires consentis au client).

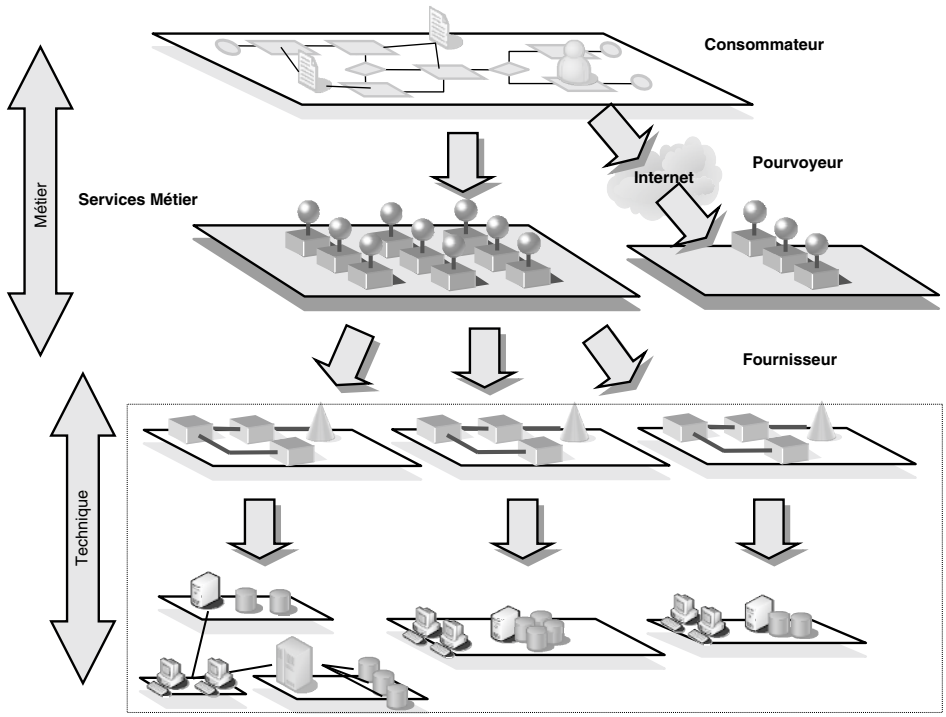



**Figure 4.1** — SOA, une liaison entre vue métier et vue informatique[1]

L'essor d'Internet permet également d'envisager d'accéder à des services offerts par le SI d'une autre entreprise. Réciproquement, il est envisageable d'ouvrir ses propres services au « monde extérieur » de façon contrôlée (accès extranet) ou non (accès internet). Un service doit donc être **interopérable** via Internet – au moins pour certains d'entre eux.

Pour faciliter la réutilisation et l'interopérabilité, tout service SOA devra fournir le résultat de ses traitements sous une forme normalisée, c'est-à-dire compréhensible par tous.

Un service SOA dialogue avec ses consommateurs sous une forme standardisée, tant sur le plan technique que sur le plan métier.

1. Les relations détaillées entre les différentes vues d'architectures répondent à des règles de cohérence 2 à 2. Elles ne sont pas illustrées par la figure pour éviter d'alourdir le schéma.

*La standardisation technique passe par l'utilisation d'un vocabulaire spécifiant les formats et la structure des données échangées. Le meta-langage XML s'impose souvent pour ce faire, même si l'on peut tout à fait utiliser SOA sans XML (cf. syntaxe .NET, Java, etc.). La standardisation métier passe en revanche obligatoirement par la promulgation d'un langage décrivant la sémantique des données échangées. Certaines grammaires XML (RDF, OWL) vont aussi dans ce sens, même si d'autres conventions sont possibles.*

Il devient ainsi progressivement possible de sortir du fameux paradigme du « plat de spaghettis » pour passer, au cas par cas, au « plat de lasagnes », en couche.

### *Caractéristiques générales d'un service*

L'analogie avec le monde réel permet de déterminer les principales caractéristiques d'un service.

**Vu du consommateur**, le service doit offrir une **valeur ajoutée** ainsi qu'une **garantie de qualité** du service rendu.

Un service de distribution de billet de banque ne sera pas utilisé s'il est trop lent ou s'il commet des erreurs. Le consommateur perçoit donc un service comme :

- un composant respectant un **Contrat de service** : tous les distributeurs respectent le même contrat de service, à savoir délivrer des billets après que l'utilisateur ait fourni les informations nécessaires;
- un composant de type **Boîte Noire** : il n'est pas nécessaire de savoir comment cela fonctionne;
- un composant **autonome** : les dépendances éventuelles entre services ne sont pas visibles.

**Vu du pourvoyeur**, le service est le plus souvent synonyme de compétition et donc d'amélioration de **la valeur ajoutée**.

Par exemple, la concurrence entre banques pousse ces dernières à enrichir le service fourni par un distributeur (consultation de compte, recharge de carte etc.) et à en améliorer l'ergonomie. Le pourvoyeur devra donc proposer des services :

- **Homogènes** (donc **faciles à utiliser** pour le client) : un service de distribution est quasiment standardisé dans chaque pays (sans avoir à lire 300 pages de notice !).
- **Performants**.
- **Offrants une qualité mesurable :** ceci permet de choisir le plus compétitif à un instant donné.
- **Indépendants du contexte du client :** c'est un gage de réutilisation pour le bénéfice du plus grand nombre, même si des éléments contractuels peuvent venir dicter des restrictions d'utilisation.

### 4.1.2 EDA, POA, SOA… Querelle de chapelles ou vrai débat ?

Même si SOA est le nouveau terme à la mode, il ne faut pas pour autant jeter aux oubliettes de l'histoire des modèles architecturaux intéressants tels que EDA ou POA. Il est en effet intéressant de positionner ces modèles pour affiner la présentation de SOA.

**EDA (*Event Driven Architecture*)** est un modèle d'architecture posant comme principe fondateur la propagation automatisée des événements métiers dans le Système d'Information. L'objectif est d'éviter, autant que possible, la désynchronisation de multiples référentiels[a], et surtout d'assurer un traitement en temps réel de ces événements, ni trop tard ou, pire, jamais effectué.

a. Un événement étant souvent porteur de nouvelles informations à dupliquer dans le SI pour cause d'hétérogénéité de ce SI.

Ce modèle a donné lieu, initialement, à la mise en place des fameux outils EAI (sur la base de MOM[1] événementiel). Le problème est qu'en ne touchant pas aux applications elles-mêmes, l'EAI suppose que ces applications sont capables de recevoir un événement métier entrant, de le traiter, et de renvoyer si besoin un événement métier vers d'autres applications. Autrement dit, elle suppose que le processus de traitement de l'événement métier est pris en compte implicitement dans chaque application concernée. Cette hypothèse est le plus souvent fausse : EDA ne permet donc pas une réelle automatisation des processus métiers. D'où l'émergence du modèle POA.

**POA (*Process Oriented Architecture*)** est un modèle d'architecture posant comme principe fondateur que toute application du SI doit être modélisée comme un processus, et comme conséquence la nécessité de mettre en place un moteur (de type workflow) pour automatiser ces processus.

Le point de départ de POA est important : en considérant que toute application du SI traite en réalité un événement métier, ce modèle recentre les efforts de modélisation sur l'objectif prioritaire du SI, à savoir la capture, le traitement et l'historique des événements métiers.

Le problème principal est que ce modèle ne se préoccupe pas vraiment de la façon d'interfacer les processus métiers avec les applications existantes. D'où l'avantage de l'approche SOA.

On peut cependant considérer que l'évolution certes récente de SOA conduit à intégrer la problématique POA dans le modèle SOA. Autrement dit, le modèle SOA propose non seulement une vision « service » de l'architecture du SI, mais également une vision « processus » (cf. chapitre 9). On peut également considérer que SOA constitue l'aboutissement de la démarche EDA, en prenant en compte les aspects « propagation d'événement et synchronisation des référentiels ».

1. MOM – *Middleware Orienté Messages*.

# 4.2 AU CŒUR DE SOA : LE CONCEPT DE SERVICE

## 4.2.1 Identifier les services SOA

SOA propose donc un modèle d'architecture informatique basé sur l'émergence d'une couche de services.

Ces services offrent une vue « logique » des traitements et données existant déjà ou à développer. Chaque service encapsule ces traitements et données et masque ainsi l'hétérogénéité du système d'information. L'exemple suivant (figure 4.2) – illustre le propos.

Un service « gestion des clients » offre par exemple une « vision client » unifiée. Pour cela il agrège les informations éparses dans le SI : il récupère la fiche d'identité du client et les noms des contacts dans le logiciel CRM, le contrat du client dans l'ERP et les éventuels contentieux dans un outil spécialisé. Tout ce travail d'agrégation est masqué aux utilisateurs de ce service.

Le service « gestion du catalogue produit » met à disposition les informations sur les produits de l'entreprise. Ce service masque l'accès à l'ERP (récupération des tarifs), l'accès à l'outil de PLM (récupération de certaines caractéristiques techniques), et l'accès au Catalogue Marketing (promotions en cours).

Le service « gestion des stocks » met à disposition les informations récupérées soit dans le mainframe de l'entreprise (pour les usines et les entrepôts) soit dans les serveurs des filiales non intégrées.

> **Remarque importante :** les grands progiciels propriétaires sont eux aussi condamnés à adopter cette approche SOA pour être plus **flexibles**. Ils doivent en effet autoriser une réutilisation plus facile de leurs fonctions dans des processus traversant les frontières des silos applicatifs, au sein d'une entreprise, voire autoriser cette réutilisation dans des processus traversant les frontières des entreprises.

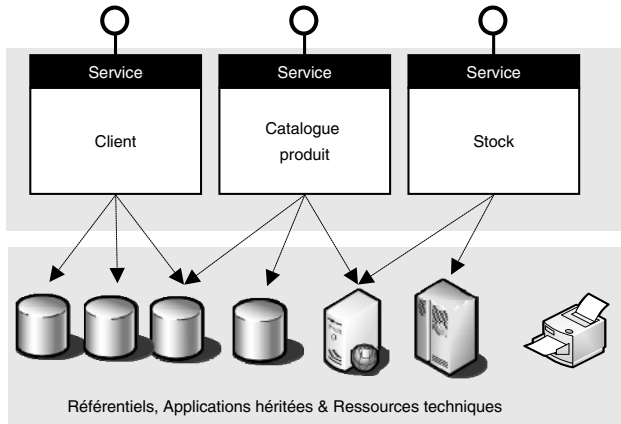



**Figure 4.2** — Encapsuler l'existant par des services

### 4.2.2 Construire de nouveaux services

Bien évidemment l'exemple qui précède est incomplet. Pour le compléter, on peut par exemple introduire la gestion de prise de commande. Lors d'une prise de commande, il faut d'abord vérifier que le client existe et son contrat est valide, puis vérifier que les produits existent dans le catalogue, déterminer ensuite quels entrepôts possèdent les stocks pour fournir les produits commandés et enfin lancer la livraison.

De plus, un partenaire commercial (concessionnaire, agent…) doit pouvoir effectuer une prise de commande via Internet.

La solution au sens SOA est d'ajouter un service « gestion de commande » – figure 4.3 – qui utilisera les services déjà en place pour exécuter le traitement décrit ci-dessus.

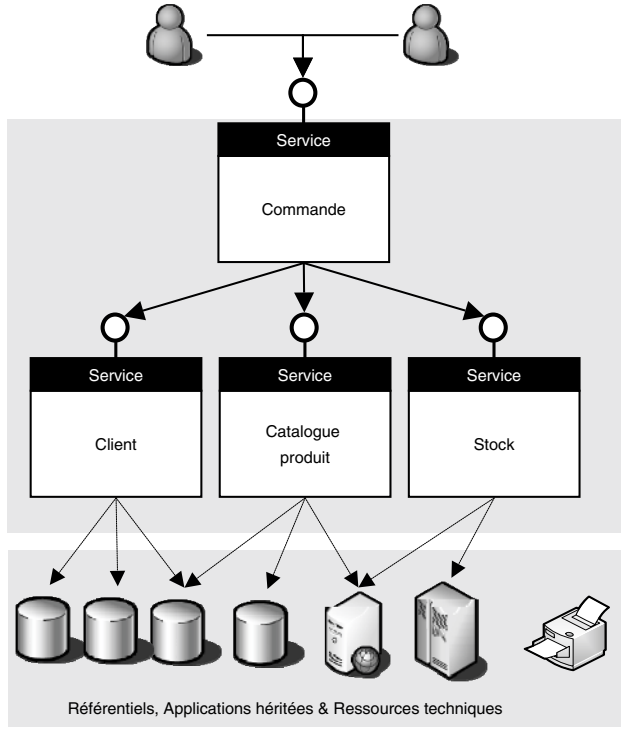



**Figure 4.3** — Création d'un nouveau service par composition

Cet exemple met en lumière un principe très important de l'approche SOA :

L'approche SOA favorise la construction de nouveaux services par **composition** de services existants.

La composition de services ne s'arrête pas non plus aux frontières du SI. Admettons, par exemple, que l'on souhaite compléter le service « Gestion de commande » pour permettre à un utilisateur de demander au service, lors de la prise de commande, le calcul de la date et de l'heure de livraison. On veut également lui permettre de demander, lorsque la commande a été prise, d'interroger le service pour savoir où en est la livraison.

La solution au sens SOA, illustrée par la figure 4.4, sera de recourir à un service offert par le SI du transporteur choisi, le service « gestion des livraisons ». Le service « commande » fera appel à ce service, d'abord pour obtenir une date prévisionnelle de livraison, ensuite pour connaître l'avancement de cette livraison. On remarquera au passage que ce concept de service peut avoir une traduction au niveau financier, puisque l'accès à ce service logistique peut dans certains cas être payant (cf. concept de SLA détaillé au chapitre suivant). C'est un exemple concret du rapprochement informatique ↔ métier favorisé par SOA.

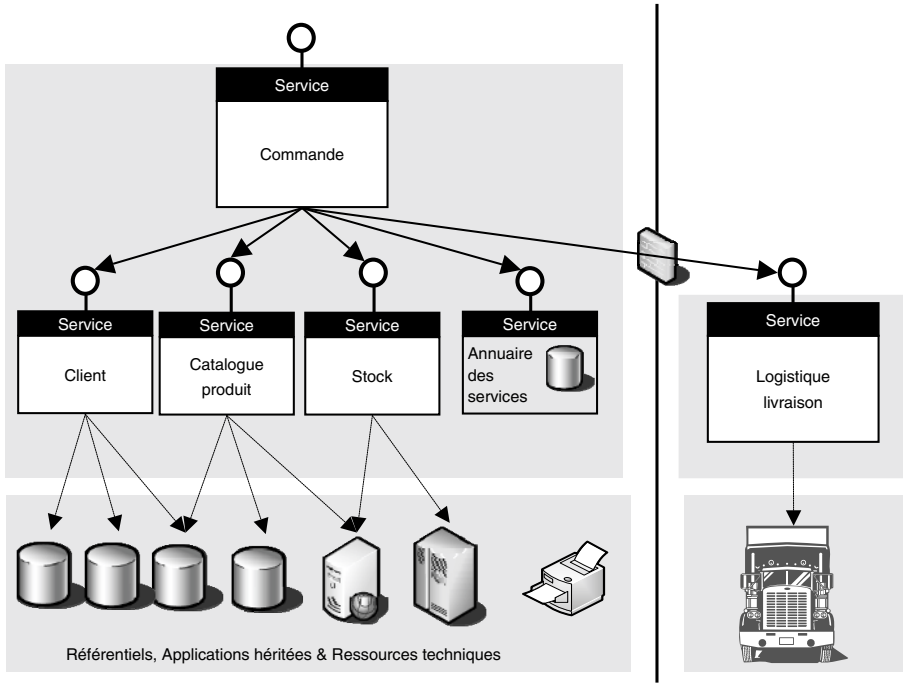



**Figure 4.4** — Composition et interopérabilité

Cet exemple met en évidence quelques points clefs dans la démarche d'identification des services :

- l'importance pour le service composite d'exposer à son tour une interface via un contrat ;
- l'importance du concept de **granularité** (composition de compositions) ;

- le besoin du maintien d'un contexte entre les services (cf. le paragraphe « La nécessité du contexte de composition » introduit dans le chapitre suivant) : ici par exemple, les lignes de commande ;
- l'intérêt d'utiliser un **référentiel des services « homologués »** pour pouvoir choisir entre différents services logistiques de livraison ;
- le type de composition à choisir pour organiser la séquence d'invocations de service (orchestration automatisée versus enchaînement humain de tâches).

### 4.2.3 Gérer le cycle de vie des services

#### *Identifier les services à mettre en place*

L'identification des services est une étape clef pour le succès de SOA. Quels sont les critères d'identification ? Un service est identifiable et n'a de réelle justification que s'il satisfait au moins un des trois critères suivants :

- **Le service est mutualisé** au sein d'une forte population d'applications consommatrices – informatiquement, il permet la réutilisation.
- **Le service facilite l'intermédiation** avec un fonctionnement existant interne à un SI *(par exemple, il est plus simple d'aller au bureau de Poste, que de porter chaque fois un colis chez son destinataire !)* – informatiquement, il permet l'interopérabilité, offerte via Internet à d'autres SI.
- **Le service est nécessaire à un enchaînement** plus global d'activités via un processus métier – informatiquement, il autorise son emploi dans une composition.

#### *Mettre en place les services*

Puisqu'il doit relier concrètement des processus/tâches métiers avec des applications et progiciels évolutifs et répartis, un service n'est pas uniquement un concept : il existe bel et bien en tant que **composant logiciel exécutant** une tâche spécifique plus ou moins ambitieuse pour le consommateur (par exemple : Rechercher une information, exécuter un processus de prise de commande).

La mise en place d'un service distingue donc d'une part la modélisation du **contrat** qui **décrit le service** rendu, d'autre part **l'implémentation** du service, qui doit respecter le contrat défini, et enfin les modalités de **déploiement**.

Les modalités de déploiement incluent la localisation du service dans la vue Physique (adresse du service), le protocole d'accès à ce service, ainsi que les contraintes de sécurité ou de qualité d'exécution à respecter.

Remarquons cependant que le modèle SOA ne préjuge nullement du fait qu'un service soit simplement un aiguilleur de la demande de service vers un logiciel existant : le service peut implémenter lui-même le traitement demandé. Dans ce cas le pourvoyeur se confond avec le fournisseur de services.

Distinguer contrat versus implémentation d'une part, et implémentation directe du traitement versus « aiguillage », a bien entendu pour avantage déterminant de pouvoir faire évoluer le SI par étape : dans une première étape, le service délègue un traitement à un existant, dans une deuxième étape le service récupère à sa charge le traitement pour de multiples raisons (performances, coût de maintenance de l'existant, volonté de mettre en place un moteur de règles de gestion externalisant les règles de calcul, etc.). Si le service continue à respecter le même contrat entre les deux étapes, les clients du service n'auront pas à être modifiés.

#### *Maintenir les services*

La gestion des évolutions d'un service est un point clef, qui passe par la **gestion de versions** et d'évolutions de composants logiciels (à ne pas confondre avec la gestion de variantes décrites dans le chapitre 5).

Cette gestion n'est pas en soi une nouveauté apportée par SOA ; cependant, les services étant la pierre angulaire de l'approche, SOA impose une mise en place rigoureuse de cette gestion.

En outre, il pourra s'avérer très utile, notamment en cas de multiples consommateurs d'un même service, de savoir distinguer :

- les principaux statuts d'un service. Par exemple : « En Définition », « En Test » *(ou « Bouchoné[1] »)*, « En Production » *(« ou Public »)* et « En Retrait » *(ou « Obscolete »)* ;
- les principaux rôles d'intervention et leurs taches associées à chaque évolution de statut : le responsable catalogue, le responsable solution métier[2], le responsable conception, le responsable implémentation, le responsable exploitation, le responsable des services transverses, etc.

Cela met en exergue de façon plus générale l'importance de l'outillage associé à la démarche : le chapitre 6 présente un premier aperçu de cet outillage, puis la partie 7 le détaille.

## 4.3 ARCHITECTURE DE RÉFÉRENCE SOA

### 4.3.1 Le concept d'application composite

**Qui peut consommer des services ?** Répondre à cette question nécessite d'introduire un nouveau type d'applications, qualifiées de « **composites** » dans une appro-

1. L'analogie du bouchon désigne ici une implémentation provisoire permettant d'utiliser le service avant que sont implémentation véritable ne soit rendue disponible.
2. Voir le chapitre 7.

che SOA, c'est-à-dire construites en composant des appels de services. Remarquons bien que l'approche Service ne peut apporter sa valeur ajoutée aux utilisateurs finaux que via ces applications composites.

Une architecture de référence se dessine donc peu à peu pour toute architecture orientée service.

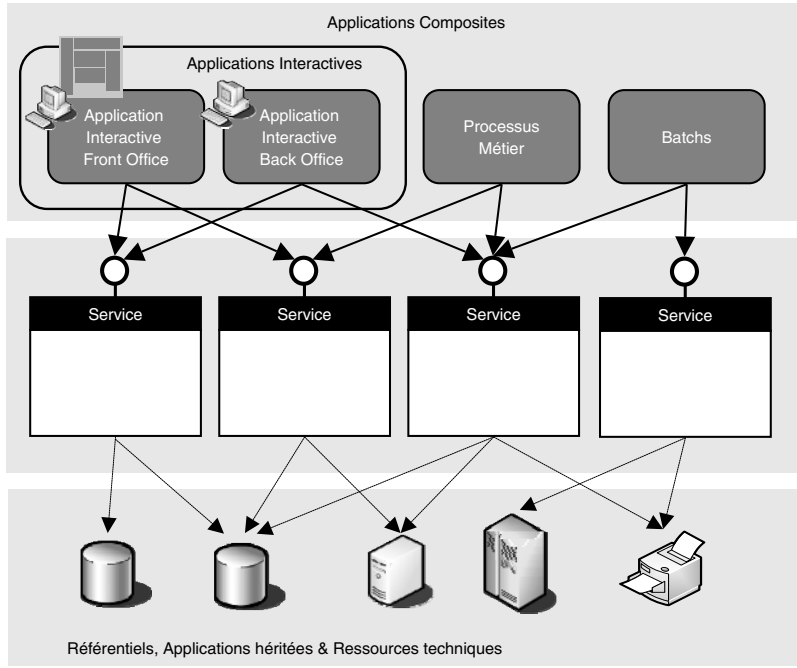



**Figure 4.5** — L'architecture de référence SOA

Un service peut donc être consommé soit par :

- un utilisateur, via une application composite interactive,
- un système traditionnel (applicatif non SOA : **batch** ou autre application),
- un processus métier,
- un autre service SOA.

### *La nécessité du contexte de composition*

Lorsque le service est partie prenante d'une composition, il doit être sollicité en cohérence avec les autres services composés. Ceci implique de partager un contexte d'échange entre les services. Ce **contexte** permet le passage d'informations entre plusieurs services, que ce soit des informations métier ou des informations à caractère

technique, sécuritaires (par exemple, ticket d'authentification, etc.) ou transactionnels (par exemple : date de début d'une transaction, etc.).

Cependant tous les services n'ont pas forcément besoin d'un contexte transactionnel. Ceci est d'autant plus vrai si la ressource du SI encapsulée par le service n'est pas elle-même transactionnelle *(e.g : un serveur de fichiers)*. Dès lors, un service ou une composition de service doivent expliciter leur mode de prise en compte du contexte :

- **Service sans contexte :** L'échange contient l'intégralité des informations requises par le pourvoyeur.
- **Service avec contexte :** Le contexte est passé par valeur ou par variable durant les échanges entre le consommateur et le pourvoyeur. Ceci oblige le service à connaître plus de chose sur le client et pourra donc influencer l'évolutivité, la performance, et la réutilisation du service.

L'approche SOA implique de gérer le concept de contexte. Le contexte n'est pas uniquement transactionnel, mais aussi métier.

Il est important de noter que le Contexte est fourni au service par son consommateur : le service n'a pas à mémoriser (stocker) ce contexte. **Tout service doit être sans état**.

Cette problématique est liée au concept de transaction longue. Ces éléments sont détaillés au chapitre 10, dédié à la modélisation des applications composites.

### *Une application composite devient à son tour un service !*

Ces applications peuvent s'exposer à leur tour sous forme de service contractualisé pour leurs consommateurs.

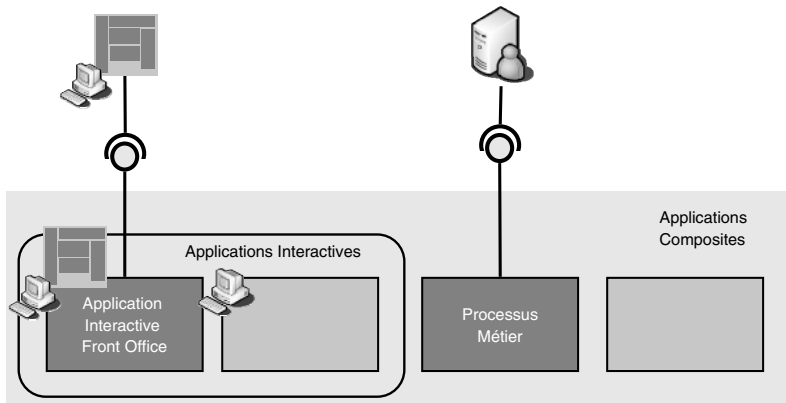



**Figure 4.6** — Une composition est un service

### 4.3.2 Typologie des applications composites

*Zoom sur l'orchestration de services au sein d'un processus métier*

Un processus métier au sens SOA est vu comme un enchaînement plus ou moins automatisé de services effectué par un unique intermédiaire, le « chef d'orchestre ». D'où le nom d'**orchestration de processus** pour ce type de composition. La logique de cet enchaînement (séquentiel, parallèle, etc.) est décrite, comme dans le workflow classique, par un modèle d'orchestration directement exécuté par le chef d'orchestre, appelé moteur d'automatisation de processus (BPM[1]). L'orchestration décrit également la logique de contrôle et le flux de données internes d'un processus métier.

> **Quelle différence entre workflow traditionnel et processus métier SOA ?** Un processus SOA exclut a priori l'homme dans la boucle. Il s'agit donc de processus de type e-Business, recevant et traitant automatiquement les événements métier.

On verra au chapitre 9 que cette différence est dans la réalité beaucoup moins tranchée. Les fournisseurs de solutions distinguent traditionnellement « l'orchestration de processus », pour désigner une coordination n'intervenant qu'au niveau de services métiers, de « l'orchestration de services » désignant un processus d'intégration technique avec une ou plusieurs sources back-ends y compris des workflows applicatifs invoquant des services techniques d'accès bas niveau ou natif à certaines ressources (mainframe, système de messagerie, etc.).

*Zoom sur la composition interactive de services*

L'architecture « MVC[2] » est l'architecture de référence des applications composites interactives. MVC est bien connue dans le monde de la programmation objet et permet d'isoler d'une part un modèle métier, d'autre part des vues interactives rendues par un moteur graphique sur de l'information en provenance du modèle, et enfin des contrôleurs de dialogues. Cette architecture doit être revue avec l'approche SOA (cf. le chapitre 10 pour plus de détails) :

- Le modèle métier n'est plus local à l'application, mais encapsulé dans un ou plusieurs services métier. Le « contrôleur MVC » doit donc coordonner l'appel à ces services et se transforme donc en « coordinateur », capable d'effectuer une séquence d'invocations attendues.
- Au fur et à mesure des invocations, le coordinateur doit en général préserver un ensemble d'informations communes entre les services. Le « modèle MVC » est donc remplacé par un contexte local à l'application (voir ci-dessus le paragraphe « La nécessité du contexte de composition »).

1. BPM – *Business Process Management*.
2. « MVC – *Model-View-Controler* » est la terminologie anglo-saxonne signifiant Modèle – Vue – Contrôleur.

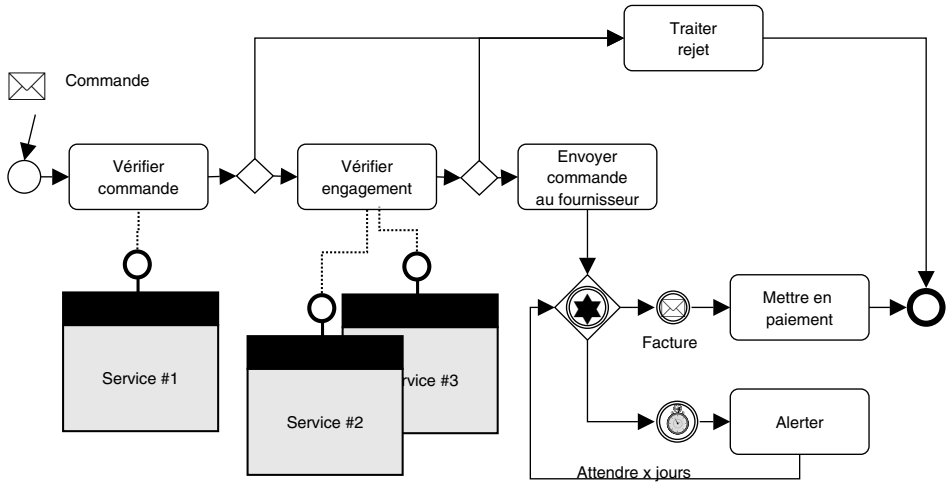



**Figure 4.7** — Illustration d'une composition orchestrée de services

*Zoom sur l'ordonnancement de services*

Un batch peut lui aussi réutiliser des services (*e.g : le calcul de facturation d'un contrat d'assurance lors de sa tacite reconduction est en général effectué en batch*). Le point clef est alors classiquement les performances de ces services. Il peut être nécessaire d'aménager le contrat de services pour prendre en compte ce type d'utilisation du service.

### 4.3.3 Topologie des applications composites

Il existe plusieurs topologies de déploiement client/serveur des applications composites, comme l'illustre la figure 4.8.

*Topologie client lourd*

L'application composite (Moteur graphique et implémentation « MVC revisitée ») réside, dans ce cas, sur le côté poste client. Elle élabore les vues en local sur ce poste client. Elle appelle des services hébergés par le(s) serveur(s). La communication client ↔ services peut soit utiliser XML, soit reposer sur des technologies moins gourmandes en bande passante mais confinées à l'intérieur du SI (EJB, DCOM…).

*Topologie client léger*

L'application composite réside sur le serveur : elle élabore les vues et envoie ces vues à un moteur graphique distant : le navigateur Web. L'application composite fait alors appel à des technologies de pages web dynamiques spécifiques, plus ou moins standardisées (*e.g. jsp, jsf, struts, asp, asp.net,* etc.).

*Topologie client distant*

L'application composite devient dans ce cas un service. Le client est alors un portail distant, qui utilise un protocole de communication dédiée pour dialoguer avec ce ser-

vice. L'application composite transmet ses vues sous un format standardisé (cf. WSRP[1] décrit dans la partie 4). Le portail distant va alors agréger différentes vues, fournies par ses propres applications locales et par l'application composite distante.

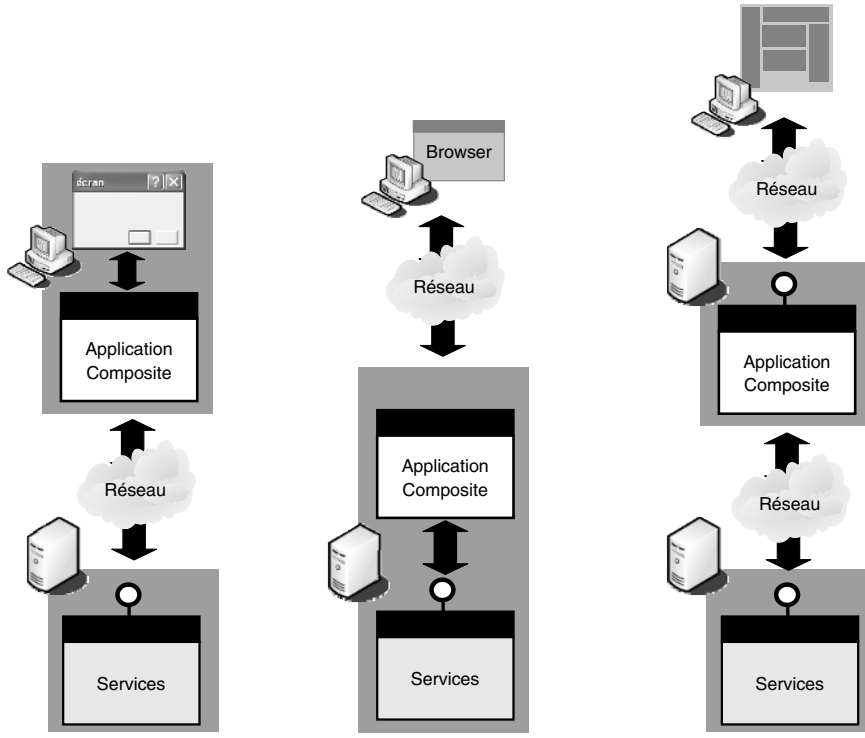



**Figure 4.8** — Topologie de déploiement des applications composites

## 4.4 POUR UNE DÉMARCHE SOA GRADUÉE

### 4.4.1 Pourquoi débuter une démarche SOA ?

La démarche SOA peut obéir à plusieurs types de stratégie de la Direction Informatique et/ou des maîtrises d'ouvrage :

- Stratégie « *pilotée par les métiers* » : les besoins métiers imposent à l'informatique d'évoluer.
- Stratégie « *pilotée par les technologies* » : l'évolution des technologies offre des opportunités aux métiers (par exemple, l'introduction des mobiles et Internet comme nouveaux canaux de communication).

1. WSRP – *Web Services for Remote Portlets*, cf. la partie 4.

- Stratégie « *pilotée par les coûts* » : des coûts de maintenance trop élevés, la surmultiplication de licences, etc., peuvent imposer une refonte profonde de certains silos du SI.

L'étude des besoins métiers fait ressortir trois axes fondamentaux :

- La mise en place de **processus** métiers rationalisés et automatisés (e-Business) : ce besoin est lié à l'essor d'Internet comme canal de communication entre entreprises permettant une intégration en quasi « zéro délai » de leurs SI respectifs. Ce besoin est certainement l'incitation la plus forte actuellement pour entamer une démarche SOA, car elle présente un retour sur investissement potentiel attrayant pour les maîtrises d'ouvrage.
- La capitalisation des informations notamment pour faire émerger la fameuse « **vision client** » **unifiée** dans l'optique client/fournisseur : ceci peut nécessiter d'agréger et de mettre en correspondance des données disparates.
- La suppression des multiples saisies manuelles afin de garantir la **synchronisation des informations dupliquées**, en définissant un seul maître pour chaque information dupliquée (afin de supprimer les multiples saisies manuelles).

### 4.4.2 Comment débuter ?

SOA ne se décrète pas : il est nécessaire de mettre en place une démarche progressive d'adoption.

Sur le plan de l'urbanisation, cette démarche ciblera en priorité le silo métier dont les modifications peuvent laisser espérer un retour sur investissement rapide. Elle s'étendra progressivement aux autres silos.

Sur le plan de l'organisation, on mettra en place en priorité un plan d'acculturation des équipes, y compris les équipes de production, trop souvent négligées au démarrage alors qu'elles devront, comme toujours, finir par assumer le fonctionnement du SI, 24 heures sur 24 et 365 jours par an, quelles que soient ses évolutions. On définira en parallèle un cycle de vie global, tel que celui présenté dans le chapitre 11.

### 4.4.3 Quelles conséquences sur le SI ?

Le SI doit répondre à des exigences plus techniques, à savoir :

- **Une garantie de performance :** Cela implique de décrire, mesurer et contrôler une qualité des prestations à la fois en amont de la sollicitation de services (il s'agit de l'accessibilité), pendant l'exécution (il s'agit de la fiabilité) et jusqu'en aval (il s'agit alors de la disponibilité).
- **Une fourniture d'infrastructure :** Elle fait face à l'évolutivité et au changement de taille prévisible tout en se prémunissant de l'hétérogénéité technologique (donc des formats, protocoles, patrimoine existant, etc.) mais aussi de

la complexité des métiers, en prenant en compte des variantes spécifiques, des réglementations particulières, etc.

- **L'intégrité des biens informatiques :** Son but est de maintenir des référentiels ou chaque fonction n'a plus qu'une et une seule place de référence dans le SI, tout en assurant la préservation, à la fois en interne et en externe, des données, des échanges et des systèmes.

### 4.4.4 Quels problèmes clés ?

Que le périmètre d'une démarche SOA soit à l'échelle d'un îlot, d'un silo applicatif ou du SI, elle reste empreinte d'une ambition certaine. Elle nécessite donc de bien appréhender les risques encourus. Ces risques portent notamment sur :

- La modélisation adéquate des services et des processus (partie 3, chapitres 8 et 9).
- La conception des applications composites, en tenant compte des spécificités de SOA (partie 3, chapitre 10).
- La maturité progressive des équipes, en matière de pratiques d'urbanisation, de développement et d'intégration (partie 3, chapitre 11).
- La compréhension des normes et standards (partie 4) et leur maîtrise (partie 5).
- Le choix des outils et le déploiement de ces outils en une infrastructure cohérente, compatible avec l'existant et administrée (partie 7).

### 4.4.5 Premier bilan

Cet ouvrage s'efforce de présenter le périmètre des notions à connaître pour envisager au mieux l'évolution du SI dans le respect du cadre SOA, que cette évolution soit ambitieuse ou plus modeste.

Il est cependant important de noter une divergence certaine d'intérêt entre les différents acteurs du monde SOA. Beaucoup d'éditeurs se préoccupent surtout d'outiller la vue « technique », la plus liée aux technologies d'infrastructure et aux serveurs d'applications. Cependant l'objectif principal d'une entreprise s'engageant dans une démarche SOA concerne la **rationalisation du métier et de l'organisationnel** (par conséquent la vue métier et la vue logique ou « vue des services ») : par conséquent, il est bien plus important de raisonner architecture, méthodologie et organisation, que de raisonner technologies et normes.

SOA est, d'abord et avant tout, une démarche architecturale et organisationnelle : le choix des technologies et des outils reste secondaire. Il est parfaitement possible de mener une démarche SOA sans utiliser les Web Services, même si ces outils présentent des avantages indéniables (cf. partie 5).

L'approche SOA défendue dans ce livre souhaite donc avant tout fournir des éléments pour satisfaire cette ambition « métier » : d'où l'importance des aspects modélisation de l'architecture (partie 3), la nécessité de faire la part de l'utile et du futuriste dans le catalogue de normes disponibles (partie 4), l'utilité d'avoir un premier retour d'expérience sur la technique sous-jacente (partie 5), et enfin la nécessité de se doter de critères de comparaison d'une offre désormais pléthorique (partie 7).

## 4.5 QUELQUES IDÉES REÇUES SUR SOA

### 4.5.1 Ce que SOA n'est PAS !

Plusieurs amalgames sont entretenus par le marketing de fournisseurs en mal de vente, qu'il est nécessaire de clarifier ici.

**SOA n'est pas une technologie.** SOA est d'abord un ensemble de concepts constituant un modèle cohérent d'architecture pour faciliter la flexibilité du SI via l'émergence de services intégrant/réutilisant des applicatifs existants et, par ailleurs SOA est une démarche progressive d'application de ce modèle.

SOA est utilisable qu'il s'agisse de technologies JEE, .NET, ou autres, via des implémentations propriétaires ou Open Source, etc. Ce modèle d'architecture délègue en outre à une plate-forme transversale l'infrastructure d'intégration.

**SOA ne signifie pas Web Services.** À l'intérieur des frontières du SI, SOA peut se réaliser sans Web Services, sans même d'ailleurs utiliser le Web ou les technologies du Web.

**Web services ne signifie pas SOA. À l'inverse, il est aussi erroné de croire qu'en exposant des Web Services, tous les** concepts SOA seront respectés (cf. la présentation de REST dans la partie 4).

**SOA n'oblige en rien au synchrone.** Au contraire l'intégration asynchrone de services est tout autant possible voire recommandable pour pallier des pics de montée en charge où la fiabilité pourrait se retrouver compromise en utilisation synchrone. *(e.g. : L'appel synchrone de deux services respectivement disponibles à 99 %, fournit un service qui ne sera plus disponible qu'à 99 x 99 = 98,01 %)*.

**SOA n'est pas de l'orienté objet avec un peu plus de marketing.** La différence fondamentale est que l'approche Objet a souvent été dans le passé synonyme de « big bang » et a de ce fait découragé les bonnes volontés malgré les promesses (réelles) de réutilisation. L'approche SOA est de ce fait plus pragmatique.

L'approche orientée objet et l'approche orientée services sont cependant très complémentaires, l'une comme modèle d'architecture l'autre comme technologie efficace d'implémentation de ce modèle pour les nouvelles applications et services.

**SOA ne se réduit pas à la mise en place d'un broker d'intégration (EAI).**

Une hirondelle ne fait pas le printemps, et la mise en place d'un outil ne fait pas émerger spontanément les services et encore moins les processus métiers.

**SOA n'est pas une solution miracle** : SOA n'est pas une solution clé en main, ni un dogme générique universellement applicable. Elle nécessite un effort en continu, hors de tout effet de mode.

Une architecture par construction se doit d'être toujours adaptée aux besoins de chaque entreprise (existant, schéma directeur, budget, planning, maturité des équipes, etc.). De ce point de vue, l'émergence de la couche de services nécessite un travail en continu, pour identifier les services « à valeur ajoutée », vérifier que les services répondent aux besoins qui émergent, et réciproquement que les besoins émergeant ciblent bien les services existants.

Il s'agit également d'un effort global au niveau du SI ou tout au moins d'un silo important du SI. La démarche SOA reste donc inopportune pour les applications embarquées à fortes contraintes techniques ou les petites applications départementales. Le niveau minimal de besoin est typiquement l'émergence de services métiers pour la création, la consultation, ou la modification sur des entités métiers (*e.g : le client*, *la facture*, *le fournisseur*, etc.) ou des services techniques (*e.g : les traces*, *les notifications*, *les impressions*, etc.) (cf. chapitre 8).

### 4.5.2 Ce que SOA ne dit PAS !

**SOA ne résout pas les problèmes existants dans les implémentations** : Si les concepts SOA peuvent apporter un cadre pour l'agilité et la réutilisation d'un existant, SOA ne résout pas pour autant les problèmes d'implémentation d'un existant telle que la simplification ou l'optimisation d'un code vieillissant ou trop monolithique, la non adéquation des algorithmes métier ou leur difficulté à évoluer pour supporter de nouvelles règles de gestion, des performances insuffisantes, etc.

Ces éléments pourront néanmoins désormais être traités en procédant silos après silos, sans impact pour les applications consommatrices dès lors que celles-ci feront références à des descriptions stables de contrats de services (cf. chapitre 5).

**SOA nécessite un langage métier commun.**

Les avantages de la contractualisation ayant été couverts précédemment, il reste important de souligner néanmoins la difficulté au sein d'une entreprise et a fortiori pour une ou plusieurs entreprises, de se mettre techniquement d'accord sur un langage métier commun, ou langage « pivot », pour faciliter l'échange de données valide dans le temps et pertinent en terme d'usage (exactitude et précision par rapport au besoin).

Au-delà du principe général de l'utilisation d'un tel langage pivot, l'ensemble de cet enrichissement sémantique se traduit par l'ajout de « méta-données » exprimées le plus souvent dans une grammaire XML au moyen de balises plus ou moins normalisées avec leurs valeurs associées. Il doit par ailleurs conserver un caractère extensible, permettant d'accueillir une communauté grandissante de consommateurs.

**SOA est une affaire de compromis.**

Sur la centralisation des services versus le clonage de ces services : l'avantage de la réactivité lié à l'unicité des services à un niveau global, doit être confronté à la performance qu'autorise localement la redondance des données et des traitements.

Sur un langage pivot généralisé à l'ensemble de l'entreprise versus des langages locaux : adopter un langage pivot dans l'échange permet de limiter les risques d'incompréhension entre consommateurs et pourvoyeurs de service. Pouvoir étendre, ou particulariser les échanges au gré des changements requis, évite que le consommateur et le pourvoyeur se retrouvent bloqués par une version spécifique de service (cf. chapitre 5 sur la notion de message et de gestion de variantes).

Sur l'ouverture du SI versus la sécurité : l'ouverture du SI, notamment au moyen des services Web, possède une contre partie en matière de sécurité. À niveau de sécurité constant, les surcoûts d'investissement (DMZ, Firewall, VPN, etc.) et d'exploitation induits par l'ouverture du SI ne sont pas négligeables.

**SOA ne résout pas la gestion des performances en production** : l'idée du service unique et réutilisable reprise dans un contexte d'exécution synchrone exacerbe les problématiques de disponibilité et de capacité à monter en charge différents services.

## En résumé

SOA se traduit dans l'architecture, par une vue dédiée à la convergence et l'unification du SI : la vue des Services.

Un service est **contractualisé** entre un consommateur et un pourvoyeur. La **composition** de service permet d'envisager tout type de flux d'interaction avec les services en respectant les principes liés à la notion de service.

Vu de l'utilisateur final, la valeur ajoutée des services se perçoit au travers **d'applications composites**, dont il convient de comprendre la typologie et de spécifier la topologie de déploiement.

SOA se met en place au travers d'une démarche fixant un point d'équilibre conciliant plusieurs compromis à la fois, dont principalement le retour sur investissement attendu, les coûts et délais, la performance et bien sûr la flexibilité.

# 5

# Au cœur de SOA : le concept d'orientation service

**Objectif**

Ce chapitre approfondit le concept de service, pierre angulaire de la démarche SOA.

Tout s'organise autour du **contrat de service**, indissociable du concept de service, et en détaille progressivement les éléments constituants. Ce faisant, le chapitre parcourt le cycle de vie d'un service, depuis son identification jusqu'à son déploiement.

Dans ce parcours du cycle de vie, le chapitre ne présente que les points spécifiques de la démarche SOA par rapport au cycle de vie classique d'un composant logiciel.

## 5.1 FORMALISER LES SERVICES

### 5.1.1 Service et contrats

Le concept de service a fait ressortir ce qui relie d'un côté des **consommateurs**, ou **client, requérant** le service, et de l'autre des **pourvoyeurs** capables de proposer le service (*e.g. Lire l'historique de commande, Mettre à jour le statut de livraison*, etc.). Il est dès lors nécessaire de formaliser ce lien entre consommateur et pourvoyeur, afin que la réutilisation du service en soit facilitée et, s'il y a lieu, son interopérabilité garantie : on introduit alors comme on l'a vu le concept de **contrat**.

Pour demander un service, le consommateur émet une **requête** vers le pourvoyeur. Celui-ci renvoie (en général) une **réponse**.

À chaque couple (requête, réponse) correspond une **opération** effectuée par le service : l'opération est exécutée par le service sur réception de la requête associée et est chargée de renvoyer la réponse. Le contrat spécifie donc la liste des opérations offertes par le service et, pour chaque opération la requête et la réponse.

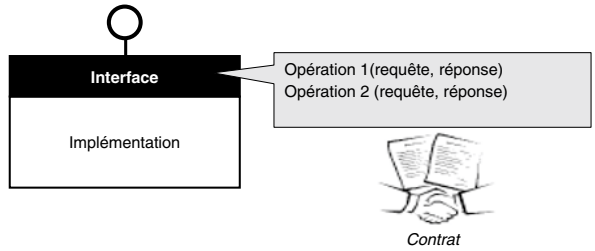



**Figure 5.1** — La notion de contrat

L'**interface** ne détaille pas comment les opérations sont implémentées : il masque au consommateur comment l'exécutant travaille, c'est-à-dire comment il est implémenté, et avec quelle technologie.

Le contrat peut-être formalisé dans n'importe quel langage respectant un standard, interne à l'entreprise ou partagé en respectant un standard basé sur XML (cf. WSDL décrit en partie 4).

Sans ce contrat, l'émetteur et le receveur de requêtes d'invocations devraient présupposer qu'ils utilisent, par exemple, le même langage informatique ou du moins des technologies compatibles par nature, et qu'ils sont dans des conditions préétablies de bonne communication pour échanger. Or, pour deux applications prises au hasard dans la cartographie d'un SI, et à plus forte raison entre SI, force est de reconnaître qu'au final ces deux postulats ne se vérifient spontanément qu'exceptionnellement.

**Remarque :** l'orientation Service peut donc s'appliquer à des fonctions liées à des concepts métiers (par exemple, calculer le montant d'une commande) aussi bien qu'à des concepts techniques (par exemple : enregistrer des traces).

### 5.1.2 Service et messages

Lorsqu'un consommateur souhaite utiliser (requérir) une opération d'un service, il lui transmet **un message** transportant sa requête.

> On notera dès à présent, que les requêtes des consommateurs et réponses des pourvoyeurs s'effectuent uniquement par messages[1], tandis que les invocations dans le SI s'effectuent selon une technologie propre à chaque îlot.

1. La notion de message entre service est approfondie au chapitre 5.

Le message est transporté sur le réseau via un **protocole de communication** formalisant et organisant les échanges sur le réseau. En retour, et lorsque cela sera nécessaire, le client recevra un autre message, qui sera également transporté via le même protocole ou un autre protocole de communication.

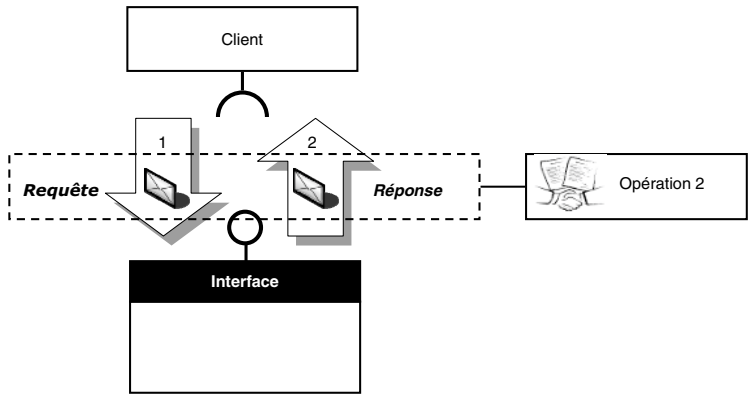



**Figure 5.2** — L'échange de messages entre consommateur et pourvoyeur

Cela traduit une volonté de **faible couplage** entre le pourvoyeur de service et le consommateur de service. Le pourvoyeur est, de ce fait, libre d'améliorer ou de changer à volonté l'implémentation du service pourvu qu'il respecte son **contrat** d'engagements initiaux avec chaque consommateur.

Le faible couplage permet donc au pourvoyeur, dès lors qu'il ne touche pas au contrat, de maîtriser :

- une plus grande diversité de consommateurs (tous les cas de dialogues étant explicitement décrits dans le contrat);
- la garantie de la qualité de son service (respect des conditions d'utilisations, suivi de l'utilisation, etc.);
- les modifications de l'implémentation de son service (par exemple pour suivre les réglementations, optimiser la performance, etc.).

Le faible couplage permet par ailleurs au consommateur :

- une facilité de développement (pas besoin de connaître l'implémentation);
- une fiabilité de transaction (le contrat est explicite, il n'y a donc pas de surprise);
- un changement possible de fournisseur (en cas de non satisfaction), si le fournisseur alternatif accepte de respecter le même contrat.

## 5.2 SPÉCIFIER UN CONTRAT DE SERVICE

### 5.2.1 Que contient un contrat « de base » ?

Ce contrat minimaliste est encore souvent appelé, par abus de langage, contrat d'interface. Il est constitué d'une liste d'une ou plusieurs opérations[1]. Chaque opération est décrite par les messages de requête et de réponse échangés avec les consommateurs du service : le triplet [nom de l'opération, message de requête, message de réponse] est appelé **signature** de l'opération[2].

Les messages doivent aussi transporter des informations métiers, dites sérialisées. Pour cela le contrat doit aussi préciser la **structure et le format** de chaque information.

Les opérations sont accessibles via un ou plusieurs **protocoles** de communication. Enfin, le service est localisé sur une **ressource** physique donnée, autrement dit, en général, un serveur.

Protocole et localisation du service sont dépendants d'une technologie réseau donnée. En revanche, opérations et signatures sont indépendantes de toute technologie.

Pour obtenir un service, de même que pour offrir un service, consommateur et pourvoyeur du service devront respecter ce contrat c'est-à-dire se conformer à un protocole de communication et échanger des messages entrants et/ou sortants définis par le contrat.

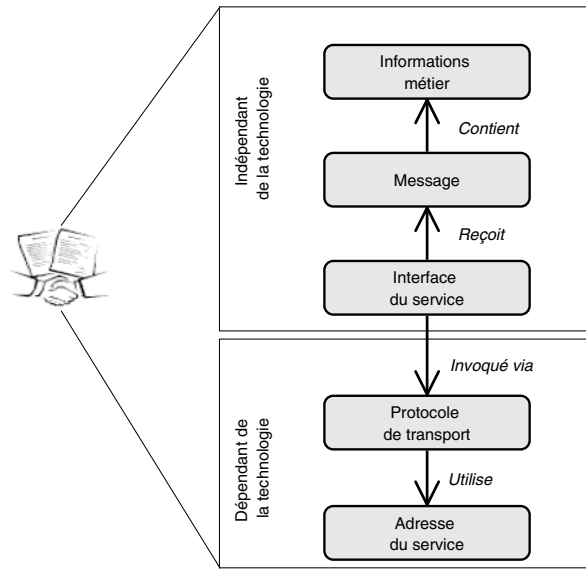



**Figure 5.3** — La description d'un contrat « de base »

1. À l'instar de méthodes disponibles sur un objet en programmation objet.
2. À l'instar d'une signature de fonction dans la programmation procédurale.

### 5.2.2 Contrat et qualité de service

#### *L'exigence de qualité*

La contractualisation du service doit traduire la volonté d'un accord entre consommateur et pourvoyeur non seulement sur les réponses apportées par le service, mais aussi sur la qualité du service. En général le consommateur souhaite non seulement que le service réponde à ses demandes, mais encore qu'il le fasse vite et sans erreur !

Le consommateur doit donc non seulement connaître l'offre de services (le contrat « de base »), mais aussi **les garanties** offertes par le pourvoyeur, garanties qui peuvent proposer un ou plusieurs **niveaux de qualité** selon le service disponible.

C'est l'intérêt du pourvoyeur d'offrir de telles garanties : en effet, face à une absence de garantie, tout consommateur sera réticent à utiliser un service. En effet, même si une première utilisation se passe bien, un consommateur n'a pas de garantie sur la répétition de ce bon fonctionnement. Un pourvoyeur offrant des garanties maximise l'utilisation de son service et ainsi le rentabilise. Le pourvoyeur doit donc communiquer ouvertement sur ses capacités et mettre en regard ces capacités avec les **exigences** du consommateur.

En retour, le pourvoyeur pourra exiger que le client respecte certaines contraintes et obligations d'utilisation. Ces contraintes sont autant de **directives** à respecter. Les directives expriment la nécessité de prendre en compte certaines réglementations et contraintes internes (mode d'accès) ou externes, et s'appliquent en général à une catégorie (ou groupe) de services.

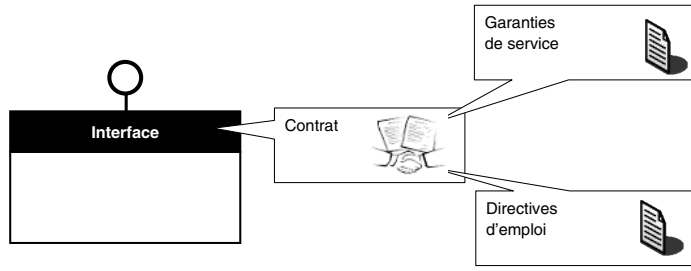



**Figure 5.4** — Le contrat « étendu »

Satisfaire les exigences (ou attentes) du consommateur implique **d'associer à chaque garantie** une **mesure** permettant de **vérifier** que les capacités attendues sont bien mises à disposition par le pourvoyeur. Réciproquement, satisfaire les directives imposées par le pourvoyeur implique d'associer à chaque directive une façon de vérifier le respect de cette directive.

Le contrat « étendu » par les garanties et directives définit ainsi le service rendu par le pourvoyeur, tant sur le plan métier, que sur le plan d'une qualité mesurable. Dès lors, le consommateur se référera à ce descriptif complet pour choisir et utiliser le service en pleine connaissance du résultat à en attendre.

### *Concept de SLA*

Ce contrat étendu peut aller jusqu'à couvrir les obligations d'un véritable **SLA**[1] résultant d'une **négociation** entre le consommateur et le pourvoyeur. Pour mettre en œuvre ce SLA, un contrat d'utilisation répertoriant les conditions particulières admises pour un consommateur donné devra en général venir compléter le contrat de qualité de service ainsi que le contrat de base, vus précédemment.

Ce contrat d'utilisation pourra définir par exemple :

- l'identification du consommateur ou d'une famille ou regroupement de consommateurs;
- le nombre de transactions consommables dans une plage de temps;
- les plages d'exclusivité éventuelles;
- les cas ou nombre d'erreurs acceptables;
- le respect de débit ou temps de réponses;
- etc.

Un accord de SLA est nécessaire dès lors que pourvoyeur et consommateur(s) du service n'appartiennent pas à la même entreprise voire à la même entité d'entreprise en cas de refacturation interne.

Cependant, la question se pose tout autant au sein de très grands comptes souhaitant travailler en mode de refacturation interne.

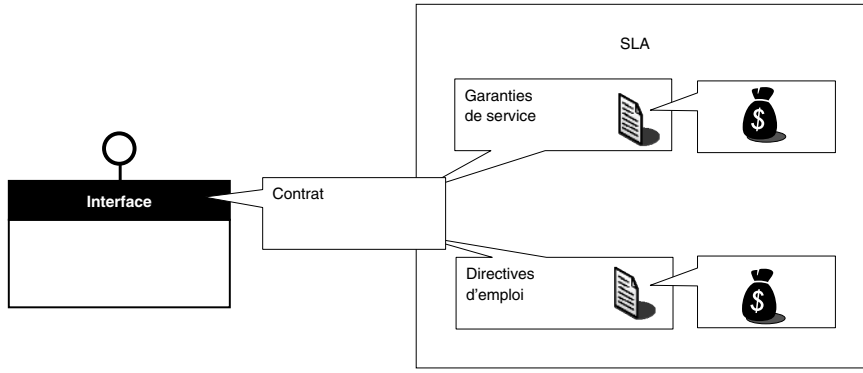



**Figure 5.5** — Le contrat avec SLA

La négociation du SLA portera sur :

- L'aspect « **métier** » :
  - la **Durée** de mise à disposition du service donc de la validité du SLA (à ne pas confondre avec la sous-durée d'accessibilité du service);

1. Issu de la terminologie anglo-saxonne « *Service Level Agreement* (SLA) », utilisée notamment par les opérateurs de réseau, les sociétés d'outsourcing, etc.

  - les garanties de performances (temps de réponses et nombre d'accès simultanés);
  - les garanties de disponibilité incluant le mode de calcul de sa mesure;
  - les garanties de sécurité : un client du service veut être sûr que le pourvoyeur limitera l'accès du service à certains profils de salariés de ce client.
- L'aspect « support technique » :
  - le **mode de surveillance** et d'alerte en cas de problème : ce mode pourra être périodique ou permanent, en précisant les points de mesures accessibles au consommateur;
  - le **mode de résolution** de problèmes lors de la remontée de défauts concernant la délivrance du service conformément au SLA ainsi que les temps de prise en compte, les éventuels outils ou infrastructure et moyens disponibles pour la gestion de tickets.
- L'aspect « **financier** » comprenant :
  - le coût d'un appel de service pour le consommateur en regard des ressources et garanties proposées;
  - le coût d'un défaut de qualité payé en compensation par le pourvoyeur du non-respect de l'accord (en particulier pour indisponibilité ou non-performance).

### *Comment s'associent contrat, SLA et service ?*

Les liens entre contrat, SLA et implémentation d'un service ne sont pas bijectifs, comme l'illustre la figure 5.6.

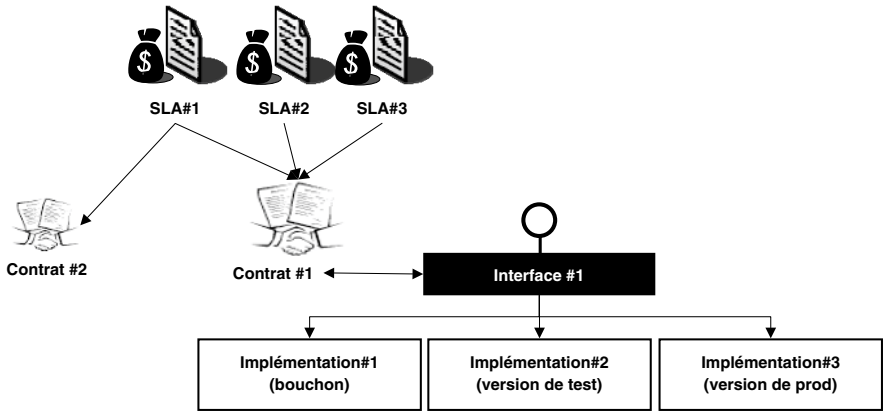



**Figure 5.6** — Les liens entre contrat, SLA et implémentation d'un service

Un même contrat de service peut être associé à plusieurs implémentations.

Un premier exemple typique concerne les services d'informations. Une première implémentation sera prévue pour un accès public et un niveau de SLA gratuit, et une seconde implémentation sera plus riche pour un accès, cette fois, authentifié et un niveau de SLA payant.

Un autre exemple est d'associer systématiquement à chaque service une implémentation « bouchon » permettant aux consommateurs de tester sans attendre l'implémentation complète de ce service. Ceci constitue d'ailleurs une bonne pratique SOA.

Un même contrat de service peut être associé à plusieurs SLA.

Par exemple, certains consommateurs de services bancaires sont prêts à payer plus cher pour obtenir des informations boursières en temps réel, alors que d'autres se contenteront d'une fourniture périodique, toutes les heures par exemple.

Autre exemple, un premier SLA pourra concerner des utilisateurs externes à l'entreprise, et un autre SLA concerner une filiale de cette entreprise. La différence pourra porter sur les conditions de facturation, les contraintes sécuritaires, etc.

Un SLA peut concerner simultanément plusieurs (contrats de) services.

Il s'agit dans ce cas d'éviter de multiplier les SLA, potentiellement longs à négocier.

Une même implémentation de service peut (parfois) servir à plusieurs contrats.

Peut-il enfin y avoir plusieurs contrats sur une même et unique implémentation de service ? Oui, si le pourvoyeur souhaite masquer certaines possibilités d'utilisation à certains consommateurs – mais c'est un cas de figure plus rare, car il vaut peut-être mieux proposer plusieurs opérations dans le même contrat, certaines opérations étant réservées à certains profils de consommateur.

### 5.2.3 Maîtriser la coordination de services

#### *Externaliser la séquence d'invocation de services*

La notion de composition introduite au précédent chapitre fait ressortir le concept de contrat externalisé étendu à un service souhaitant enchaîner plusieurs autres services. Ceci implique que des garanties ou SLA soit donc à nouveau définie pour chaque composition.

Au-delà de l'orchestration, la collaboration est un autre type de coordination automatisée principalement utilisée pour de la synchronisation entre plusieurs processus métiers. La collaboration permet à chaque processus de se limiter à connaître et seulement utiliser les documents ou messages entrants et sortants d'autres processus en interaction avec lui, sans avoir à connaître le déroulement précis (interne) de chacun d'entre eux.

Dès lors, la différence majeure entre les systèmes traditionnels de workflow et les moteurs de composition interactive ou automatisée est la capacité de décrire la séquence, non seulement séparément de l'implémentation, mais surtout séparément

du contrat de base afin de garder une flexibilité de recomposition et donc de maîtriser la coordination (cf. le chapitre 14 de la partie 4).

### *Vérifier la cohérence entre processus*

Au-delà de la composition relative à l'exécution de séquence de services, se pose aussi la question de la cohérence entre plusieurs séquences dès la phase de définition des services. En effet, dans certains cas mixant des processus implantés entre différents acteurs (usine, entrepôt, agent commercial…), une description des dialogues entre les processus, appelée **chorégraphie**, servira de « meta-modèle » et permettra de décrire la cohérence entre les principaux événements échangés entre processus.

La figure 5.7 résume les nuances de terminologie pour la coordination de services au travers de l'exemple d'une réservation de voyages impliquant la composition automatisée d'un service de réservation d'hôtel et d'un service de réservation de billet d'avion.

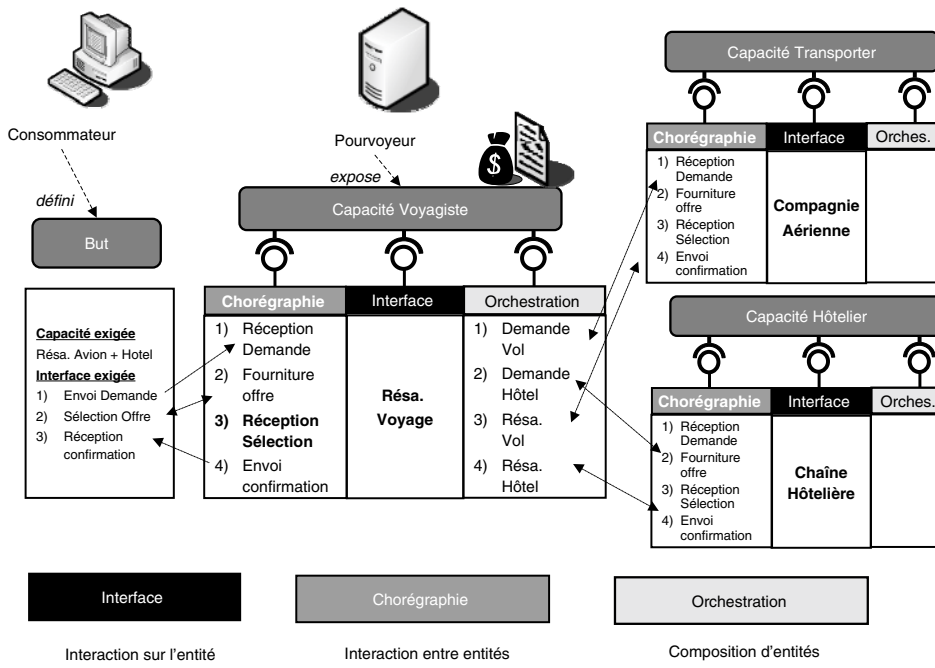



**Figure 5.7** — Notions liées à la coordination de services

## 5.3 METTRE EN PRODUCTION ET EXPLOITER LES SERVICES

Pour être complète, une description de services doit aussi comporter :

- d'une part, les éléments relatifs à la configuration et la gestion de variantes des services;

- d'autre part les éléments de packaging et de déploiement de ces packages de services (en tenant compte des dépendances de composition inter services);
- et enfin, les indicateurs relatifs à la supervision du service.

### 5.3.1 Le déploiement des services

S'il est utile du point de vue de la flexibilité de décrire explicitement le contrat d'un service, il est tout autant utile, pour sa mise en production, de décrire aussi explicitement les dépendances qu'il peut avoir vis-à-vis d'autres services.

À l'instar de spécifications existantes, mais fortement couplées à une technologie (comme celles des EJB ou des frameworks d'injection de dépendances comme Spring pour Java), l'utilisation d'une norme, indépendante de la technologie d'implémentation et d'exécution des services, permettra d'unifier ce descriptif, qu'il s'agisse de services déployés en technologies .NET, Java, Web Services, etc.

Ce besoin crucial pour la réussite de l'approche service a induit l'émergence de la norme SCA (cf. partie 4 – chapitre 14).

### 5.3.2 La gestion des variantes d'un service

Le concept de **variante** de service désigne l'intérêt voire l'obligation de pouvoir paramétrer lors de son déploiement le comportement d'un service.

Prenons le cas d'un service bancaire d'évaluation du risque associé à un portefeuille boursier : ce service est déployé dans chaque filiale, et doit donc raisonner à chaque fois dans une unité monétaire différente ($, €, ¥, etc.) tout en appliquant les mêmes règles prudentielles, c'est-à-dire en utilisant la même implémentation.

L'objectif d'une gestion de variante est de ne pas être contraint à implémenter autant de versions d'un service que de cas pouvant se présenter. Cette flexibilité se paie par :

- la complexité accrue des tests du service;
- la complexité d'implémentation du service, dans les cas les plus complexes;
- lorsque le nombre de configurations augmente sensiblement, un outil dédié se révélera indispensable pour gérer et centraliser le paramétrage des variantes.

Trois techniques principales sont exposées ici, et doivent être retenues suivant les contraintes de chaque projet.

#### *L'approche déclarative*

Cette technique s'applique pour changer la valeur d'un paramètre global.

Les paramètres globaux sont déclarés dans le contrat de service, plus exactement dans la partie déploiement de ce contrat, via l'usage d'une simple grammaire XML.

C'est l'approche adoptée par la norme SCA (cf. partie 4). Cette technique a quelques inconvénients :

- elle se limite à des variantes très simples : changement de valeur d'un paramètre de type simple (string, date, monnaie, etc.);
- elle se limite aux paramètres définis très tôt, dès la phase de modélisation; l'ajout dynamique de nouveaux paramètres n'est pas possible.

### *L'approche programmatique*

Cette technique s'applique lorsque le service doit utiliser des variantes de certaines règles métiers ou des variantes d'algorithme de calcul.

Elle présuppose le recours à un langage objet pour implémenter les services. On appliquera les patterns Objets appropriés[1], en particulier le pattern Stratégie (pour paramétrer dynamiquement le service par un comportement, ou stratégie). On évitera en revanche le recours trop systématique à l'héritage, qui peut favoriser des propagations automatiques de comportements non souhaitées lors de la démultiplication des variantes.

Cette technique a l'inconvénient d'introduire subrepticement la nécessité pour un consommateur de connaître les stratégies en question, c'est-à-dire de connaître (une partie de) l'implémentation du service, ce qui est contraire à SOA.

### *L'approche assertive – L'usage de langages d'assertions*

Cette technique peut s'appliquer pour contrôler de façon différenciée l'accès au service (ce consommateur a-t-il le droit d'utiliser le service ?) et/ou les messages d'invocation et de réponse du service (ce message est-il valide lorsque c'est ce consommateur qui l'envoie ?) et/ou le contexte d'invocation (le contexte est-il complet lors de l'invocation ? lors de la réponse ?) et/ou le respect du SLA (les conditions de sécurité sont-elles remplies ? etc.).

La technique s'appuie sur la définition de pré ou de post conditions déterminant soit le déclenchement effectif du service, soit l'envoi d'une réponse. Ces pré et post conditions sont exécutées par un outil approprié. La créativité dans ce domaine est intense, mais hélas ne contribue pas à faire émerger un candidat incontestable ! On citera néanmoins la Programmation Orienté Aspect comme candidat sérieux[2].

Sur le plan de la modélisation, OCL (*Object Constraints Language*), lié à UML et encore trop peu connu, pourra se révéler pertinent dans une optique MDA (cf. chapitre 11), c'est-à-dire couplé à un générateur de code vers l'outil cible.

---

1. Cf. la bible de référence : *Design Patterns – Elements of Reusable Object-Oriented Software*, Erich Gamma et alii, Addison-Wesley, 1995.
2. Le recours à des langages exotiques comme Eiffel paraît définitivement à proscrire. Le recours à des langages construits au dessus de Java ou C# (Jcontractor, Jass, JMSAssert, .NetContract, etc.) est plus réaliste, mais attendra la maturité de l'un de ces outils.

### 5.3.3 Le suivi des services

Concernant la supervision, un modèle classique consiste à instrumenter chaque service au moyen d'un agent plus ou moins intrusif (par exemple agent JMX ou WMI).

Cet agent est notamment en charge de dater chaque message entrant ou sortant, en retransmettant ces mesures auprès d'une console centrale.

Cette console doit fournir des moyens simples d'exprimer des filtres et des règles de corrélations entre les mesures établies afin de ne pas noyer l'exploitant sous une masse inutilisable de chiffres.

## 5.4 POUR UNE DESCRIPTION FORMELLE EXPLICITE DU CONTRAT DE SERVICE

Il apparaît clairement le besoin de spécifier progressivement chaque service, depuis sa phase d'identification jusqu'à sa phase de déploiement, en passant par son implémentation et sa mise au point. Le recours à une « fiche type » permet de regrouper l'ensemble des caractéristiques à décrire.

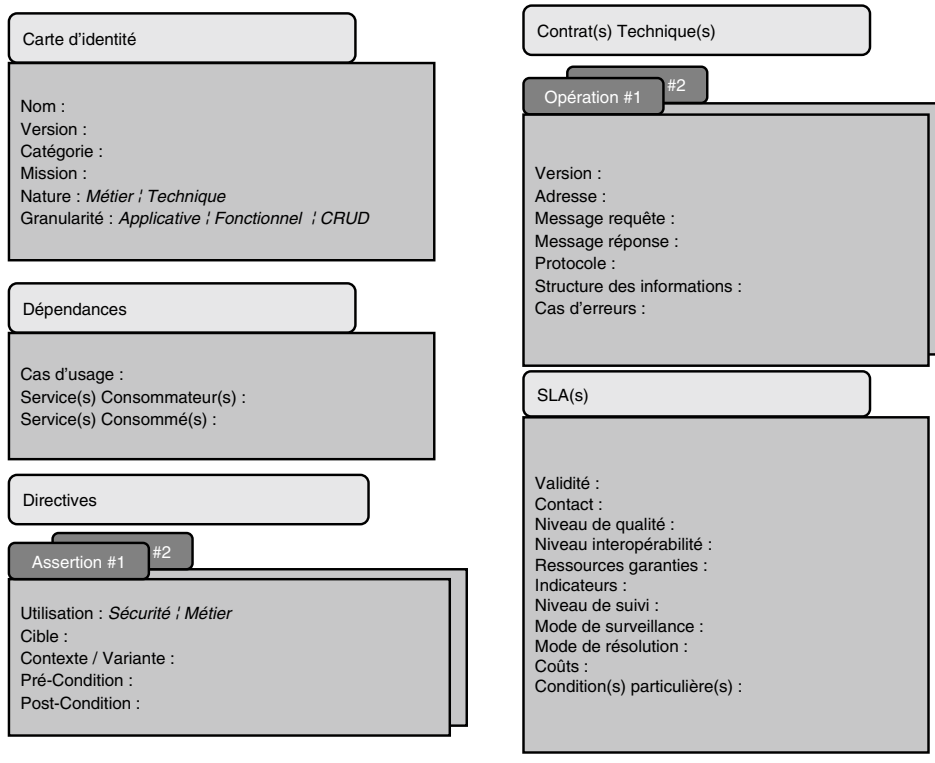



**Figure 5.8** — Exemple de fiche modèle spécifiant un service

Que doit contenir cette fiche descriptive ? Il s'agit des éléments techniques nécessaires aux différents acteurs du SI : urbanistes, consommateur (responsables et développeurs d'application), pourvoyeurs (responsable du service et responsables de services composant ce service), équipe d'administration du référentiel de services, équipe d'intégration.

Toutes ces informations sont autant de méta-données qu'il faut éviter d'oublier ou d'enfouir dans l'implémentation logicielle des services. Les normes et outils nécessaires ne couvrent pas encore vraiment l'intégralité de ces notions (cf. le standard WS-Policy décrit au chapitre 13 de la section 4), mais cela ne doit pas freiner le démarrage de la démarche SOA.

Cette fiche descriptive inclut notamment :

- La **carte d'identité** du service permettant la publication du service et sa recherche dans un référentiel : nom du service, mission (en une phrase), classification du service (métier/technique).
- **Le contrat de service** commenté.
- **Les directives techniques** (sécurisation, etc.).
- **Le ou les accords de SLA**.
- **La matrice de réutilisation** du service, c'est-à-dire l'ensemble des liens des applications et/ou services consommant ce service, et l'ensemble des services composés (donc consommés) par ce service.
- **Les paramètres de variantes de déploiement**.
- **L'historique des versions** du service.
- **D'éventuelles contraintes d'utilisation**, listées sous forme de pré-conditions et de post-conditions textuelles conditionnant la consommation du service – ce qui est contradictoire avec l'exigence d'autonomie du service, certes, mais apparaît comme utile dans certains cas.

À partir de l'ensemble de ces fiches, il est possible d'éditer différents guides de mise en œuvre :

- **Un guide d'utilisation des services à l'usage du consommateur**, c'est-à-dire une description « mode d'emploi » comprenant le contrat étendu voire l'accord de SLA lorsque cela est justifié.
- **Un guide de publication et de recherche**, c'est-à-dire un ensemble d'information permettant à un tiers de retrouver le service suivant certains critères de classification.
- **Un guide de sécurisation** décrivant les règles que le service adopte pour authentifier et identifier ses consommateurs, ainsi qu'éventuellement les règles de protection privée (chiffrement, etc.) ou de non répudiation de sa sollicitation.
- **Un guide de déploiement à l'usage des équipes d'intégration et de production** décrivant le ou les protocoles d'accès, l'implémentation à déployer des services assemblés par composition au sein de ce service, etc.

- **Un guide de supervision métier, destiné aux maîtrises d'ouvrage**, c'est-à-dire la présentation des indicateurs permettant de suivre les SLA.
- **Un guide de supervision technique**, destiné aux équipes d'exploitant, c'est-à-dire la présentation des indicateurs permettant de suivre la performance technique du service.

### 5.4.1 Bilan des contraintes et des bénéfices

Un contrat de service satisfait les critères suivants :

- **Formalisé :** le contrat est défini dans un langage formalisé, WSDL par exemple.
- **Publié :** le contrat est publié, accessible et compréhensible par les consommateurs.
- **Concentré** sur une mission fonctionnelle et une seule : le service est clairement positionné dans la taxonomie des services (métier versus technique, gros grain versus grain fin).
- **Étanche :** le contrat ne fait pas référence à l'implémentation du service. Il est en ce sens une boîte noire.
- **Explicite** sur la Qualité de Service.

Le service lui-même (plus exactement, son implémentation) satisfait les critères suivants :

- **Autonome :** il faut favoriser le couplage faible entre services.
- **Isolé** des aspects techniques client/serveur : le pourvoyeur sépare la logique métier de la logique de communication client/service).
- **Flexible :** le paramétrage de variante doit être possible sans intervention sur le code, et être aussi tardif que possible, au moment du déploiement si possible.
- **Sans état :** le service ne garde pas de trace de ses invocations successives, c'est au consommateur de s'en préoccuper.
- **Transparent :** le service fournit des informations sur son exécution, ses performances, etc.

Les bénéfices sont néanmoins nombreux :

- L'usage de contrats explicites assure **l'indépendance d'évolution** entre consommateur et pourvoyeur.
- L'usage de contrats stables et publics, lié à une démarche d'homologation systématique, favorise la **réutilisation** des services.
- L'expression de niveaux de qualité de service contribue à la mise en place progressive d'un **suivi proactif**.
- Les techniques de compositions interactives et orchestrées, permettent d'envisager un **découplage de la logique de coordination** métier et des services déjà disponibles.

- Le recours à des techniques de **prise en compte des variantes** contribue à fiabiliser l'usage d'un service malgré des contextes divergents de consommation et facilite son déploiement.

## En résumé

Le **contrat** de service réalisé par le pourvoyeur exprime un mode d'invocation au travers de messages. Il doit être étendu par des garanties, selon des directives au profit d'un ou plusieurs consommateurs, et **réutilisable par composition** pour un pourvoyeur de service de plus haut niveau.

La richesse de ces concepts liés au service implique un effort certain lors de la spécification des services et des **SLA**, les normes et outils n'étant pas encore complets. C'est pourquoi la mise en place d'une fiche type de description de service s'impose.

L'expression de niveaux de qualité de services différents, contribue à la mise en place progressive d'un suivi proactif, autorisant un découplage jusqu'à l'exploitation.

Plusieurs bénéfices sont à retirer de la formalisation du service.

# 6

# L'émergence d'une plate-forme SOA

## Objectif

L'échange de messages entre consommateurs et pourvoyeurs de service implique la mise en place d'un middleware adapté : le bus de messages. Le premier objectif du chapitre est de présenter le rôle et la place d'un tel bus dans le cadre de SOA.

Mais le bus à lui seul ne suffit pas. Déploiement des services, orchestration de processus, capacité d'accéder aux systèmes existants, suivi de la qualité de service, etc. sont autant de besoins identifiés à prendre en compte avec SOA. Il conviendra d'y répondre via les outils adéquats.

Le deuxième objectif du chapitre est de présenter de façon synthétique l'ensemble de ces outils et de mettre ainsi en évidence l'émergence d'une véritable plate-forme cohérente d'infrastructure SOA.

Enfin, la nécessité d'industrialiser le développement des services dans un cadre productif et selon une méthodologie adaptée, conduit au concept de chaîne de fabrication des services, dont la présentation constitue le troisième objectif du chapitre.

Cette présentation, synthétique, sera approfondie aux chapitres 20 et 21 de la partie 7, tandis que le chapitre 22 fournira un rapide tour d'horizon de l'offre.

## 6.1 L'ÉMERGENCE DU CONCEPT DE BUS DE MESSAGES

Le consommateur d'un service invoque ce service en lui envoyant un message de requête, et en attendant éventuellement un message de réponse. L'ensemble des messages échangés entre consommateur et pourvoyeur de service transite via un outil que l'on qualifiera pour l'instant du terme relativement vague de **bus de messages** SOA.

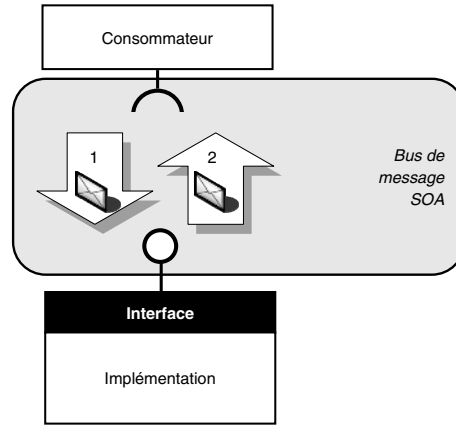



**Figure 6.1** — Le bus de messages SOA

### 6.1.1 Bus de messages et contrats formalisés

Les contrats de service doivent être formalisés : ceci permet aux consommateurs de déléguer au bus de message le soin de vérifier que les messages échangés sont bien conformes aux contrats concernés. Les EAI apportaient initialement leur formalisme propriétaire. Il conviendra donc de recourir à un vocabulaire plus universel (d'où XML) et si possible via l'usage de standards WS-* (cf. chapitre 12)[1].

Pour opérer ces vérifications, ce bus doit accéder à un référentiel enregistrant les contrats de service WSDL, ou registre des services.

### 6.1.2 Bus de Messages et protocoles sous-jacents

L'approche SOA souhaite isoler les consommateurs des technologies utilisées par les fournisseurs pour implémenter les services, mais elle souhaite également isoler pourvoyeurs et consommateurs des protocoles de communication utilisables pour échanger les messages.

1. Il serait d'ailleurs plus conforme à l'histoire, de dire que WSDL (et les autres standards de base, comme SOAP – voir plus loin) sont apparus en premier, et que les difficultés de mise en œuvre de ces standards ont, dans un second temps, suscitées les réflexions menant à la cristallisation de l'approche architecturale SOA.

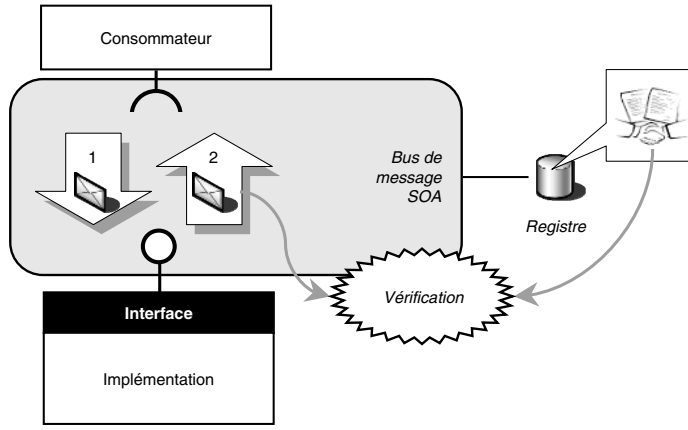



**Figure 6.2** — Le bus de message SOA et le registre de contrats

Si l'on prend une analogie du monde réel, cela peut se comparer à un service postal offrant d'acheminer un courrier reçu sous forme électronique, soit comme un courrier « papier » en recommandé, soit comme un fax, soit comme un courrier électronique sécurisé.

Or pour qu'un message soit transportable sur une grande variété de protocoles, qu'ils soient basés sur un mode texte (comme HTTP) ou binaire (comme MQ, JMS ou CICS), il doit être transporté dans une enveloppe normalisée. **L'enveloppe** précise dans un **en-tête** l'adresse du service, une demande d'accusé de réception, l'encodage ou la compression du contenu transporté, des informations de sécurité, etc. L'indépendance vis-à-vis des protocoles de transport conduit à privilégier l'usage naturel d'une structuration de l'enveloppe et du contenu en XML. Le W3C recommande depuis 2003 SOAP v1.2 comme protocole de messagerie standardisé pour les services Web (cf. La section 4, détaillant notamment les avantages de SOAP par rapport à REST).

En conclusion, un **bus de messages** fournit un mécanisme de communication commun aux consommateurs et pourvoyeurs en imposant :

- **Une langue commune :** la grammaire ou schéma des contrats de base, éventuellement retraduite vers WSDL.
- **Des directives communes** (les ordres de messages).
- **Une infrastructure de transport de haut niveau :** Il s'agit de masquer la variété des protocoles de transport effectivement utilisés en les retraduisant vers un message standardisé extensible comme SOAP – autrement dit, et pour faire bref, il est possible (i.e. la norme n'interdit pas) de transporter des messages SOAP sur des protocoles aussi divers que HTTP, FTP, SMTP, JMS, etc.

**Ceci renforce le principe SOA ≠ Web Services.** Comme on l'a vu, l'approche Service ne peut se réduire à la seule mise en œuvre des protocoles Web Services. SOA est d'abord un modèle de conception d'architecture, les WS-* ne sont qu'un moyen de

déployer les services sur une infrastructure Web, et sont à mettre en balance avec d'autres techniques pouvant dans des contextes moins ambitieux d'interopérabilité se révéler plus optimum (cf. Corba, EJB Session, .NET Assembly, etc.).

Ainsi, il sera fondamental de veiller à ce que le bus dispose de moyens d'optimiser les éventuelles étapes de retraductions internes et n'oblige pas, par exemple, lorsque consommateur et pourvoyeur sont dans des technologies compatibles, à un surcoût de performance.

### 6.1.3 Les modes d'échange de messages

Il est fondamental de comprendre qu'un service ne nécessite pas obligatoirement une requête et une réponse. Le modèle d'interaction (entrant seul, sortant seul, entrant/sortant etc.) devra faire l'objet d'une formalisation dans le contrat de base.

Ainsi par exemple, le consommateur s'abonnant à un service de presse, souhaite le faire grâce à une seule requête, lui permettant de disposer en retour de nombreuses réponses (autant que le réclameront les choix aux critères retenus dans son abonnement ou sa sélection). Un service peut donc se contenter d'une simple invocation entrante, de même qu'il peut imposer une réponse pour chaque requête entrante, ou encore s'obliger à renvoyer une réponse ou une erreur (pour signifier sa non réponse).

La variété des interactions bilatérales ou multilatérales entre services est donc à prendre en compte par le bus de messages (cf. les caractéristiques décrites au chapitre 18).

### 6.1.4 Et la sécurité ?

Une architecture SOA doit répondre aux problématiques de sécurité présentée dans la partie 1. Elle doit répondre en particulier aux questions suivantes :

- Comment s'assurer de l'identité du fournisseur, du pourvoyeur ou du consommateur du Service ?
- Comment définir et exposer les droits d'accès à un Service ?
- Comment assurer la confidentialité des échanges ?
- Comment assurer la conservation des messages lors d'un échange sensible mettant en jeu plusieurs partenaires ?
- Etc.

Dans le cadre d'une architecture SOA, la sécurité peut être abordée de manière traditionnelle :

- En mettant en place une sécurité par le réseau (Firewall, DMZ, VPN, etc.).
- En utilisant une sécurité au niveau protocolaire (SSL, IPSEC, etc.).
- En déployant des systèmes d'authentification (LDAP, certificats, etc.).
- Etc.

Cependant, la sécurisation des messages échangés suivant ces méthodes peut se relever complexe lorsque de nombreux tiers entrent en jeu dans cet échange. Ainsi, il peut être plus intéressant de reporter les opérations de chiffrement menées au niveau protocolaire sur les corps des messages eux-mêmes.

Ce principe est celui de l'échange des e-mails sécurisés avec S/MIME : ce sont les corps des e-mails qui sont chiffrés et non les protocoles SMTP et POP. Ces aspects sont détaillés dans la partie 4 au chapitre 13.

## 6.2 DU BUS DE MESSAGES À LA PLATE-FORME SOA

Au fur et à mesure de la progression du déploiement des services et de la volonté d'une industrialisation à l'échelle globale de l'entreprise, se justifie le besoin d'une infrastructure capable de couvrir l'ensemble des besoins identifiés dans cette partie.

La figure 6.3 présente les différents ingrédients de ce qui constitue une véritable plate-forme de mise en place des Applications et des Services.

Cette plate-forme regroupe :

- Le **bus de service**[1] pour unifier les communications entre services.

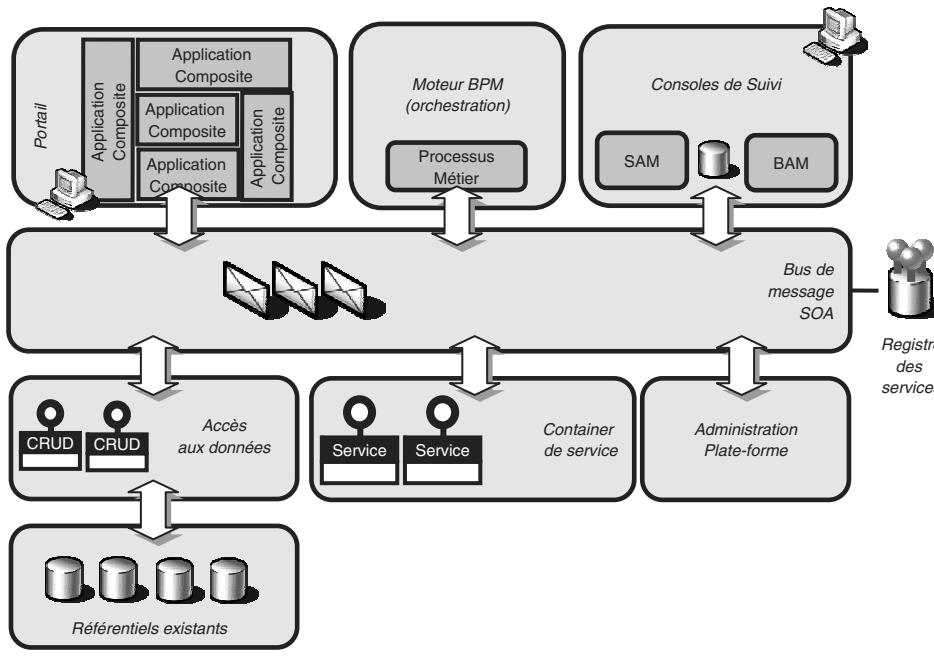



**Figure 6.3** — Les composants d'une plate-forme SOA

1. La terminologie anglo-saxonne utilise le terme « *Enterprise Service Bus* (ESB) ».

- L'annuaire de service (ou **registre**[1]) pour faciliter la localisation des contrats de services.
- L'**orchestrateur**[2] pour exécuter les processus automatisés.
- Le **requêteur d'accès aux référentiels existants**[3] pour unifier l'accès aux données.
- Le **moniteur de services et de processus** [4] pour utiliser en cohérence des directives d'exploitation et unifier la remontée pertinente d'indicateurs par profil.

### 6.2.1 Le contrôle et la supervision des services

Les aspects qualité de service ont été précédemment largement évoqués. Il est donc nécessaire de superviser cette qualité. On parlera de SAM, *Service Activity Management*.

La supervision d'un service consiste à le surveiller et à agir sur ses paramètres opérationnels pour qu'il satisfasse les demandes des consommateurs et les contraintes des pourvoyeurs. De manière générale, elle couvre différentes fonctionnalités parmi lesquelles le suivi des erreurs, de la performance, de la qualité et de la sécurité.

Cette supervision SAM repose sur l'utilisation d'indicateurs visualisant et synthétisant des mesures prises sur des services unitaires. La difficulté est de fournir une console unifiée permettant d'offrir différentes vues en détail ou en synthèse adaptées

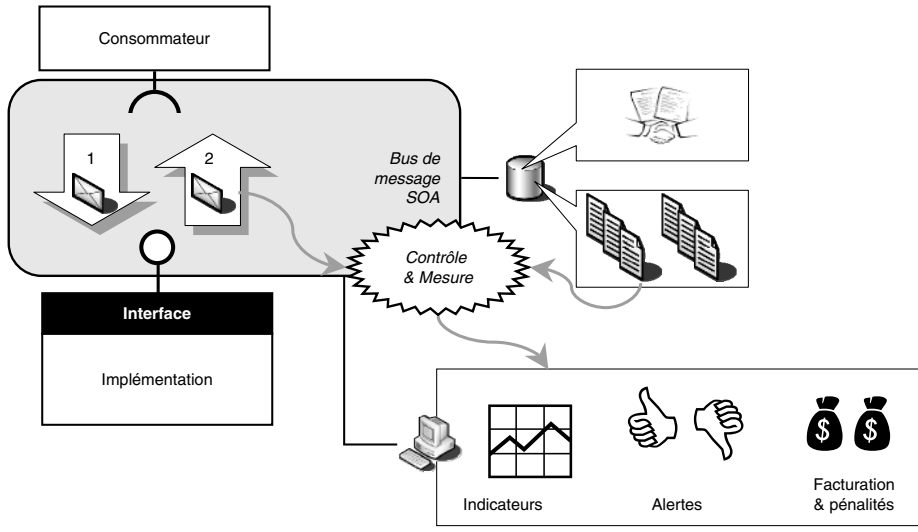



**Figure 6.4** — La supervision de la qualité de service, ou SAM

1. La terminologie anglo-saxonne utilise le terme « *service registry* ».
2. La terminologie anglo-saxonne utilise le terme « *Business Process Management* (BPM) ».
3. La terminologie anglo-saxonne utilise le terme « *Enterprise Information Integration* (EII) ».
4. La terminologie anglo-saxonne utilise le terme « *Business Activity Monitoring* (BAM) » et on introduit par analogie le SAM, « *Service Activity Monitoring* ».

aux besoins opérationnels de l'informatique interne, mais aussi aux besoins des métiers internes voire des consommateurs finaux.

À noter que la console de suivi SAM peut aussi être unifiée avec une console de production déjà en place dans l'entreprise (par exemple Tivoli, HPOpenView, Patrol ou Microsoft). Ceci étant d'autant plus nécessaire si les indicateurs à intégrer s'éparpillent depuis le niveau réseau, jusqu'aux invocations de messages (voire à la disponibilité des personnes humaines éventuellement impliquées).

La détermination des « bons » indicateurs métiers et techniques est un processus par nature adaptatif et itératif. Par conséquent, il est vital de pouvoir positionner ces indicateurs et règles d'une façon la moins intrusive possible dans l'implémentation, d'où le recours à des descriptifs explicites plus ou moins extensibles utilisés en pré/post compilation ou mieux en réaction à des événements.

Différents niveaux de suivi sont bien entendu envisageables :

- Un premier niveau se contente de visualiser les indicateurs d'accessibilité, de performance et de disponibilité plus ou moins individualisés sur chaque opération de chaque service.
- Un second niveau permet d'automatiser le suivi du niveau de qualité de service négocié, via la définition de seuils et d'alerte.
- Un troisième niveau cherche à établir un diagnostic de dysfonctionnement voire à corriger en temps réel le dysfonctionnement.

L'expression de directives permet par exemple d'exprimer des règles pour limiter le nombre de demandes au-delà d'un certain seuil de saturation des ressources allouées à l'opération d'un service. Des règles permettront au contraire d'allouer ou retirer des ressources au fur et à mesure des demandes sur un service. D'autres règles peuvent aussi faciliter l'établissement d'un niveau de priorité de prise en compte d'un message entrant afin de réguler la charge d'une infrastructure d'exécution de services.

Un point délicat concerne le lien entre supervision SAM et respect des SLA en ce qui concerne l'impact financier du non respect de ces SLA. Le lien entre SAM et système de facturation reste encore du domaine du futur.

### *La planification des capacités de l'infrastructure*

Une vision futuriste, prônée par les cabinets de veille (Gartner, etc.) et les grands éditeurs, met en avant la possibilité non seulement de détecter mais encore d'anticiper sur l'évolution des performances. On parle alors d'anticipation de la demande[1].

Cette vision est en fait le point de jonction entre deux tendances lourdes de l'informatique : l'approche SOA d'une part, et le concept de « grille de calcul virtuelle » d'autre part.

Le concept de grille désigne simplement la possibilité de considérer l'ensemble des composants de l'infrastructure (PC, CPU d'un serveur multi-processeurs, bande

1. La terminologie anglo-saxonne parle de *Capacity Planning*.

passante du réseau, etc.) comme des ressources banalisées auxquelles on peut affecter dynamiquement des tâches de traitements machine. Dans ce cadre, la possibilité d'anticipation de la demande désigne la capacité d'exploiter les indicateurs mesurés pour proposer l'allocation au Bus de Message de ressources existantes (bande passante par exemple), voire la mise en place de nouvelles ressources.

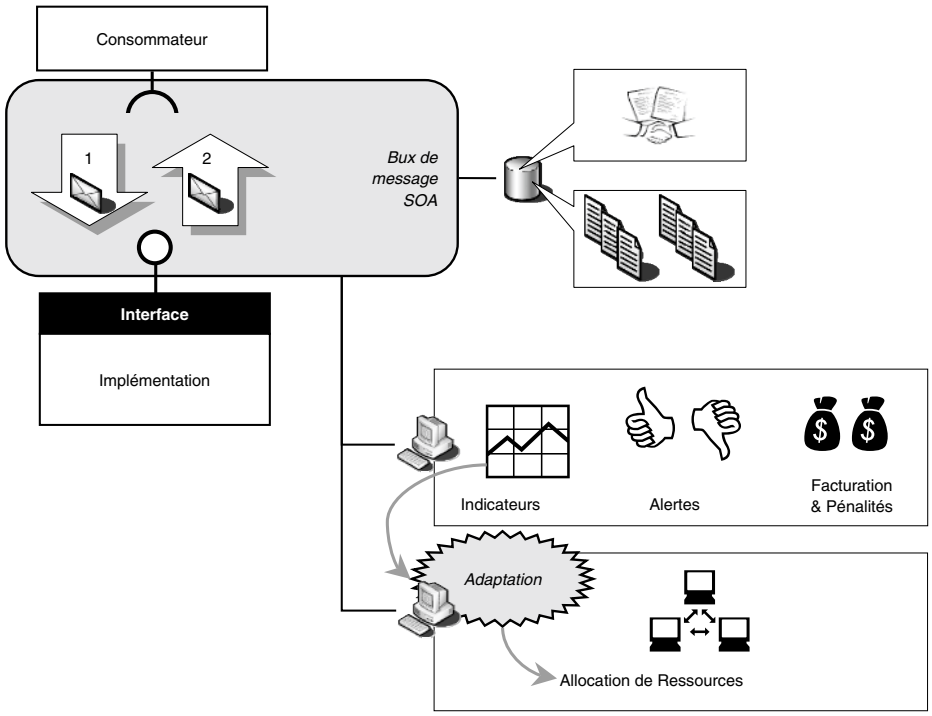



**Figure 6.5** — Supervision et capacity planning

## 6.2.2 Rôle d'un registre de service

Dès que le nombre de services homologués devient important, le dialogue entre consommateurs (les développeurs d'applications composites et de services composés) et pourvoyeurs de services devient de plus en plus difficile en point à point.

> Dans une multitude de services, comment un éventuel consommateur sait-il seulement que le service dont il a besoin existe déjà ? Répondre à cette question, conduit au concept de registre de services, ou annuaire.

L'introduction d'une infrastructure servant de **registre** d'intermédiation entre pourvoyeurs et consommateurs se révèle donc de plus en plus utile. Ce registre offre les cas d'usage suivants :

- le pourvoyeur **publie** la description de son service dans le registre (fiche d'identité, contrat, SLA, etc.). La fiche d'identité contient la catégorisation du service (c'est-à-dire les mots clés qui en faciliteront la recherche);

- le consommateur recherche un service correspondant à son niveau d'exigences auprès du registre ;
- le registre transmet au consommateur la ou les descriptions correspondant à sa recherche.

Le registre peut aussi jouer un rôle de zone de **publication des implémentations** de services « validés » et faciliter ainsi les déploiements sur plusieurs environnements (pour le développement, le test, la pré-production etc.).

**Le registre peut-il s'utiliser dans un cadre plus large que celui du SI d'une seule entreprise ?** Il reste difficile à ce jour d'utiliser dynamiquement des services auparavant « inconnus », notamment sur le plan commercial et juridique. Mais du fait que l'entreprise peut par exemple fédérer un réseau de partenaires commerciaux, il peut être nécessaire d'ouvrir le registre à ces partenaires.

Sans verser dans l'ambition des annuaires planétaires, il n'en demeure pas moins intéressant d'ouvrir son registre de service au sein d'un périmètre délimité, ou bien, mettre en réseau son registre avec ceux de partenaires agréés. (cf. les topologies de registre décrites dans le chapitre 20 de la partie 7).

### 6.2.3 L'accès aux référentiels (services CRUD)

Il s'agit d'un distributeur de requêtes et d'informations. Une nouvelle génération d'ETL[1] orienté service (baptisée solutions EII/ECI[2] suivant les éditeurs) émerge pour faciliter l'agrégat voire la synchronisation temps réel de services d'informations ou de contenus distribués sur plusieurs sources dans le back-end du SI. Ceci permet de répondre à des besoins de haute disponibilité, en lecture (voire en écriture), sur les données des référentiels de l'entreprise. Ces solutions permettent aussi la fabrication rapide de la mise en correspondance entre des vues logiques et des données (à l'instar des modèles conceptuels de données) en les exposant comme autant de services. Ainsi il devient possible de centraliser en un seul point les règles de SLA, de suivi (ou gouvernance) et de mise en cache de données réparties. Cet intermédiaire apporte ainsi l'accès à plusieurs sources via une seule requête et dispose de technologies complémentaires aux ETL, EAI ou BPM[3] tels que la gestion de cache, de la sécurité, de l'indexation ou de l'optimisation de requêtes distribuées.

Reste alors à gérer l'externalisation et la conservation de tous les descripteurs d'information ainsi que les contextes de services. Souvent baptisés outils de MDM[4]

1. *Extraction – Transformation – Load* : outil d'extraction, de transformation et de rechargement de données.
2. *Enterprise Information/Content Integration*.
3. *Business Process Mangement* : outil d'automatisation des processus.
4. *Master Data Management*.

par la plupart des éditeurs, ils facilitent pour chaque branche de l'entreprise le partage de la même définition d'entités, par exemple le client ou la facture[1].

### 6.2.4 Gestion et supervision des processus métiers (BPM et BAM)[2]

Une infrastructure de gestion des processus métiers SOA (encore appelé *Business Process Management* – BPM) englobe plusieurs composants :

- La modélisation amont des processus métiers. Celle-ci comporte, en général, une description des d'activités d'un processus (sa séquence interne), dont certaines pourront se réduire à invoquer un service ou une composition de services implémentée de façon programmatique.
- L'exécution des processus métiers. Il s'agit de l'orchestration de processus décrite précédemment, traitement des événements métiers, sollicitation d'un utilisateur humain ou d'un service (automatique) pouvant recouvrir lui-même une orchestration de services, une intégration à un ou plusieurs systèmes back-end, etc.
- Le suivi et l'exploitation aval des processus métiers (pilotage de la performance opérationnelle ou technologique dont le suivi de l'exécution des services).

Les outils de BAM se basent sur les fonctionnalités de SAM pour :

- La consultation de l'état des processus.
- La consultation ou l'édition de statistiques à la volée : par exemple, combien de processus de tel type sur telle période, combien d'exceptions traitées actuellement par telle personne ? Ceci doit pouvoir se personnaliser à l'aide de tris et de filtres, que doit pouvoir effectuer un responsable opérationnel à l'aide de son propre poste de travail (cf. le chapitre 9 de la partie 3).
- La gestion d'alertes : Par exemple, un processus bloqué depuis plus d'un certain temps doit-il déclencher un processus alternatif ou une action humaine.

Il peut y avoir par ailleurs un aspect plus décisionnel, comme l'établissement de statistiques agrégées sur plusieurs axes, construites en temps différés ou nécessitant un générateur d'état.

Un outil de BAM peut s'envisager directement câblé au système d'information, mais pourra tirer toute sa valeur ajoutée qu'une fois intégré au système de gestion des processus métiers en y capturant et en reconsolidant les événements métiers. Cela implique en général le recours à un moteur de règles (BRM[3]) ainsi qu'un serveur de traitement des événements (CEP[4]). Le moteur BRM permet de définir des règles métiers, au travers d'une console d'assistance, effectuant des filtrages ou fonctions de corrélations sur une base d'événements et messages émis ou reçus capturés par le

1. Certaines solutions globales se proclament alors EIMS (*Enterprise Information Management System*). Elle propose d'aller du connecteur SQL orienté service jusqu'au MDM avec gestion de variantes des définitions et une réconciliation des sémantiques sans oublier un générateur unifié de requêtes vers des sources multiples.
2. *Business Activity Monitoring.*
3. BRM (*Business Rules Management*).
4. CEP (*Complex Event Processing*).

serveur CEP. Ceci permet en outre d'exploiter automatiquement des tendances. Par exemple, afin de déterminer si une prise de commandes suit un cycle particulier d'occurrence et au-delà d'exploiter à nouveau cette information soit en déclenchant un nouveau processus métiers soit en routant l'événement vers un opérateur humain. Le système CEP permet, durant l'exécution des processus métiers, de confirmer l'applicabilité des règles ainsi définies, à échelle de certains seuils. Il devra alors émettre d'autres événements synthétisant à plus haut niveau les événements unitaires ainsi corrélés. La clef réside donc dans le traitement en temps réel de flux d'événements pouvant atteindre plusieurs centaines de milliers par seconde dans le domaine boursier par exemple.

Ainsi, il devient possible d'assurer un cycle complet, durable et d'amélioration continue, depuis la modélisation, l'exécution, le suivi et jusqu'à l'optimisation des processus.

## 6.3 LA CHAÎNE DE FABRICATION SOA

Une architecture SOA devra s'appuyer sur le triptyque suivant :

- des applications composites et des services;
- une infrastructure d'échange;

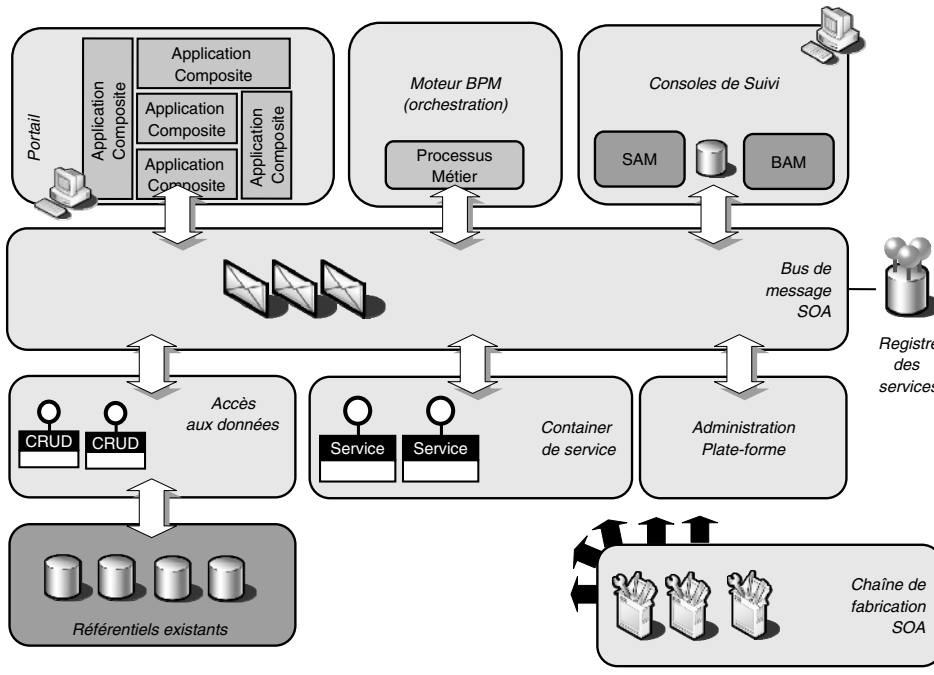



**Figure 6.6** — La chaîne de fabrication complète de la plate-forme SOA

- mais aussi des ateliers de modélisation, développement, paramétrage et déploiement des applications, des services et de l'infrastructure.

Du modèle d'interface de services aux modèles de processus, du développement au déploiement, ces ateliers, doivent constituer une véritable chaîne de fabrication SOA[1], pour permettre d'appliquer de façon productive une logique méthodologique (cf. chapitre 11 de la partie 3).

### 6.3.1 Industrialiser pour mieux livrer

À l'instar du mode de fabrication rencontré dans l'industrie, les outils nécessaires aux développeurs, testeurs, responsable d'interface homme machine, responsable métiers, responsable qualité, architectes, etc., se fédèrent progressivement autour d'une chaîne plus ou moins automatisée de fabrication des services.

Les services ainsi réalisés en bout de chaîne suivent un certain nombre d'étapes systématiques (voire automatiques), ils sont testés, fiabilisés et répondent aux exigences suivant la méthodologie retenue par l'entreprise pourvoyeuse de services. En outre, il pourra s'avérer plus ou moins nécessaire, selon le volume de services, ainsi que la diversité et l'activité des équipes, de fédérer cette chaîne autour d'un référentiel homogène. Ce référentiel servira dès la conception des services (on parle alors de catalogue de services lorsqu'il s'agit d'éléments principalement descriptifs et informationnels ou d'annuaire lorsqu'il s'agit d'éléments reliés au code source et/ou à la génération de code). Le référentiel peut encore s'étendre jusqu'au déploiement et à l'exécution des services (on parle alors de « registre de services »).

Il permet ainsi de gérer en cohérence plusieurs vues et profils d'intervenants autour du cycle de vie de chaque service (par exemple, la vue de l'analyste et des cas et impacts d'utilisation, la vue du concepteur, du testeur, et de l'architecte, la vue de conformité aux réglementations, la vue des statistiques d'utilisation pour le pilotage etc.).

On notera que si ce type de référentiel peut initialement s'amorcer aisément à l'aide d'un simple serveur de fichiers ou d'un wiki collaboratif, il devra progressivement évoluer en proposant en amont une structuration des données gérées, ainsi qu'en aval des tableaux synthétiques et cohérents pour chaque profil agrégeant une ou plusieurs sources d'informations parfois distribuées selon le patrimoine ou la compléxité de l'implémentation retenue par l'entreprise.

Cette chaîne doit donc se concevoir, se fiabiliser et se pérenniser au fur et à mesure des projets du programme SOA de l'entreprise (cf. le chapitre 11 de la partie 3). Autrement dit, il ne faut pas nécessairement chercher la chaîne « idéale » dès le premier projet.

1. La terminologie anglo-saxonne est celle de « *software factory* ».

### 6.3.2 Modéliser pour mieux réutiliser

L'approche SOA peut parfois donner tendance à croire que le mot « développement » est condamné à disparaître. Force est de reconnaître qu'il faut quand même encore développer, ne serait-ce que des façades d'appels à des implémentations existantes.

À ce titre, il est important de constater que si la fabrication de services sur de nouvelles implémentations peut rester assez naturelle et directe, celle de services construits sur des implémentations existantes pourra, dans certains cas d'imbrications, nécessiter un travail important de réécriture voire de migration de ces applications « trop monolithiques ». Il n'y a donc pas de miracle ! De toute façon, un développeur sera au moins présent pour les services métiers « pivots » (même s'ils encapsulent l'accès aux référentiels ou à des moteurs de règles) ainsi que pour les nouvelles applications composites « orientées services ».

Sur le plan de la modélisation, SOA est un aboutissement des démarches entreprises pour concevoir des architectures objets. Architecte et Responsable de développement peuvent donc envisager pour certaines Applications Orientées Services une assistance à la génération du code depuis une modélisation préalable : c'est le rapprochement souhaitable entre l'approche MDA[1] et l'approche SOA, décrite au chapitre 11 de la partie 3.

### 6.3.3 Paramétrer la supervision

Les outils de SAM et BAM sont associés à un atelier de paramétrage. Celui-ci doit en outre permettre de :

- décrire les indicateurs clefs de la performance métier, notamment dans le séquencement des processus métiers, ainsi que dans le suivi des directives à contrôler et leurs paramètres possibles pour chaque service métier, (intensité de sollicitation, temps entre activités, statuts du processus, etc.);
- d'établir un mode de relevé (par exemple via agent ou modèle de collecte intermédiaire);
- de choisir un mode des rendus :
  - via des représentations statiques proposant des tableaux ou rapports exportables en tableur, PDF ou données brutes, etc.;
  - via des représentations interactives dédiées ou unifiées au sein d'une interface homme-machine de type Portail, Frontal décisionnel ou console permettant plus facilement de descendre plus ou moins en profondeur dans la supervision;
- de choisir le mode d'alerte (par messages de type e-mail, SMS, boîte de dialogue sur l'écran d'un opérateur, nouveau processus ou services à déclencher, etc.). Le

1. Approche *Model Driven Architecture* (MDA).

paramétrage peut aussi porter sur les seuils et règles de mise en escalade ainsi que sur le niveau de détail à restituer (fréquence de rafraîchissement, verbosité des traces, etc.).

## En résumé

Bien que SOA soit avant tout une approche architecturale, il ne saurait être question de négliger l'infrastructure requise pour le déploiement, l'exécution et la supervision des Applications, Processus et Services.

On constate l'arrivée de multiples nouveaux outils nécessaire à SOA, comme le bus de communication, le registre de services, ou l'orchestrateur de processus. Cette infrastructure peut inclure les outils nécessaires pour les administrateurs métiers, notamment les consoles BAM et SAM.

L'utilisation de ces outils BAM est un stade ultime de maturité sur une plate-forme SOA, puisqu'il requiert à la fois une modélisation précise des performances, une gestion des capacités, et la remontée en temps réel des ressources disponibles. Il est dans ce cas souhaitable de commencer par un système SAM.

# TROISIÈME PARTIE

# SOA : tout repose sur la méthode

L'objectif de cette partie est de fournir un guide pratique permettant de vaincre le « syndrome de la page blanche » lors du lancement d'un projet SOA : comment débuter puis mener à bien un tel projet ? Comment modéliser services et applications composites ? Qu'est-ce qu'un Processus Métier dans le contexte SOA ? Comment organiser un projet SOA ? Quels sont les pièges à éviter ? Quels sont les thèmes sur lesquels maîtrise d'ouvrage et maîtrise d'œuvre doivent collaborer étroitement ? Telles sont les exemples de questions auxquelles cette partie se propose de répondre.

Pour débuter un projet, il faut d'abord en définir le périmètre au sein du système d'information : le chapitre 7 montre que raisonner en terme d'application n'est plus suffisant et propose le concept de « solution métier » pour remédier à cette insuffisance.

Il s'agit ensuite d'analyser et de modéliser l'architecture à mettre en œuvre. Pierre angulaire de l'approche SOA, cette démarche a pour objectif de décrire les choix structurants en terme de composants à construire, avec leur typologie, et les relations entre ces composants. Il s'agit de guider ainsi les responsables de projet et les architectes dans leur démarche de spécification, de modélisation et de choix d'outils.

Les services sont les composants de base de l'approche SOA : le chapitre 8 propose une démarche pour modéliser ces composants.

Cependant, comme on l'a vu dans la partie précédente, SOA ne se réduit pas à la mise en place de services. Il est également nécessaire de modéliser les nouvelles applications composites (processus métier et applications interactives) clientes de ces services.

Le chapitre 9 montre tout d'abord comment toute architecture SOA doit prendre en compte la dimension « processus métier ». Le chapitre 10 décrit ensuite l'architecture d'une application interactive, en revisitant le modèle architectural traditionnel de ces applications.

Un projet SOA reste un projet informatique : les questions de robustesse et de performance sont des facteurs à analyser dès le départ, et seront donc abordées dans ces chapitres, en se focalisant uniquement sur les spécificités SOA.

Une fois la cible définie et son architecture modélisée, il est possible de proposer une organisation et une démarche pour réaliser et déployer cette cible. Le chapitre 11 propose une réflexion sur ce thème sensible, car il touche à la dimension humaine du métier de l'informatique.

Cette partie est consacrée à la modélisation d'une architecture SOA lors des phases d'expression de besoin et de spécifications[1] d'un projet : elle se veut donc indépendante de toute technologie permettant de réaliser des services et des applications composites. On espère ainsi convaincre concrètement le lecteur que SOA n'est décidément pas réductible à la seule mise en œuvre des technologies Web Services.

1. Si l'on respecte le cycle de vie défini par la méthode *Unified Process* (UP) associée à UML, ces phases correspondent aux phases d'expression de besoin, d'analyse et de conception générale décrites dans UP.

# 7

# Définir la cible

**Objectif**

L'objectif de ce chapitre est de présenter le concept de solution métier, puis d'illustrer ce concept en l'appliquant à un cas d'école, que le chapitre décrit, et qui servira de « fil rouge » pour les parties 3, 4 et 5 de cet ouvrage.

## 7.1 APPLICATION OU SOLUTION MÉTIER ?

### 7.1.1 Le point de départ

Pour débuter un projet informatique, il est nécessaire d'en connaître la cible. Pour cela, on raisonne en général en terme de besoin, et de réponse à ce besoin en terme d'application : application de gestion des commandes, application de gestion du catalogue produit, application de centre d'appel, etc.

Qu'est-ce qui caractérise une application ? Elle est à la fois :

- une unité métier : 1 application = la réponse à 1 attente majeure d'une ou plusieurs directions métier;
- une unité organisationnelle au sein de la direction des études : 1 application = 1 projet à réaliser, 1 ligne budgétaire identifiée, 1 équipe dédiée;
- une unité technique au sein de la direction de l'architecture et des outils : 1 application = 1 architecture et des outils de développement;
- et une unité au sein de la direction de la production : 1 application = 1 infrastructure matérielle et logicielle de base à compléter et intégrer, puis 1 exécutable à déployer, administrer et supporter.

Or l'architecture de référence SOA fait voler en éclat cette unité, en distinguant différents types de composants :

- Les services, dépositaires de la logique métier.
- Les applications « composites », orientées utilisateur final, qui sont clientes des services.
- Les référentiels, sur lesquels s'appuient les services.

### 7.1.2 Que devient alors la notion d'application ?

Application = services ? Mais d'un point de vue métier, ce qui intéresse les utilisateurs, ce ne sont pas les services mais bien les applications composites dont ils sont amenés à se servir, et qui automatisent tout ou partie des processus métier. Un service en soi n'est en effet pas directement utilisable sur le plan des usages métier, il n'intéresse les utilisateurs qu'au travers de ces fameuses applications composites.

Il est alors tentant de poser application = application composite. Mais on ne peut déployer une application composite sans que les services dont elle est cliente soient eux-mêmes déployés. D'autant que certains services n'existent peut-être pas encore. De plus, il n'y a pas nécessairement une seule application composite à mettre en place pour satisfaire le besoin métier, mais plusieurs applications composites : processus métier, application interactive permettant aux utilisateurs finaux d'intervenir dans les processus ou de traiter les exceptions levées par ces processus, application *back office* permettant d'administrer les référentiels et les processus métier, etc.

### 7.1.3 Le concept de solution métier

Il devient donc nécessaire de raisonner en terme de **solution métier**. Une solution métier regroupe les différents macro-composants nécessaires à la satisfaction des besoins utilisateurs. Une solution métier regroupera donc :

- Le ou les processus métier qui organisent le traitement des événements métier reçus par l'entreprise, et peuvent être plus ou moins automatisés.
- La ou les applications interactives « front office » qui outillent les processus métier, et permettent aux utilisateurs finaux de participer au traitement des événements métier.
- La ou les applications composites *back office* qui permettent d'administrer les référentiels (en particulier, les applications de mise à jour des codiers[1] et référentiels de paramétrage) et les processus métier (en particulier, les applications de monitoring des processus).
- Les services utilisés par les différentes applications composites, notamment les services métier.

1. Les codiers sont les référentiels permettant de centraliser et de mettre à jour les différentes nomenclatures utilisées dans l'entreprise.

- Les référentiels accédés par les services.
- Les ressources techniques utilisées par les applications composites ou par les services.

### 7.1.4 La caractéristique principale d'une solution métier

Par rapport au concept classique d'application, la solution métier garde donc bien une unité « métier » : elle regroupe tout ce qui est nécessaire à la satisfaction du besoin métier.

Mais sur le plan de la direction informatique ? La solution métier n'est plus vraiment une unité de déploiement physique, car tout ou partie des référentiels, des ressources techniques, et surtout des services utilisés est sans doute déjà réalisé et physiquement déployé. Et c'est certainement un des avantages de l'architecture SOA que de permettre la réutilisation de ce qui fonctionne déjà en production.

La solution métier n'est pas non plus une unité technique en soi, car l'un des objectifs de SOA est bien de mettre en place une infrastructure logicielle réutilisable entre les différentes solutions métier : autrement dit, ce n'est pas à la solution métier de définir sa propre architecture (Web Services versus EJB…), d'assembler son propre framework applicatif, de choisir ses outils (ESB…), etc. Chaque solution doit

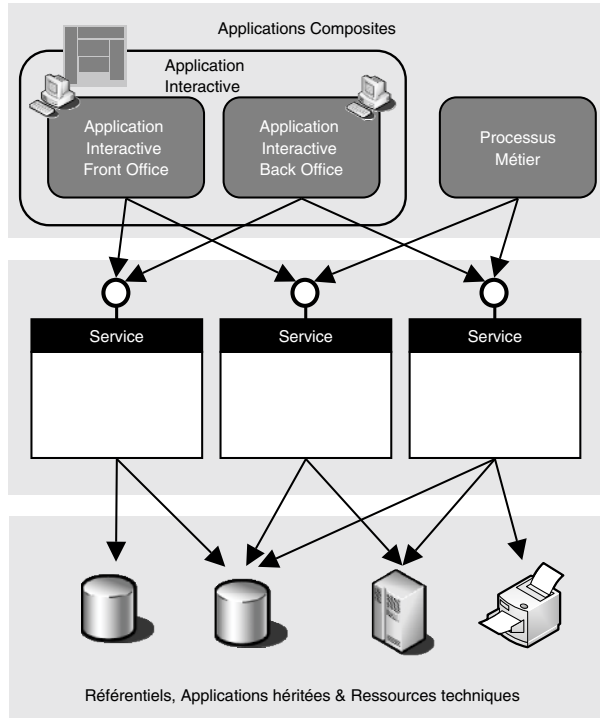



**Figure 7.1** — Une solution métier s'appuie sur le modèle de référence SOA

bien entendu contribuer à ces choix communs, mais elle ne doit pas en toute rigueur en être responsable.

De plus, le fait que des services métier soient réutilisés par différentes solutions ou que l'infrastructure logicielle soit partagée entre ces solutions a bien entendu un impact positif sur le budget présenté aux directions métier, mais sur le plan des usages et de la finalité métier, ce n'est pas la préoccupation première de ces directions.

La préoccupation première d'une direction métier est que ses utilisateurs soient efficaces dans leur métier, c'est-à-dire qu'ils puissent traiter un événement métier de bout en bout, en étant productifs (nombre d'événements traités par heure/jour/semaine) et réactifs (durée de traitement moyen d'un événement, etc.).

De ce fait, toute solution métier est d'abord et avant tout une unité d'intégration : le responsable d'une solution métier garantit à ses clients (la ou les directions métier) que l'ensemble des composants impliqués dans la solution dont il est responsable réponde collectivement (c'est-à-dire de façon intégrée) aux besoins exprimés. Il offre ainsi une **garantie de bonne fin**.

Cette intégration porte donc sur les différents composants illustrés par la figure 7.1.

## 7.2 PRÉSENTATION DU CAS D'ÉCOLE

Ce paragraphe présente le cas d'école qui nous servira de fil rouge, puis présente la solution métier que ce projet « fil rouge » doit livrer aux utilisateurs finaux.

### 7.2.1 L'entreprise concernée

L'entreprise concernée est un distributeur d'énergie (gaz et électricité) d'Europe de l'Ouest. Ses clients sont soit des particuliers, soit des entreprises, PME et grands comptes.

Le métier de cette entreprise est donc (i) de gérer et maintenir un réseau de distribution de fluide sur son territoire de chalandise, (ii) d'installer chez ses clients des Points De Distribution (PDD) branchés sur son réseau, et (iii) de facturer la consommation relevée dans chaque Point De Distribution. Cette entreprise est ce qu'on appelle généralement un **opérateur** de réseau.

Jusqu'à présent, cet opérateur distribuait l'énergie fournie par le monopole national de production d'énergie, dont il était de facto une filiale. Son système d'information était donc intégré dans celui du monopole, puisque ses clients étaient exactement ceux du monopole de production.

### 7.2.2 Contexte métier

Sous la pression des instances européennes, la libéralisation du marché de l'énergie est en route, et notre opérateur de réseau doit s'ouvrir à des producteurs alternatifs.

Ces producteurs veulent distribuer leur énergie vers leurs propres clients, via les points de distribution et le réseau gérés par l'opérateur. Ce réseau représente en effet un investissement si considérable, qu'il n'est pas concevable que chaque producteur alternatif se dote de son propre réseau de distribution.

Cette ouverture est inéluctable. L'opérateur souhaite transformer cette contrainte légale en une opportunité commerciale : les producteurs alternatifs deviennent en quelque sorte de nouveaux clients, auxquels l'opérateur doit fournir des services, payants, sur un pied d'égalité avec l'ex monopole national de production d'énergie.

L'ouverture à la concurrence se fait progressivement : dans un premier temps, les producteurs alternatifs sont autorisés à servir les clients professionnels. Dans un second temps, la concurrence concerne également les clients particuliers.

### 7.2.3 Objectifs du projet

L'opérateur souhaite mettre en place dans une première étape un portail destiné aux producteurs. Ce portail permet à un producteur de demander un service à l'opérateur, tel que la mise en place d'un nouveau Point De Distribution, ou la consultation des relevés de consommation associés à un PDD.

La mise en place du portail s'accompagne de la mise en place des processus *back office* de traitement de ces demandes. Il s'agit par exemple de planifier et lancer les interventions des équipes techniques d'intervention sur les PDD. Pour cela la solution métier s'appuie sur des applications existantes (planification des interventions, gestion des comptes-rendus d'intervention).

Le premier objectif métier est donc de permettre à un producteur d'émettre manuellement ces demandes de prestation via des formulaires proposés par le portail, et de traiter ensuite ces demandes.

Dans une seconde étape, il s'agit de faire face à l'accroissement prévisible des demandes de prestation, accroissement lié à l'ouverture aux particuliers. L'opérateur souhaite donc relier directement le système d'information de chaque producteur à son propre système d'information.

Le deuxième objectif métier du projet est donc que chaque producteur envoie directement ses demandes via un protocole spécifique du monde de l'énergie (protocole standard basé sur XML). Formulé autrement : il s'agit de mettre en place un accès multi-canal à l'offre de prestations de service de l'opérateur de réseau.

Quel que soit le canal d'acheminement de la demande de prestation, l'opérateur souhaite ne pas multiplier ses équipes commerciales : le troisième objectif métier est de rechercher une automatisation maximale des processus de traitement des demandes de prestation.

Le dernier objectif métier est bien évidemment d'optimiser les coûts et encore plus les délais du projet.

Pour répondre à ces objectifs, et malgré la relative jeunesse de cette approche, il est décidé de s'appuyer sur une architecture SOA, pour bénéficier des avantages de cette approche, notamment :

- Mise en place de processus automatisés flexibles.
- Réutilisation des processus et des services quel que soit le canal d'envoi d'une demande de service (mise en place d'une approche multi-canal).
- Intégration avec les systèmes existants via des services spécifiques.

Le projet se nomme PORTOS (Portail Opérateur Réseau pour la Transmission de demandes, Orienté SOA).

### 7.2.4 Cas d'utilisation général

#### *Scénario d'usage*

Les différentes étapes du scénario général d'utilisation de PORTOS sont :

- Étape 1 : Via le portail, un producteur dûment authentifié demande une prestation :
  - L'utilisateur peut s'interrompre en cours de saisie (concept de « demande brouillon »).
  - L'utilisateur peut également consulter un certain nombre d'informations : sur les demandes en cours, sur les PDD qui lui sont affectés, sur les relevés de consommation associés à ses PDD, etc.
- Étape 2 : PORTOS enregistre la demande et crée un processus métier automatisé de traitement de la demande.
- Étape 3 : Le processus planifie les interventions des équipes techniques sur le terrain, puis lance ces interventions (envoi de message à l'équipe concernée en temps et en heure).
- Étape 4 : Chaque producteur peut demander au système l'état d'avancement de chacune de ses demandes. Ceci implique donc l'existence d'une base « historique des demandes ».
- Étape 5 : Le système avertit le fournisseur par mail de certains événements métier : refus de la demande, fin de traitement de la demande.

#### *Contraintes principale*

Les contraintes du projet sont principalement les suivantes :

- La solution doit gérer la volumétrie des demandes de prestation, liée à l'ouverture à la clientèle des particuliers.
- La solution doit être évolutive à peu de frais, afin d'offrir de plus en plus de services aux producteurs alternatifs : consultation de leurs factures, paiement en ligne, etc. La mise en place progressive de ces offres doit être aussi légère que possible.

- La solution doit être sécurisée.
- La solution doit être mise en place dans des délais courts.

### 7.2.5 Contenu de la solution métier

La solution métier PORTOS comprendra donc au moins les composants suivants :

- Application interactive : demander une prestation.
- Processus métier : traiter automatiquement une demande de prestation.
- Application interactive : rechercher une demande déjà transmise.
- Application interactive : consulter l'état d'une demande particulière.
- Application interactive : rechercher les Points De Distribution utilisés par le producteur.
- Application interactive : consulter les relevés de consommation associés à un PDD.
- Services : services métier d'accès à l'information.
- Services : services d'interface avec les systèmes existants.
- Référentiels métier : demandes, PDD, relevé de consommation, identités des utilisateurs du portail – nous ne précisons pas ici si ces référentiels sont dupliqués ou non, cette problématique sera abordée dans le chapitre 8. Si une décision de duplication était prise, la solution métier devrait intégrer une application de réplication de données.
- Référentiel technique : journal de bord de l'utilisation du système.

### 7.2.6 Architecture générale envisagée

La figure 7.2 présente l'architecture très générale de la solution envisagée. Cette architecture tient compte dès le départ de la contrainte de sécurisation, séparant et isolant *front office* et *back office*.

Les applications interactives seront intégrées à un portail qui assure à ses utilisateurs un accès homogène sur le plan ergonomique, unifié sur le plan de la sécurité d'accès, et intégré sur le plan technique.

Le portail assure également l'accueil et l'information des utilisateurs. Cette information concerne aussi bien les services offerts via le portail (informations marketing et commerciales, informations juridiques, accès à d'autres sites web, etc.), que le mode d'emploi de ces services (guides d'utilisation, FAQ…).

Les demandes envoyées automatiquement via un flux XML seront accueillies par un serveur baptisé « Frontal XML ». Ce frontal effectue les tâches suivantes :

- sauvegarder tous les flux reçus, afin de garantir qu'aucune demande ne sera perdue dès lors qu'elle est reçue par l'opérateur de réseau;

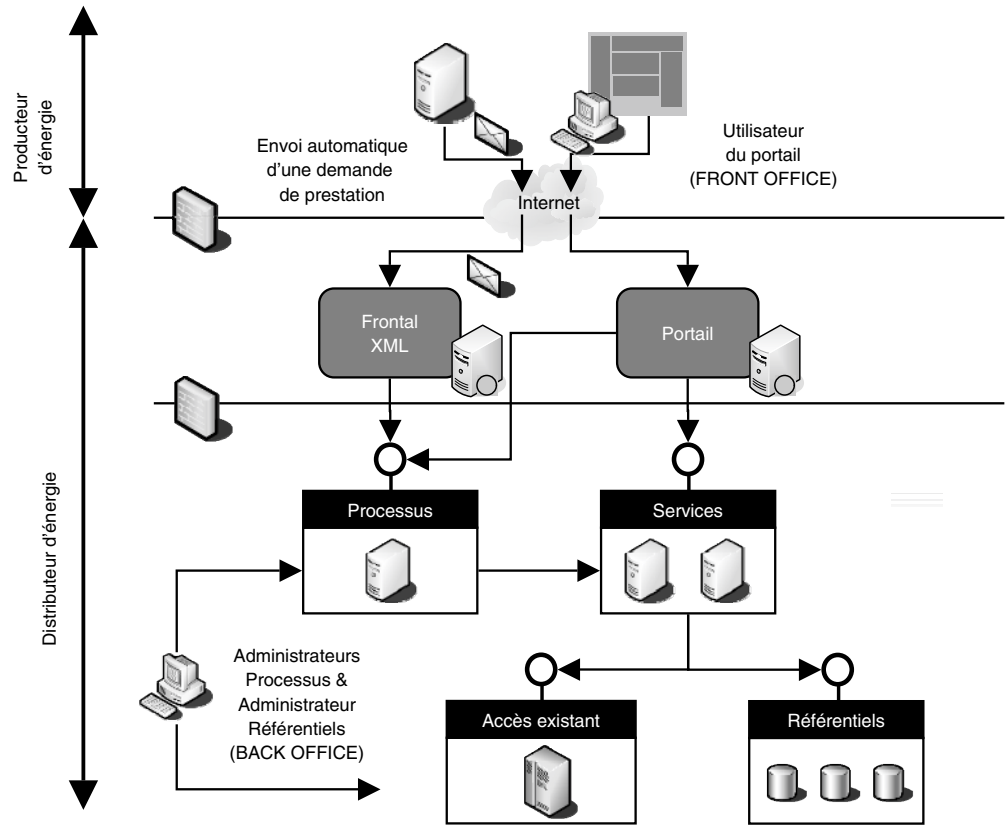



**Figure 7.2** — Architecture générale de la solution métier PORTOS

- contrôler la syntaxe XML des flux reçus, c'est-à-dire s'assurer que chaque information contenue dans un flux respecte sa définition (contenue dans un schéma XML);
- aiguiller chaque flux vers le gestionnaire de processus.

**Un *pattern* architectural ?**

Le fil rouge présenté est un cas d'école qui est de facto générique. Dès lors qu'une entreprise souhaite ouvrir son système d'information à ses clients via le canal Internet, la solution métier qu'elle devra mettre en place distinguera d'une part un front office et d'autre part un *back office*.
Le front office sera en général composé de deux canaux d'accès : un accès interactif via un portail, intégrant des applications composites interactives, et un accès de type EDI, via un frontal réceptionnant des flux XML.
Le front office accédera à un back office, offrant (1) des processus métier (un processus dans le cadre SOA étant lui-même considéré comme un service de haut niveau), (2) des services (services métier, services techniques) et (3) des applications interactives de back office.

Les chapitres suivants vont nous permettre de préciser le contenu de cette architecture, en modélisant certains de ces composants, en faisant apparaître des composants complémentaires, et (dans les parties suivantes) en précisant les outils/normes/langages utilisés et leur mise en œuvre concrète.

## En résumé

Le concept d'application n'est plus adapté au contexte SOA. Il est préférable de parler de solution métier.

Une solution métier comprend l'ensemble des composants nécessaires pour rendre le service attendu par les utilisateurs métier. Elle garantit leur bonne intégration fonctionnelle et technique, y compris l'intégration des composants déjà en production.

# 8

# Modéliser les services

## Objectif

L'objectif de ce chapitre est de présenter une démarche de modélisation des services à déployer. Par définition, l'approche SOA consiste avant tout à mettre en place des services réutilisables et/ou interopérables. Or une fois défini ce concept de service, commencent les difficultés : que sont donc ces services ? Y a-t-il plusieurs sortes de services ? Quelles sont les précautions à prendre pour obtenir une modélisation acceptable ? Comment prendre en compte les systèmes existants ?

En bref, une fois l'effervescence de la nouveauté passée, il s'agit de modéliser une architecture de service, et d'utiliser cette architecture pour concevoir, développer et déployer une première version de la bibliothèque de services.

Les premiers projets SOA ont fait émerger cette architecture de service par étapes, chaque étape faisant apparaître des forces et des faiblesses de l'architecture mise en place. Le chapitre s'appuie sur le fil rouge pour mettre en évidence ces différentes étapes, et permettre ainsi de gagner du temps.

On verra également que le critère de performance joue un rôle clef, comme souvent.

## 8.1 ARCHITECTURE DE SERVICES : LES QUESTIONS CLEF

### 8.1.1 La typologie des services

La première question concerne la typologie des services : existe-il plusieurs catégories de service ? La réponse est clairement oui.

**Règle :** on distinguera (i) les services métier et (ii) les services techniques.

#### *Les services métier*

Le rôle d'un service métier est d'offrir un ensemble cohérent de traitements métier : cela peut concerner par exemple l'accès à la « vision client » (obtenir un ensemble synthétique d'information sur le client, l'état de ses comptes, l'historique de ses relations avec l'entreprise…), ou la simulation d'un crédit immobilier, ou le calcul d'impôts et de taxes diverses, etc.

Un service métier peut être soit un service d'accès à des informations, soit un service de calcul et de vérification de règles métier, soit une composition des deux.

#### *Les services techniques*

Le rôle d'un service technique est de donner accès à une ressource technique donnée : on citera par exemple l'accès aux bases de données relationnelles, au CICS d'un mainframe, à une imprimante, à un système de GED, à un outil de log, à une messagerie, à un EAI, etc.

Un service technique est très souvent générique : il n'est pas lié à une ressource particulière mais à une catégorie de ressources. Par exemple, un service d'impression ne sera pas lié à une imprimante particulière, mais à une catégorie d'imprimante : il prendra en paramètre l'adresse de l'imprimante sur laquelle sortiront les informations à imprimer.

Un service métier peut s'appuyer sur un ou plusieurs services techniques pour exécuter ses traitements. Un service métier « synthèse de la vision client » pourra par exemple s'appuyer sur un service technique d'accès SQL, sur un service technique d'accès CICS (si on suppose que les informations nécessaires à cette synthèse sont réparties entre des référentiels légataires sur mainframe et des référentiels plus récents sous SGBDR), et sur un service technique d'impression.

### 8.1.2 La granularité des services

La seconde question concerne la granularité des services. Le terme de « granularité » a été introduit lorsque les évangélistes de l'objet ont constaté sur le terrain que modéliser un objet « client » ou un objet « facture » n'était pas de même nature que modéliser un objet « adresse électronique » : les premiers sont des « gros » objets, et contiennent eux-mêmes d'autres objets, alors que les seconds sont de « petits » objets, avec peu d'attributs. On parle d'objets « à gros grain » et d'objets « à grain fin », traduction de l'américain « *coarse grain* » et « *fine grain* ».

On retrouve la même problématique dans le monde SOA : il y a des services à gros grain et des services à grain plus fin. Les services à gros grain accèdent aux services à grain fin pour effectuer leurs traitements.

Le paragraphe suivant approfondit cette réflexion en proposant une typologie des services métier, liée à leur granularité.

# 8.2 MODÉLISER LES SERVICES

L'objectif de ce sous-chapitre est de proposer une démarche de modélisation montrant concrètement comment émerge, étape par étape, une typologie hiérarchique de services, liée à la granularité de ces services. On illustre ces étapes au travers du fil rouge, mais en précisant d'emblée que cette démarche et surtout ses résultats sont désormais suffisamment éprouvés pour que l'on puisse en garantir l'applicabilité à la plupart des projets SOA, notamment dans le monde de la gestion. On s'intéresse donc principalement aux services métier.

## 8.2.1 Étape 1 : les services CRUD

Les services métier qui émergent spontanément au démarrage d'un projet SOA sont les services CRUD (acronyme anglais qui signifie *Create Read Update Delete*). Ces services permettent de créer, rechercher, mettre à jour et supprimer de l'information dans les référentiels du système d'information. Comment modéliser ces services ?

Chaque service CRUD est associé à un objet métier « racine »[1], et offre l'ensemble des opérations permettant de créer, rechercher et mettre à jour les instances de cet objet métier.

Mais qu'est-ce qu'un objet métier « racine » ? C'est un objet métier qui constitue un point d'entrée dans le système d'information. Par exemple, le Client, le Contrat liant l'entreprise et le client, la Commande, le Catalogue des produits, la Facture… peuvent être autant d'objet métier « racine ». A contrario, l'adresse bancaire d'un Client sera un objet métier mais ne sera pas un objet « racine » du système d'information. En terme d'Urbanisation, on peut dire qu'un bloc d'urbanisation est en charge de gérer un objet métier racine[2] et repose sur un ensemble d'opérations CRUD, regroupées dans un service.

Un service CRUD se focalise donc sur les opérations « universelles » de manipulation de l'information. Ces opérations sont :

- Créer un nouvel objet métier « racine » (plus exactement créer une nouvelle instance de la classe métier) : par exemple, « créer le client [nom prénom] » est une opération offerte par le service CRUD associé à l'objet métier « client ».

1. Du point de vue informationnel, un objet métier est une catégorie, ou **classe**, d'individus ayant les mêmes attributs et obéissant aux mêmes règles de gestion métier. Un individu étant une chose concrète (un contrat) ou abstraite (un intervalle de temps), une personne concrète (personne physique) ou abstraite (personne morale), un lieu…
Du point de vue technique, un objet métier « logiciel » est construit avec le concept de classe des langages objet.
On parlera d'**instance** pour désigner un individu appartenant à la classe, sur le plan informationnel comme sur le plan technique.
2. Ou un petit nombre d'objets métier racines fortement liés entre eux, comme la Commande et la Ligne de Commande, ou le Catalogue produit et ses Rubriques.

- Rechercher dans le ou les référentiels concernés l'ensemble des objets métier satisfaisant un critère de recherche : par exemple, « rechercher l'ensemble des clients "habitant la Haute Savoie" ».
- Mettre à jour un objet métier existant : par exemple, « mettre à jour l'adresse électronique du client "Jean Martin" ».
- Supprimer un objet métier – comme il est rare de détruire réellement un Objet Métier dans un système de gestion, le delete sera souvent complété[1] par une opération d'archivage.

Pour lire ou écrire un objet métier « racine », le service CRUD associé accédera en général à plusieurs tables du référentiel concerné (table « client », table « adresse »…), voire à plusieurs référentiels. Un service CRUD masque donc la complexité intrinsèque du modèle d'objets métier manipulé.

Ces services CRUD sont importants, car ils sont fortement réutilisables par les solutions métier.

### *Étape 1 bis : les services techniques*

Certains services techniques émergent également très rapidement, en liaison avec les services CRUD (c'est pourquoi on parle d'étape 1 bis), comme par exemple les services de communication avec les systèmes existants, les services d'accès aux référentiels SQL, et les services de gestion de log.

### *Application au fil rouge*

La figure 8.1 illustre l'application de l'étape 1/1 bis sur l'exemple « fil rouge ».

La figure met en évidence la nécessité de sécuriser l'accès aux services offerts par le système d'information. De ce fait, le portail est situé dans une zone démilitarisée (DMZ), et le dialogue entre portail et service doit franchir le pare-feu, ce qui conduit à utiliser de préférence les web services, c'est-à-dire HTTP, comme protocole de communication (par rapport à d'autres outils tels que les EJB Session par exemple)[2].

Dans cette architecture, pour traiter une demande de prestation, le portail accède via les services CRUD aux référentiels pour (1) vérifier les droits de l'utilisateur, (2) sauvegarder la demande de prestation, (3) accéder à la base Contrat pour vérifier que la demande correspond bien aux termes d'un contrat valide entre l'Opérateur et le Demandeur, (4) accéder au Gestionnaire de Processus pour créer un nouveau processus automatisé de traitement de cette demande, (5) envoyer si besoin un mail à un responsable commercial de l'Opérateur (6) enregistrer des logs si besoin. Le portail peut également rechercher des informations sur les Points De Distribution, sur l'état d'une demande, etc.

1. L'opération de suppression ne doit pas être supprimée, car elle peut être utilisée, par exemple en phase d'intégration, par une application de purge des référentiels de tests.
2. On consultera avec profit le livre de référence : *Sécurité des Architectures Web*, G. PLOUIN et alii, DUNOD éditeur, 2004.

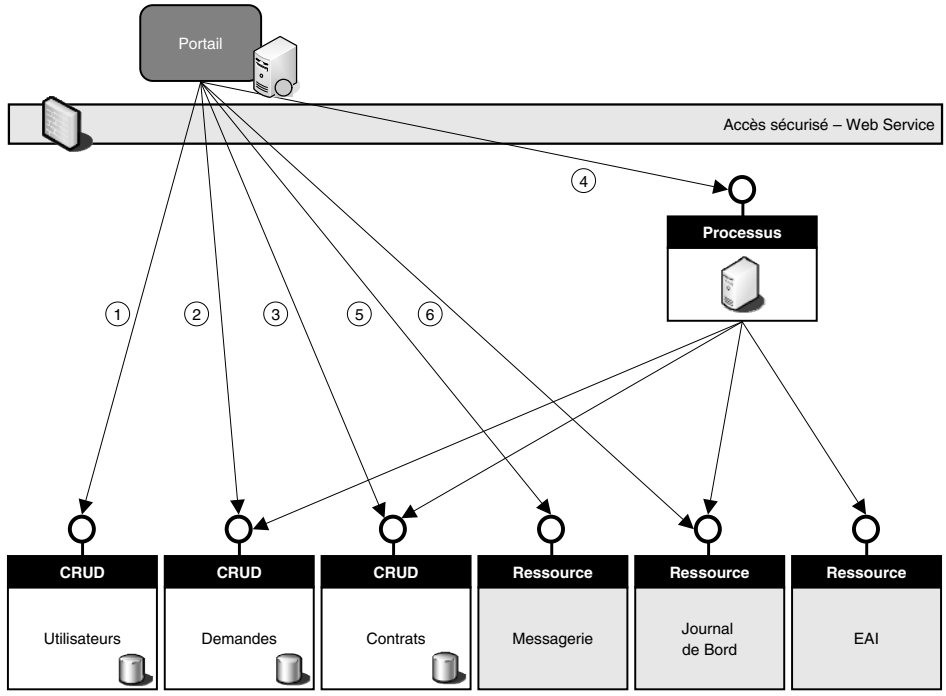



**Figure 8.1** — Première étape : les services CRUD et les services techniques

Pour éviter de surcharger la figure, la liste des services CRUD n'est pas exhaustive. Les services CRUD nécessaires sont associés aux objets métier suivants :

- Demandes : le service CRUD associé permet notamment d'enregistrer une demande et d'effectuer des recherches sur la base des Demandes.
- Fournisseurs et Contrats : le service CRUD associé permet notamment de rechercher si le fournisseur émettant une demande de prestation possède un contrat en bonne et due forme, puis de lire ce contrat en base.
- PDD : le service CRUD associé permet notamment d'effectuer des recherches sur les PDD utilisés par un fournisseur.
- Utilisateurs portail (gestion de l'identité) : le service CRUD associé permet notamment de rechercher le profil d'un utilisateur identifié.

### *Analyse des étapes 1 et 1bis*

L'architecture obtenue met en évidence à l'issue de ces étapes un socle de services que l'on pressent comme fortement réutilisables. Un premier pas important est donc franchi – mais quels sont les défauts de cette architecture ?

Le premier problème concerne les performances et provient de la granularité trop fine des services mis en place. Une telle architecture conduit en effet à multiplier les appels aux (web) services CRUD et aux (web) services techniques. Ceci se traduit par une utilisation intensive des outils de traduction [message XML ↔ objets métier

JAVA] – si on utilise JAVA comme langage d'implémentation des services. Or cette utilisation est coûteuse en temps CPU voire en mémoire.

Le deuxième problème concerne la réutilisabilité. On peut le résumer en se posant la question suivante : dans cette architecture, où sont les traitements métier proprement dit, et en particulier les règles de gestion ? En associant directement applications composites (le portail, le Gestionnaire de Processus) et services CRUD, l'architecture contraint l'application à effectuer elle-même ces traitements métier. D'où deux défauts : il n'y a pas de réutilisation possible de ces traitements et de plus l'application composite devient obèse. Cette architecture n'est donc pas facile à mettre en œuvre pour les concepteurs d'application composite, et n'optimise pas la réutilisation de logiciel.

Le troisième problème, et pas le moindre, concerne la robustesse de l'architecture : les Web Services ne sont pas transactionnels. Il n'y a donc aucune solution simple pour rendre cette architecture robuste. Des travaux sont certes en cours sur ce sujet, mais d'une part ils ne sont pas à ce jour terminés, et, une fois terminés, il restera d'une part à en évaluer la facilité de mise en œuvre et les performances, et d'autre part à attendre la maturité des implémentations.

### 8.2.2 Étape 2 : Les Services Applicatifs

L'étape 2 de la démarche proposée vise donc à résoudre ces problèmes. Elle consiste à mettre en place des services façades, c'est-à-dire des services de haut niveau qui masquent aux applications composites les services CRUD, de trop bas niveau.

#### *Application au fil rouge*

La figure 8.2 illustre le propos. On crée un service façade « enregistrer demande », service qui est appelé par le portail une fois qu'un utilisateur a rempli puis validé un formulaire de demande de prestation.

Les services mis en place sont des services de haut niveau, directement liés aux applications composites concernées. C'est pourquoi on parlera de Services Applicatifs. Une application délègue donc à un ou plusieurs Services Applicatifs le soin de coordonner les appels aux services CRUD et aux services techniques, et l'implémentation des règles de gestion métier.

#### *Analyse de l'étape 2*

Cette étape résout certains des problèmes listés précédemment – mais pas tous.

Sur le plan des performances, l'architecture minimise désormais l'accès du portail aux Web Services. De plus, comme le Service Applicatif est « du bon côté de la barrière » (c'est-à-dire du pare-feu), il peut faire appel aux Services CRUD sans utiliser les protocoles WS, donc sans utiliser les transformations XML ↔ langage d'implémentation (cf. partie 5). De ce fait, les performances du portail sont meilleures.

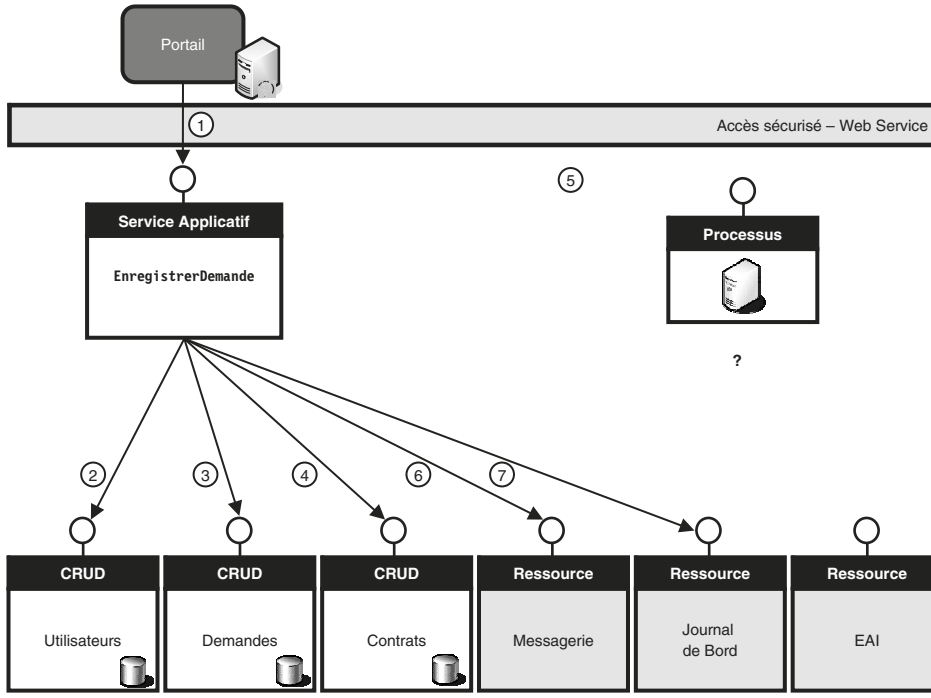



**Figure 8.2** — Deuxième étape : émergence des Services Applicatifs

De plus, la vie du concepteur du portail est simplifiée : l'architecture met à sa disposition un ou quelques services de très haut niveau, et n'a plus à se préoccuper des services de granularité trop fine.

Mais il reste encore des défauts importants. Sur le plan de la réutilisabilité, cette solution ne fait que déplacer le problème. L'application composite n'est plus obèse, mais ce sont le ou les Services Applicatifs qui risquent de l'être. De plus, cette solution, si elle facilite la mise en œuvre du portail, ne concerne pas le Gestionnaire de Processus, car les Services Applicatifs ne sont pas réellement réutilisables puisque spécifiques d'une application composite, le portail en l'occurrence. Le Gestionnaire de Processus est donc toujours contraint d'accéder aux services « de grain fin ».

Sur le plan de la robustesse, il est possible d'utiliser un mécanisme tel que les EJB Session pour implémenter l'accès aux services CRUD. Ce mécanisme supporte un protocole transactionnel de type « commit à deux phases ». Il serait donc possible d'améliorer la robustesse de l'architecture en se reposant sur les EJB, sans perdre en performance. Mais il est nécessaire de noter que certaines ressources techniques ne sont malheureusement pas des ressources transactionnelles : par exemple, un système classique de gestion de fichier ou un gestionnaire de mail ne sont pas, en général, transactionnels. Autrement dit, « mélanger » dans un même Service Applicatif l'accès à des ressources transactionnelles (les référentiels SQL) et des ressources non transactionnelles revient à avoir un service non transactionnel : le protocole de

commit à deux phases ne fonctionnera pas et l'architecture ne sera pas plus simple à rendre robuste qu'avant, malgré le recours aux EJB.

### 8.2.3 Étape 3 : Les Services Fonctionnels

La dernière étape vise donc à mettre en place des Services Métier encapsulant tout ou partie des règles de gestion et des traitements métier. On les appellera de ce fait des Services Fonctionnels, par analogie avec les fonctions métier qui regroupent dans les systèmes classiques un ensemble cohérent de règles et traitements métier.

Un Service Fonctionnel s'appuie le cas échéant sur un ou des services CRUD pour lire les informations métier dont il a besoin et/ou sauvegarder les informations métier qu'il a modifiées/créées.

#### *Application au fil rouge*

La figure 8.3 illustre le propos sur notre « fil rouge », en mettant en évidence trois services fonctionnels :

- Un Service Fonctionnel associé à la gestion des demandes.
- Un Service Fonctionnel associé à la création d'un nouveau processus métier afin de traiter une nouvelle demande.

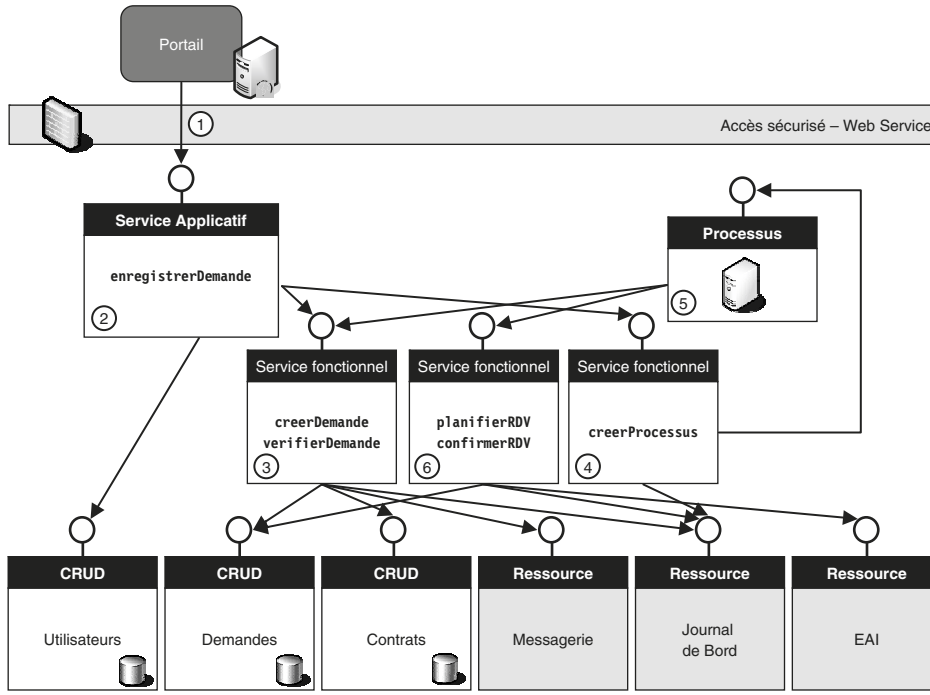



**Figure 8.3** — Troisième étape : émergence des services Fonctionnels

- Un Service Fonctionnel associé à la gestion automatisée de la prise de rendez-vous.

Plus précisément, le Service Applicatif « enregistrer la demande de prestation », appelé par le portail, fait lui-même appel au Service Fonctionnel « créer une demande » puis au Service Fonctionnel « créer le processus métier associé à cette nouvelle demande ».

Le Service Fonctionnel « créer une demande » exécute les traitements suivants :

- Appel du service technique (non représenté sur la figure) « créer un numéro unique de demande » : ce service permet de récupérer un numéro unique d'identification auprès du référentiel.
- Appel du service CRUD « Créer Demande » (correspondant à la sauvegarde dans le référentiel concerné des informations du formulaire, avec le numéro unique d'identification de la demande).
- Appel du service CRUD « Mettre à jour Historique de la Demande » : l'état de la demande devient [demande enregistrée].

Le Service Fonctionnel « créer le processus métier associé à une demande » exécute les traitements suivants :

- Exécution de la règle de gestion « Extraire de la demande les paramètres d'appel du gestionnaire de processus ».
- Appel du service « Créer un processus dans le BPM » – ce service renvoie l'identité `Id_processus` du processus créé.
- Appel du service CRUD « Mettre à jour Demande : `{demande.processus = Id_processus}` ».
- Appel du service technique « Mettre à jour le journal de Bord ».

On remarquera que le Service Fonctionnel « créer une demande » peut être considéré comme transactionnel, car il fait appel à des services CRUD qui reposent eux-mêmes sur une ressource transactionnelle (le SGBDR). En revanche, le Service Fonctionnel « créer un processus » n'est pas transactionnel, car ni le moteur de gestion de processus ni le journal de bord (c'est-à-dire le système de gestion de fichier) ne sont une ressource transactionnelle.

### *Analyse de l'étape 3*

L'architecture obtenue résout les problèmes évoqués auparavant. La facilité de mise en œuvre de cette architecture est meilleure, car les concepteurs des Applications Composites, en particulier les concepteurs des processus métier, se voient proposer des services de haut niveau plus facilement réutilisables.

> Le prix à payer pour obtenir une telle architecture, c'est la réflexion qui doit présider à l'émergence de la bibliothèque de services fonctionnels. Si en effet les Services Applicatifs et les services CRUD émergeront relativement rapidement, il n'en va pas de même pour les Services Fonctionnels.

La modélisation des Services Fonctionnels sera guidée par une double réflexion :

- Réflexion fonctionnelle, « top – down » : les Services Fonctionnels émergent soit de la modélisation des processus métier – chaque étape d'un processus implique l'appel à un Service Fonctionnel –, soit de la modélisation des traitements et règles de gestion métier – un Service Fonctionnel regroupera un ensemble cohérent de règles et/ou de traitements.
- Réflexion technique, « bottom – up » : le concepteur s'arrange pour que le regroupement des services de bas niveau (CRUD et technique) se fasse de façon à ce qu'un maximum de Services Fonctionnels soient transactionnels.

On remarquera également que l'architecture SOA proposée « n'interdit pas » à un Service Applicatif (ni même à une application composite) d'appeler directement un service CRUD, lorsque c'est utile. La démarche proposée n'est pas de multiplier les couches de services pour le plaisir, ni d'ériger en dogme des pratiques où le bon sens doit primer avant tout.

### 8.2.4 Synthèse de la démarche : modéliser l'architecture de services

La figure 8.4 illustre le résultat de cette démarche.

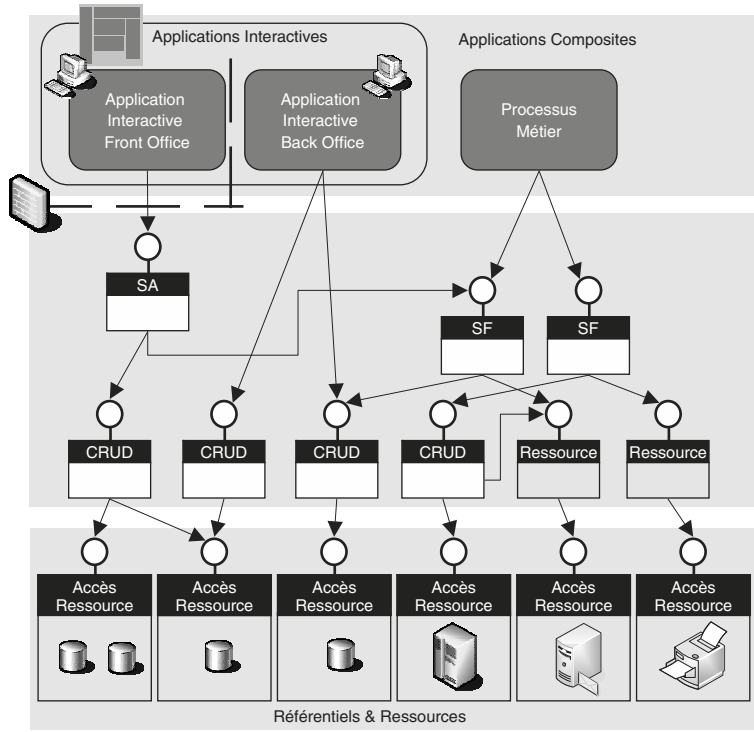



**Figure 8.4** — Synthèse de l'architecture proposée

**Règle :** appliquer une démarche ayant pour objectif la mise en place d'une typologie de service fondée sur la granularité des services et leur spécialisation.

Cette typologie constitue de facto un guide de modélisation pour l'architecte des services.

## 8.3 ZOOM SUR LES SERVICES CRUD

Les services CRUD jouent un rôle important dans une architecture SOA. Ce chapitre propose une description plus détaillée de ces services.

### 8.3.1 Quelles sont les opérations offertes par un service CRUD ?

Un service CRUD est, rappelons-le, associé à un objet métier « racine », et résulte de la nécessité pour une application SOA d'interroger et de mettre à jour le(s) référentiel(s) dans lequel sont enregistrées de façon persistante les informations décrivant les instances de cet objet métier.

Un tel service offre en général les opérations suivantes :

- Créer un nouvel objet métier dans le référentiel concerné.
- Rechercher un ensemble d'objets métier selon des critères de recherche : par exemple, « rechercher l'ensemble des clients avec le critère {habite dans la commune identifiée par le code postal "91190"} ».
- Lire un objet métier selon une clef d'accès unique, que ce soit une clef technique (exemple : primary key d'un SGBDR), ou une clef métier (exemple : numéro de sécurité sociale).
- Lire une collection d'objets métier selon une collection de clefs d'accès uniques : il s'agit d'une variante « batch » de l'opération précédente, qui permet d'optimiser la performance de la recherche en base de données.
- Mettre à jour un objet métier.
- Archiver un objet métier ou un ensemble d'objets métier.

Certains architectes préféreront séparer dans deux services distincts ce qui relève de la Recherche dans les référentiels (opération de Lecture et de Recherche), et ce qui relève de la Mise à jour de ces référentiels, autrement dit, avoir un service « R » et un service « CUD ». Ceci pour mieux contrôler l'accès aux services de mise à jour, critiques quand au maintien de l'intégrité des référentiels.

On peut également, en fonction des besoins, ajouter des opérations, qui sans être strictement CRUD, n'en sont pas moins fort utiles, comme :

- Imprimer un objet métier.

- Exporter un objet métier vers un format donné (les principaux formats étant XML, PDF, CSV pour EXCEL).
- Importer un objet métier à partir d'un format donné.

### 8.3.2 La signature des opérations CRUD

La signature d'une opération regroupe le nom de cette opération, et la liste de ses paramètres d'entrée et de sortie. S'interroger sur la signature des principales opérations CRUD permet de mettre en évidence quelques questions clefs concernant l'efficacité de ces opérations.

#### *Opération de lecture*

La signature d'une opération de lecture est la suivante :

```
LireXXX(ClefDeLecture, ScenarioDeChargement, Contexte)
```

XXX doit être remplacé par le nom de l'objet « racine » concerné : lireClient, lireDemandeDePrestation, lireContrat, etc.

Expliquons le rôle de chacun des paramètres d'appel de cette opération.

- **Clef** de lecture : la clef de lecture contient les informations nécessaires à la lecture de l'objet : ce peut être une clef métier simple ou complexe, une clef primaire d'accès SQL, etc.
- **Contexte** de l'appel : ce paramètre contient l'ensemble des informations déjà obtenues par l'application qui fait appel à l'opération. Par exemple, le contexte contiendra le profil de l'utilisateur qui a initialisé la session d'utilisation de l'application. Ce profil peut être utilisé pour contrôler que l'application ou l'utilisateur ont bien le droit d'accéder au service concerné. Le chapitre 10 revient en détail sur ce concept de Contexte.
- **Scénario** de chargement : ce paramètre décrit le scénario de chargement des informations reliées à l'objet « racine ».

Pourquoi ce paramètre « scénario » ? Pour une raison fondamentale : un objet métier n'est jamais isolé, il est toujours relié à d'autres objets métier par un graphe de relations. Par exemple, un Client sera relié à son Contrat, à ses Demandes de Prestations, à ses Factures etc. Le Contrat est lui-même relié à l'historique des évolutions du Contrat. La Facture est reliée à un Paiement, etc.

En conséquence, lorsque le développeur fait appel à une opération de lecture d'un objet « racine », il doit préciser quels sont les objets reliés dont il a besoin. Dans certains cas, il n'aura besoin que de l'objet « racine » lui-même. Dans d'autres cas, il aura besoin de tout le graphe d'objets reliés. Par exemple, dans certains cas, il voudra uniquement lire les informations du Client (nom, adresse…), dans d'autres cas il voudra obtenir le Client, son Contrat et les dernières demandes de Prestation, dans d'autres cas encore il voudra le Client et ses Factures, etc.

Peut-on éviter d'utiliser un tel paramètre ? Par exemple, pourquoi ne pas « tout » lire d'un coup ? Pour des raisons de performance : le risque étant de ramener de fil en aiguille l'ensemble du référentiel en une seule malheureuse lecture !

A contrario, pourquoi ne pas s'en tenir à une lecture « paresseuse », c'est-à-dire à la lecture de l'objet « racine » lui-même, sans tenir compte de ses relations ? Tout simplement parce que dans beaucoup de cas une telle lecture n'est pas suffisante, il est nécessaire de lire certains objets reliés pour que l'application cliente puisse travailler. Or une stratégie purement « paresseuse » conduit à multiplier les appels aux opérations de lecture ce qui n'est pas non plus favorable aux performances : en utilisant le concept de « scénario de chargement », le développeur des opérations de lecture peut au contraire optimiser ce chargement.

### *Opération de recherche*

La signature générale d'une opération de recherche est la suivante :

```
rechercherListeXXX(ListeCriteres, Contexte)
```

- **listeCritères :** cette liste contient l'ensemble des critères de recherche qui vont permettre de restreindre la recherche dans le référentiel :
  - Un critère de recherche est un triplet [nomAttribut | opérateur logique | valeur]. Le nom de l'attribut désigne l'un des attributs décrivant l'objet métier concerné (exemple : la date de signature du contrat client).
  - L'opérateur logique permet de comparer l'attribut d'un objet métier à la valeur du critère. Par exemple, on recherchera tous les contrats qui ont été signés « avant » telle date, ou dans l'intervalle temporel [date1, date2], ou « après » telle date. « avant », « inclus dans », « après » sont autant d'opérateurs logiques. La liste des opérateurs logiques admissibles pour un attribut dépend en général du type de cet attribut.
- **Contexte** de l'appel : cet paramètre contient l'ensemble des informations déjà obtenues par l'application qui fait appel à l'opération (cf. chapitre 10) :
  - Le contexte contiendra par exemple le profil de l'utilisateur de l'application. Ce profil peut permettre de filtrer les informations accessibles à cet utilisateur : celui-ci n'aura par exemple accès qu'aux contrats de tel ou tel type, ou qu'aux clients habitant Marseille. C'est donc un critère de recherche supplémentaire, imposé par le système pour respecter une règle de confidentialité (l'application appelante ne peut pas y déroger).

Remarquons qu'il n'y a pas de paramètre « scénario » dans la signature de base d'une opération de recherche. L'appel par une application à une opération de recherche débouche ensuite sur l'affichage par cette même application d'une liste plus ou moins longue d'objets métier : l'objectif est d'optimiser l'affichage de cette liste, et il semble inutile de ramener « beaucoup » d'information sur chaque objet de la liste, donc a fortiori il paraît inutile de chercher des objets métier reliés à ces objets.

Cependant, un lecteur expérimenté objectera que certaines des informations affichées n'appartiennent pas nécessairement à l'objet métier lui-même, mais à un des objets qui lui sont reliés. Par exemple, si on veut afficher la liste des fournisseurs d'énergie, il peut être intéressant d'afficher pour chaque fournisseur le code postal de son siège social. Or ce code postal n'est pas directement un attribut de l'objet « fournisseur » mais un attribut de l'objet « adresse géographique » associé au fournisseur. Dans ce cas, un paramètre « scénario » peut s'avérer utile.

### 8.3.3 Services CRUD et référentiels métier

*Position du problème : partager ou répliquer ?*

Les services CRUD ont pour mission d'accéder aux référentiels de l'entreprise. Mais ces référentiels sont-ils centralisés, ou répliqués ? La question peut se poser de façon plus générale : les solutions métier partagent-elles le(s) même(s) référentiel(s) métier, ou pas ?

Si l'on reprend l'exemple « fil rouge », on voit que la solution à mettre en place gérera son propre référentiel « demande de prestation », mais elle aura besoin d'accéder (au moins en lecture) aux informations « Fournisseurs et contrats », « Points de Distribution », « Relevés de consommation ». Or ces informations existent probablement déjà dans des référentiels du Système d'Information, chaque référentiel étant géré par une application responsable de sa mise à jour.

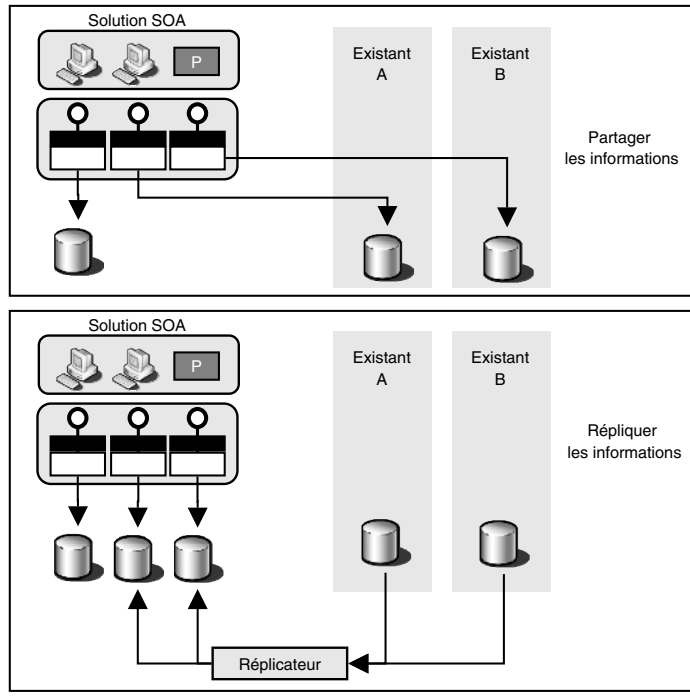



**Figure 8.5** — Partage ou réplication des référentiels métier

Si on admet que la solution de gestion des demandes accède à ces référentiels « maîtres » via les services appropriés, la conséquence immédiate est que la robustesse de cette solution sera celle du maillon le plus faible. Ce principe de partage des informations, conséquence a priori logique de l'approche SOA, est sain en théorie ; mais il risque d'être contradictoire avec une contrainte forte de fiabilité. Une solution de type e-Business doit le plus souvent satisfaire des contraintes de disponibilité, de type 24/24 – 7/7.

Dans ce type de situation, il sera donc nécessaire de dupliquer tout ou partie de certains référentiels. Cette décision de dupliquer les informations utilisées par la solution métier implique la mise en place d'une fonction de réplication. La figure 8.5 illustre les deux solutions possibles.

### *Répliquer ou agréger ?*

La partie 2 a mis en évidence que l'un des intérêts de SOA était de fournir aux applications une vision unifiée des informations métier. Prenons un exemple typique, fréquent dans les grands systèmes d'information : le problème de la « vision client ». L'objectif est de fournir aux acteurs commerciaux une vision unifiée de la situation d'un client vis-à-vis de l'entreprise (fiche d'identité, contrats en cours, historique des commandes, historiques des paiements, historique du support après-vente, etc.). Or cette situation est en général éclatée entre différents « silos » informatiques[1] : il est donc nécessaire de fournir une vision unifiée en agrégeant différentes informations issues de ces silos.

Cette vision peut être virtuelle : l'opération de lecture offerte par le service CRUD « vision client » va chercher à la volée les informations nécessaires dans les différents référentiels. Mais on peut également préférer, pour les raisons de robustesse

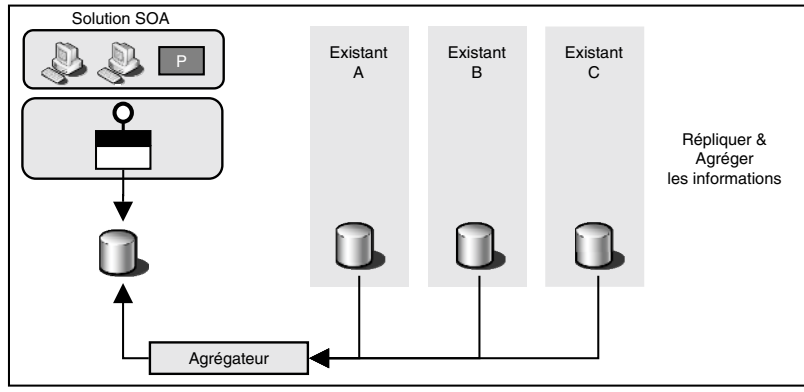



**Figure 8.6** — Réplication + agrégation des référentiels métier

1. Ces silos sont : le SI commercial (qui donne la vision contractuelle), le SI facturation (qui donne l'état des paiements du client vis à vis de la facturation), le SI Contentieux (qui dit si le client est ou a été en situation de contentieux), le SI Marketing (qui retrace l'historique des offres promotionnelles envoyées au client), etc.

évoquées précédemment, ou pour des raisons de performances, mettre en place un référentiel « vision client » dupliquant concrètement les informations nécessaires. La figure 8.6 illustre ce cas.

### *Avantages et inconvénients*

Répliquer une information implique que la solution SOA n'aura pas nécessairement accès à la toute dernière version de cette information, en fonction du mode et de la fréquence de réplication. Il existe en effet deux modes de réplication : en batch, ou bien « au fil de l'eau » (dès que l'information maître est modifiée, elle est répliquée).

Dans certains cas, le mode « batch » de réplication n'est pas acceptable sur le plan métier : cela peut être le cas des relevés de consommation s'ils sont mis à jour au fil de l'eau. Dans ce cas, les informations doivent être accédées en temps réel : donc la réplication devrait se faire « au fil de l'eau », ce qui peut être très coûteux (coût de l'outil sous-jacent, complexité du paramétrage de cet outil, baisse des performances du référentiel répliqué, etc.). La décision de répliquer une information suppose donc d'analyser la fréquence de mise à jour de cette information et d'étudier la fréquence possible de réplication.

A contrario, partager un référentiel implique que ce référentiel « central » verra sa charge de travail augmenter : la performance des accès doit être étroitement surveillée.

La question de la réplication des informations métier se pose donc dès le départ de la conception des services CRUD dans un contexte SOA.

La réponse dépend du compromis à faire entre un certain nombre de critères de décision, résumés par le tableau 8.1.

**Tableau 8.1** — Critères de décision

| | **Duplication des informations** | **Partage des informations** |
|---|---|---|
| Pro | > Robustesse de la solution SOA.<br>> Performance meilleure.<br>> La fonction de duplication peut être dotée de capacité de transformation du format des données, pour simplifier le développement des services CRUD. | > Les informations métier partagées sont toujours à jour.<br>> La solution SOA est plus simple à déployer. |
| Cons | > Les informations métier dupliquées ne sont pas nécessairement à jour (dépend de la fréquence de duplication) → certaines informations fortement transactionnelles ne peuvent être dupliquées.<br>> Si les données dupliquées sont mises à jour, nécessité de rétro-propager ces mises à jour vers la donnée de référence → complexité de la solution.<br>> La solution est plus coûteuse (prévoir la fonction de réplication/agrégation). | > La robustesse de la solution dépend de la robustesse du référentiel central.<br>> Les performances doivent être sous surveillance permanente (montée en charge de ce référentiel central).<br>> Les services CRUD doivent éventuellement effectuer des transformations de format pour satisfaire les besoins de la solution métier. |

Si on prend la décision de dupliquer les informations, alors il faut inclure une fonction de réplication ou réplication + agrégation de données dans le périmètre du projet SOA.

Cette fonction s'appuiera sur un outil : compte tenu de la diversité des besoins de réplication, l'offre des middlewares dédiés[1] est large, des outils de type ETL qui deviennent de plus en plus sophistiqués en matière de réplication, jusqu'aux outils de type MDM dont nous parlerons en partie 7 pour les aspects agrégation + réplication.

Si au contraire on prend la décision de partager les informations, alors il sera nécessaire de répondre à deux questions :

- Comment rendre les solutions métier aussi résistantes que possible à une panne d'un des référentiels, puis à sa remise en service ? Le minimum vital étant d'éviter qu'une telle panne n'entraîne un plantage brutal de la solution.
- Comment faire face à la montée en charge du référentiel ?

La première question implique une réflexion sur la conception des services CRUD d'accès au référentiel. Ces services doivent, avant d'accéder effectivement au référentiel, s'assurer que celui-ci est en fonctionnement.

### 8.3.4 Un service CRUD n'est pas un DAO !

Nous terminerons cette synthèse sur les services CRUD par une remarque importante : l'implémentation d'un service CRUD ne se réduit pas simplement à l'écriture d'une requête SQL et à la transformation du résultat de cette requête en un ensemble d'objets Java ou C# !

Prenons par exemple une opération de mise à jour du Compte Client. Comme l'illustre la figure 8.7, cette opération peut inclure :

- L'exécution de règles métier contrôlant l'intégrité des données mises à jour.
- La vérification préalable que les référentiels accédés fonctionnent correctement (pour faire face à une panne, cf. § précédent).
- L'appel au service de persistance pour mettre à jour la ou les tables du référentiel « compte client ».
- L'appel au service de persistance pour mettre à jour une table d'audit, permettant de tracer l'écriture, la date d'exécution, l'utilisateur concerné…
- Éventuellement, l'appel à un service technique pour l'envoi d'un message à un responsable commercial.

**Règle :** un service CRUD n'est pas simplement un DAO (*Data Access Object*)[a], mais bien un véritable Service Métier.

a. Le concept de DAO est un des patterns architecturaux proposés dans le cadre de J2EE.

1. Sans parler des solutions propriétaires offertes par chaque grand offreur de SGBD, ou des solutions de réplication physique liées à la mise en place d'un SAN.

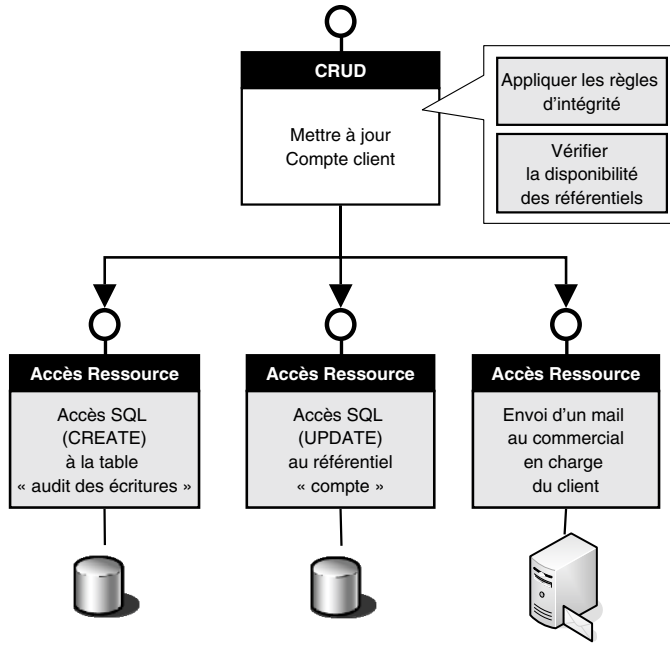



**Figure 8.7** — Un service CRUD est un vrai service métier !

## 8.4 ZOOM SUR LES SYSTÈMES EXISTANTS

L'accès aux systèmes existants est un thème récurrent de toute démarche SOA, puisque l'un des avantages de cette démarche est précisément de valoriser/réutiliser ces systèmes existants, en évitant une refonte « big bang » du système d'information le plus souvent irréaliste.

Les services métier mis en place doivent donc accéder aux informations et traitements implémentés par ces systèmes existants. Certains services métier ne seront même que de simples façades.

Un service « façade » est un intermédiaire qui permet à ses clients (applications ou services de haut niveau) réalisés en « nouvelles technologies » d'accéder à des fonctions métier existantes sans se soucier de l'hétérogénéité technologique.

Comment alors concevoir ces façades ?

**Règle :** il y a deux grands types de solution architecturale pour intégrer façades SOA et applications existantes : une solution dite « décentralisée » et une solution dite « centralisée ».

### 8.4.1 Intégration décentralisée

La solution décentralisée consiste à faire évoluer le système existant pour qu'il offre lui-même les façades, par exemple sous forme de web services. Les façades seront également appelées services légataires.

Dans ce type de solution, les services de haut niveau (applicatif ou fonctionnel) accèdent alors à ces services légataires de façon homogène, comme ils accèdent aux services CRUD, via les mêmes middlewares de communication (Web Service, EJB, ESB…).

Ce type de solution est intéressant (voire la seule envisageable) lorsque la solution SOA à mettre en place doit accéder à des sites géographiquement distants et disposant de leur propre informatique locale, comme c'est le cas des magasins, des agences, des usines ou encore des entrepôts de stockage. Cette informatique locale repose souvent sur des technologies propriétaires plus ou moins anciennes, difficiles d'accès par les technologies récentes associées à SOA. Comme ces applications locales sont souvent très efficaces, complexes et bien rôdées, il n'est pas question de les réécrire : d'où l'intérêt de les encapsuler via des services légataires « locaux ». Ces services légataires permettent par exemple de lancer un traitement local (lancement d'une fabrication, demande de réassort…) ou de récupérer des informations locales à ces sites (état d'avancement d'une commande, niveau de stock d'un produit…).

Les figures 8.8 et 8.9 présentent cette intégration décentralisée via des façades, selon que ces façades sont implémentées directement avec la technologie locale ou

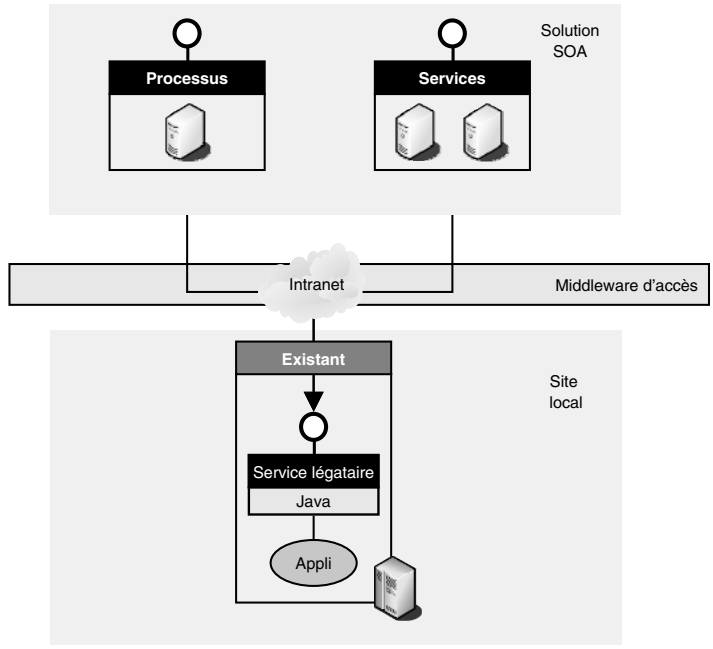



**Figure 8.8** — Intégration décentralisée/service légataire intégré sur le système local

qu'elles sont implémentées sur un serveur « frontal » également local. Le choix dépend des capacités de cette technologie locale à supporter ou non des services écrits dans un langage moderne (l'AS/400 permet par exemple de réaliser et d'héberger directement des façades Java).

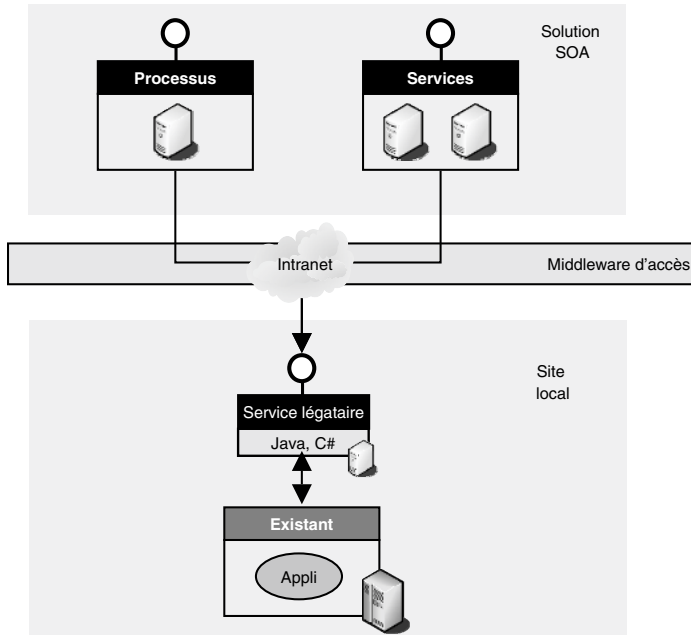



**Figure 8.9** — Intégration décentralisée/service légataire installé sur un frontal

Les avantages de cette solution décentralisée sont (i) de préserver un existant rôdé, (ii) de faire réaliser les services légataires par les informaticiens en charge de ces applications locales, et (iii) d'isoler totalement la solution métier SOA de toute adhérence avec les technologies « locales ».

Le premier point clef pour le succès d'une solution décentralisée réside d'abord dans le choix du bus de communication entre la solution métier SOA et les services légataires distants. Ce bus doit être compatible avec les technologies locales.

Un second point clef réside dans la nécessité de synchroniser fortement le projet de développement de la solution métier SOA d'une part, et le projet de développement des services légataires d'autre part.

### 8.4.2 Intégration centralisée

La solution centralisée consiste à installer les façades sur un ou des serveurs entourant le mainframe, et à encapsuler directement l'accès au(x) système(s) existant(s) dans ces façades.

Cette solution est intéressante lorsqu'il s'agit d'accéder à une application du système central (le fameux mainframe) déjà structurée sous forme de services, ou qu'il est facile de modifier pour qu'elle offre (dans sa technologie native) l'équivalent de services. Certains lecteurs constateront en effet que la démarche SOA leur rappelle « quelque chose » : il est indéniable que même dans le monde mainframe, pour ne pas parler du monde objet, des architectes et responsables d'application ont, comme Mr Jourdain pour la prose, adopté une démarche SOA avant l'heure. Dans ce cas, l'accès direct au(x) application(s) existante(s) en sera nettement facilité.

Dans une solution centralisée, les développeurs des façades SOA accèdent directement à un service technique d'accès au middleware de communication avec le(s) système(s) existant(s), que ce soit un EAI, un MOM (MQ Séries par exemple), un moniteur transactionnel (TUXEDO le plus souvent) ou une passerelle orientée mainframe IBM (passerelle CICS ou IMS).

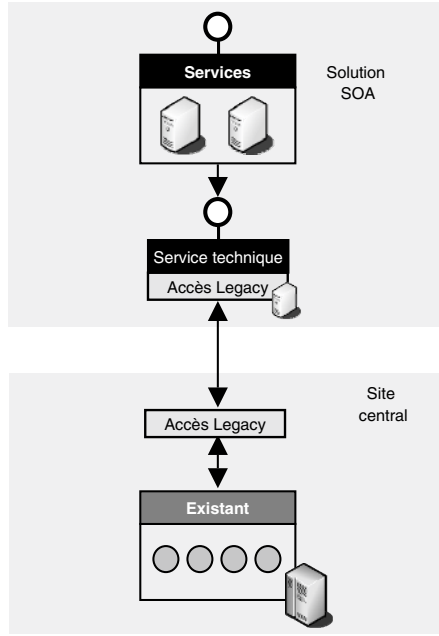



**Figure 8.10** — Intégration centralisée via un service technique

## 8.5 ZOOM SUR LES SERVICES DE HAUT NIVEAU

Spécifier et modéliser un service de haut niveau (Service Applicatif, Service Fonctionnel), c'est avant tout étudier comment un tel service fait appel à d'autres services – on parle également de composition de services.

Un service faisant appel à un autre service obéit en premier lieu à la règle de base de SOA : le service appelant utilise le contrat proposé par le service appelé,

sans rien savoir de son implémentation. Tant que le service appelé respecte ce contrat, l'implémentation de ce service appelé peut évoluer ad libitum (par exemple, pour optimiser les performances, enregistrer des traces, mieux sécuriser l'accès au service, etc.), sans que le service appelant n'ait à en tenir compte.

Édictons en second lieu quelques règles méthodologiques simples de composition, règles découlant de la typologie des services.

**Règle :** deux services métier de même niveau ne peuvent s'appeler entre eux, sauf exception.

**Règle :** un service métier utilise un/des services métier de niveau(x) inférieur(s) et un/des services techniques.

**Règle :** un service technique n'appelle pas un autre service technique, sauf exception (journalisation des opérations).

**Règle :** un service n'appelle pas un service d'un niveau supérieur. Les appels se font en cascade, de haut en bas de la hiérarchie.

**Règle :** un service doit être utilisé au moins une fois soit par une application composite, soit par un autre service, sinon son utilité est douteuse.

La composition de services jouant comme on le voit un rôle important pour construire d'autres services, il est important de se doter d'outils et de normes facilitant cette composition. C'est ce que l'on verra dans la partie 4 avec l'introduction à la norme SCA (*Services Composition Architecture*).

## En résumé

L'importance de la typologie et de la granularité des services ne saurait être sous estimée dans la réussite de la mise en place d'une Architecture Orientée Service.

Les différentes catégories de service présentées dans ce chapitre permettent de satisfaire les exigences de réutilisabilité, d'interopérabilité et d'orchestrabilité au sein des processus métier. Le tableau suivant résume notre propos :

| Type de service | Granularité | Réutilisabilité | Interopérabilité avec l'extérieur du SI | Orchestrabilité |
|---|---|---|---|---|
| Service métier Applicatif (SA) | Forte. | Sans objet : un SA est spécifique d'une application composite interactive, c'est-à-dire spécifique d'un besoin métier précis. | Forte : ce sont les services qui ont vocation à être accessible de l'extérieur du SI. | Sans objet. |
| Service métier Fonctionnel (SF) | Moyenne à Forte. | Moyenne (pour les services de type accès à des informations de synthèse) à forte (accès à des traitements de simulation tarifaire.). | Dépend du contexte. | Forte, puisque l'orchestrabilité est l'un des critères de modélisation de ces services. |
| Service métier CRUD | Faible. | Forte. | Sans objet : un service CRUD ne doit pas être directement accessible. | Faible. |
| Service Technique | Moyenne à Forte. | Forte. | Sans objet : un service technique ne doit pas être directement accessible. | Sans objet, ces services doivent être encapsulés dans des services fonctionnels. |
| Service Légataire | Forte. | Faible à Moyenne (dépend du contexte, notamment de la structure même du système existant accédé par le service légataire). | Sans objet, dans la mesure où un système existant n'a pas été conçu pour être ouvert sur l'extérieur via Internet. | Faible à Moyenne (attention aux performances). |

# 9

# Modéliser les processus

## Objectif

L'objectif de ce chapitre est de préciser ce qu'on entend par processus métier dans un contexte SOA. Le concept de processus métier est encore plus vieux que le concept de workflow, et il est indispensable d'en préciser la sémantique dans ce contexte.

La clef d'une démarche SOA centrée sur les processus réside dans l'obtention d'une « bonne » modélisation des processus. Comme pour les services, les étapes successives de modélisation des processus métier sont mises en évidence en prenant appui sur notre exemple « fil rouge ». Le chapitre montre en quoi la vision métier des processus doit impérativement être complétée par une vision technique de ces processus, et dégage les principes de modélisation qu'il est nécessaire de respecter.

Centrer le système d'information sur les processus ne nécessite pas seulement de bien modéliser ces processus. Il est également nécessaire de modéliser l'architecture de déploiement des processus, en mettant en évidence l'impact sur l'utilisateur final.

## 9.1 QU'EST-CE QU'UN PROCESSUS SOA ?

### 9.1.1 Bref rappel sur les processus métier

Un processus métier décrit l'ensemble des activités que l'entreprise doit exécuter pour traiter un événement métier qui lui est adressé. Un événement métier peut par

exemple être l'envoi d'une commande, une demande de prestation de service, une demande de simulation de crédit, l'envoi d'une facture, etc.

Le processus métier est défini par :

- **L'événement métier** « déclencheur », qui est à l'origine du processus, mais également les événements métier que le processus échange avec le monde extérieur (par exemple : événement « envoi d'une lettre au client » réclamant des informations complémentaires pour traiter la demande, et événement « réception de la réponse du client »).
- **Les activités** qui doivent être exécutées, et les règles conditionnant le passage d'une activité à une autre.
- Pour chaque activité, et en fonction des choix d'organisation de l'entreprise, **l'acteur** (ou le groupe d'acteur) qui doit exécuter l'activité[1].

Il est indispensable de bien différencier modèle de processus et processus (ou instance de processus). Un modèle de processus décrit de façon générique comment l'entreprise réagit à un type d'événement métier; une instance de processus associée à ce modèle traite un événement métier individuel. Par exemple : le processus « traitement de la demande de prestation n° XYZ en date du 5 mars 2007 émise par la société Martin » sera associé au modèle de processus décrivant comment l'entreprise gère un événement métier « traitement d'une demande de prestation ».

Le concept de processus métier est étroitement associé au concept de Gestion des Processus Métier (ou BPM : *Business Process Management*). L'idée fondamentale étant que le système d'information soit « paramétré » par les modèles de processus : la description d'un processus n'est plus enfouie dans le logiciel, mais externalisée sous forme de modèle, et il existe un outil spécifique, le moteur BPM, qui exécutera les instances de processus associées au(x) modèle(s). L'avantage clef obtenu est de pouvoir faire évoluer facilement les processus sans toucher aux applications associées.

### 9.1.2 Processus Métier et contexte SOA

Depuis le début des années 90, l'approche workflow a considéré essentiellement les processus métier sous un angle humain : un processus est vu dans cette approche comme orchestrant une suite d'interventions humaines pour traiter l'événement métier déclencheur.[2]

L'approche SOA a pris en compte dès le départ les processus métier, en privilégiant une approche entièrement automatisée : un processus est vu comme un « chef d'orchestre » enchaînant une suite d'appel à des services métier. On parle d'ailleurs communément d'orchestration de processus. Le concept d'activité est assimilé dans ce cas à un appel de service métier, et il n'y a plus a priori d'« homme dans la boucle ».

1. Soulignons d'ailleurs qu'une activité peut être plus ou moins décomposée afin de pouvoir être attribuée à plusieurs acteurs différents.
2. La littérature américaine parle de « human-driven business process management ».

Le succès de SOA vient en grande partie de cette prise en compte de ce qu'on pourrait appeler les « processus e-business » : un processus e-business est un processus automatisé, créé suite à une interaction entre une entreprise et son écosystème via le Web. L'expérience montre que les directions informatiques convainquent leur direction générale de financer le tournant SOA par l'intérêt de la mise en place de tels processus e-business.

L'intérêt est tel qu'il a suscité l'émergence de normes et outils dédiés. Un modèle de processus SOA, c'est-à-dire la description de l'orchestration du processus à partir de services métiers, sera ainsi décrite grâce à un langage dédié à cet usage et standardisé par les acteurs du monde SOA : BPEL (*Business Process Execution Language*) est désormais reconnu sur le marché pour l'exécution d'une telle orchestration. BPEL, étant une grammaire XML, il reste illisible pour une maîtrise d'ouvrage, un formalisme métier est également envisagé. Pour la phase d'analyse, de pilotage et d'optimisation, à ce jour, seul BPMN[1] (*Business Process Modeling Notation*) propose une base graphique normalisée mais encore incomplète. En outre ne sont pas encore pris en compte la modélisation d'aspects métiers traditionnels tels que les coûts, les temps, les risques, les indicateurs, etc.

Un processus associé à un tel modèle sera créé et exécuté par un « orchestrateur BPEL », qui joue le rôle d'un moteur de workflow classique.

## 9.2 MODÉLISER LES PROCESSUS : LES ÉTAPES CLEFS

Le chapitre présente les 3 étapes principales de la modélisation d'un processus métier de type e-Business : (1) exprimer le besoin métier, (2) analyser et compléter le processus obtenu et enfin (3) concevoir un modèle de processus exécutable par un moteur BPEL.

### 9.2.1 Première étape : l'expression du besoin

La première étape de modélisation décrit le processus avec le point de vue métier de la maîtrise d'ouvrage. Le modèle résultant a en général deux caractéristiques :

- il décompose très finement le processus métier;
- il liste les exceptions métier levées par le processus.

Dans le contexte du « fil rouge », le modèle MOA[2] du processus de traitement d'une demande de prestation est présenté dans la figure 9.1.

1. BPMN est désormais sous l'égide de l'OMG (*Object Management Group*) pour venir enrichir UML et l'initiative BPDM et permettre à terme une correspondance notamment avec XPDL visant à l'interchangeabilité des représentations entre outils de modélisation (promu lui par le WfMC).
2. MOA = Maîtrise d'OuvrAge, c'est-à-dire l'ensemble des utilisateurs et sponsors métier.

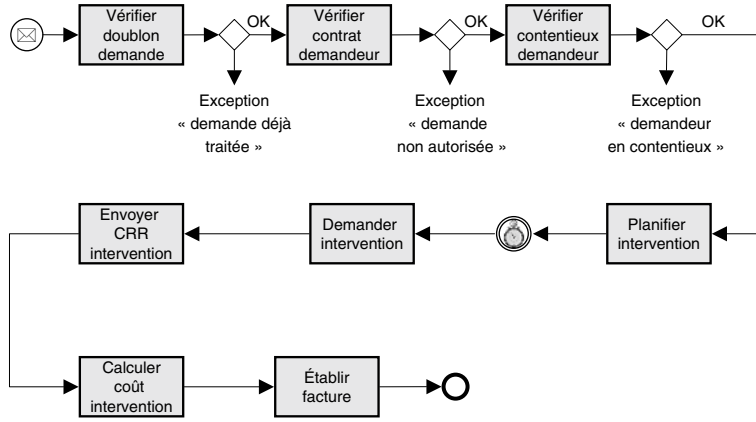



**Figure 9.1** — Modèle de Processus/vision « MOA »

Le processus débute par l'analyse de la demande de prestation émise : est-elle recevable par l'entreprise eu égard à différents critères métier ? Il se poursuit par l'envoi d'une demande de planification au système de planification des interventions (on fait ici l'hypothèse simplificatrice qu'à une prestation correspond une et une seule intervention sur le terrain, c'est-à-dire une intervention sur un Point De Distribution d'énergie). Le système de planification renvoie une date d'intervention ainsi que l'identité de l'équipe d'intervention. Le moment venu, le processus lance l'intervention (via un mail à l'équipe), envoie un compte-rendu d'intervention au demandeur, calcule la facture et l'émet vers le demandeur.

Un tel modèle permet à la MOE de bien comprendre les objectifs poursuivis par la MOA. Cependant, d'un point de vue technique, ce modèle n'est pas directement exploitable pour les raisons suivantes :

- Le modèle est découpé trop finement : exécuté tel quel par un moteur BPEL, il conduirait à multiplier l'appel à des services de grain fin, ce qui induit des problèmes de performances.
- Le modèle liste les exceptions métier, mais ne précise pas comment ces exceptions sont traitées.
- Il existe d'autres exceptions, d'ordre technique, qui ne sont pas mises en évidence : par exemple, que se passe-t-il si le système de planification ne répond pas ? Que se passe-t-il si l'intervention n'est plus possible à la date planifiée pour une raison quelconque, ou si la messagerie électronique est hors service ?

Le problème de la granularité du modèle de processus est directement lié au problème de la granularité des services, déjà évoqué dans le précédent chapitre. Il faut insister sur ce point, en édictant le principe suivant :

> La vision « MOA » d'un processus SOA ne fait pas directement émerger les services à orchestrer : pour résoudre ce problème, il est nécessaire de regrouper les activités du processus pour faire apparaître les services fonctionnels.

La gestion des exceptions pose un double problème. D'une part, il est nécessaire d'avoir une vision exhaustive des causes possibles de dysfonctionnement du processus, et pour cela ne pas hésiter à appliquer la loi de Murphy (« tout ce qui peut poser problème posera problème un jour ou l'autre »).

D'autre part, il est nécessaire de savoir qui traite les exceptions dans un processus automatisé, et comment. Répondre à cette question, c'est réfléchir à « la place de l'homme dans le système », et mettre en évidence les composants complémentaires (services, applications composites) permettant de traiter ces exceptions.

En bref, cette première étape conduit à une deuxième étape qui doit préciser le périmètre de la solution métier incluant le processus, en définissant (1) les services réellement orchestrés par le processus (services fonctionnels, services techniques) et (2) les applications composites permettant de traiter les exceptions.

### 9.2.2 Deuxième étape : l'analyse du processus et l'impact sur le périmètre de la solution métier

La deuxième étape vise donc à analyser le processus métier pour préciser le périmètre de la solution métier associée à ce processus. Cette analyse doit satisfaire aux exigences de granularité des services et d'exhaustivité de la gestion des exceptions. Ce modèle est obtenu en appliquant les principes suivants :

- regrouper certaines activités du processus « vision MOA », pour minimiser le nombre de services appelés, et faire émerger les services fonctionnels;
- décrire succinctement la façon dont les exceptions sont traitées, en interaction avec la maîtrise d'ouvrage qui statue en dernier ressort.

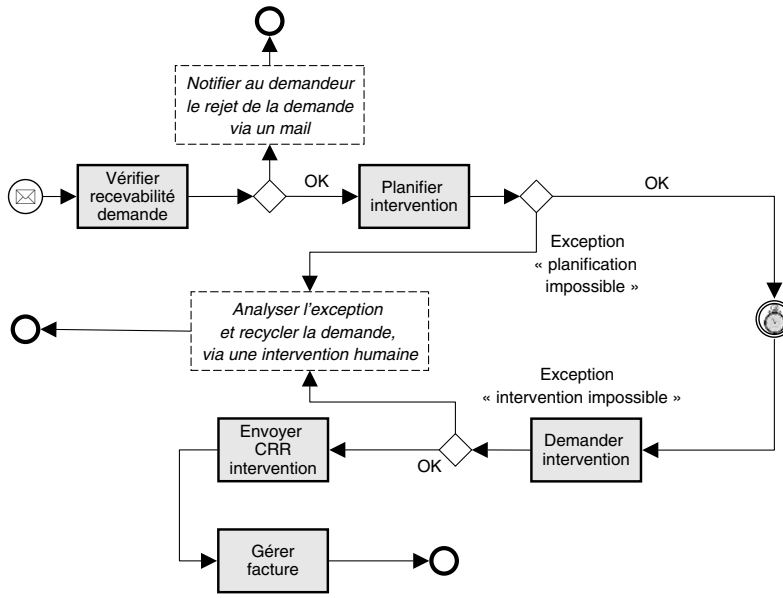



**Figure 9.2** — Modèle de Processus/vision « MOE »

La figure 9.2 décrit le résultat obtenu pour le modèle de processus du « fil rouge ».

On remarquera que l'on peut gérer les exceptions de deux façons :

- si l'exception résulte de l'application d'une règle de gestion par un service appelé par le processus, elle peut être traitée automatiquement (exemple : si le demandeur est en contentieux car il n'a pas réglé ses précédentes factures, alors le processus envoie une lettre ou un mail notifiant le rejet de la demande);
- si l'exception résulte d'un problème extérieur au processus (exemple : un système extérieur, sollicité, ne répond pas dans les délais), elle ne peut en général être traitée que par une intervention humaine.

Le traitement des exceptions par une intervention humaine conduit donc au concept de recyclage, qui se traduit par le scénario suivant :

- le processus métier lève une exception et signale cette exception à un acteur humain;
- l'acteur humain étudie l'exception : il peut décider de clore la demande, ou bien il intervient pour débloquer cette demande;
- le déblocage d'une demande entraîne automatiquement la création d'un processus dit de recyclage. Ce processus de recyclage peut ou non réutiliser certains services orchestrés par le processus « normal ».

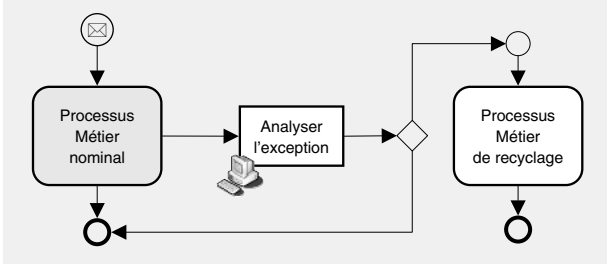



**Figure 9.3** — Concept de recyclage

La figure 9.3 met en évidence la nécessité de modéliser non seulement le processus métier en lui-même, mais également les moyens de traiter certaines exceptions, faute de quoi la solution métier déployée ne sera pas complète.

### 9.2.3 Troisième étape : la conception d'un processus exécutable

La troisième étape de modélisation vise à produire un modèle de processus exécutable par le moteur de processus[1]. Ce modèle est obtenu en appliquant les principes suivants :

- introduire dans le modèle les services de gestion des exceptions;

1. Plus exactement, cette étape vise à produire un modèle qui, traduit dans le formalisme d'exécution (BPEL dans le contexte SOA), sera (i) fonctionnellement complet et (ii) techniquement exécutable avec les performances attendues.

- introduire dans le modèle les exceptions de type « time out », c'est-à-dire les temps d'attente de la réponse d'un système externe (système de planification par exemple), et la façon de traiter les exceptions.

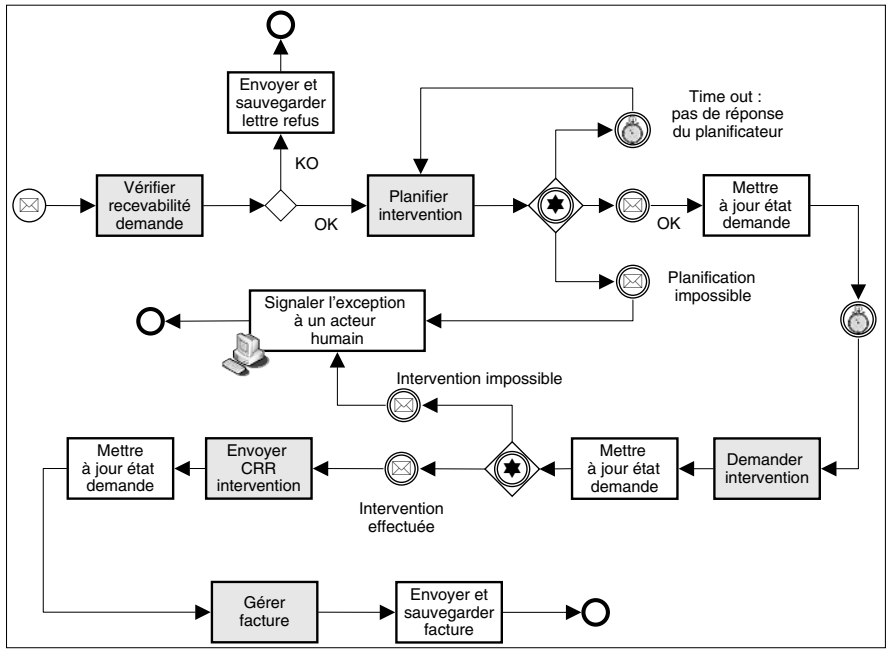



**Figure 9.4** — Modèle de processus exécutable par un moteur d'orchestration

Même à ce stade de conception du processus, le dialogue avec la maîtrise d'ouvrage est plus que jamais indispensable, car certaines décisions, apparemment techniques, peuvent la concerner très directement, comme le précisent les paragraphes suivants.

### *La gestion des time-out*

Premier exemple de ce dialogue nécessaire entre MOA et MOE : la gestion des time-out. Admettons par exemple que l'opérateur de réseau sous-traite certaines interventions sur le terrain à une société de service : dans ce cas, le système de planification n'est plus géré dans le SI de l'opérateur, mais dans celui de la société de service. Le processus métier reste identique, mais cette société de service doit s'engager à ce que son système de planification fournisse une réponse en moins de x secondes, sous peine de pénalité. Ce type de clause fait partie intégrante du contrat de sous-traitance, via le SLA à négocier.

Plus généralement, lorsqu'un processus métier fait appel à un service offert par un système extérieur, le contrat de service doit inclure un SLA sur la réactivité de ce service (cf. partie 2).

### *La gestion de l'état du processus*

Second exemple du dialogue avec la maîtrise d'ouvrage : le modèle proposé par la figure 9.4 introduit des étapes de mise à jour de l'état de la demande. En effet, lorsqu'un processus métier dure longtemps, il est en général impératif, sur le plan marketing, de permettre au demandeur de poser via le portail la question « où en est le traitement de ma demande ? ». Cela implique de modéliser les différents états de cette demande. Un tel modèle est présenté par le diagramme d'état UML (simplifié) de la figure 9.5.

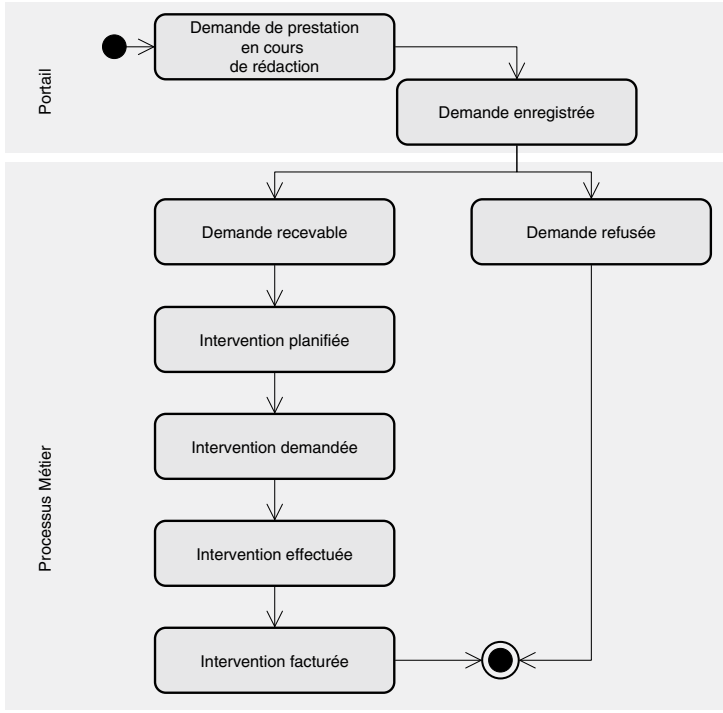



**Figure 9.5** — Diagramme d'états de la demande de prestation

Il sera donc nécessaire d'ajouter dans le processus un ou des appels à un service CRUD de mise à jour de l'état de la demande. Cet ajout peut être direct (on ajoute directement au modèle de processus les étapes de mise à jour de la demande, comme l'illustre notre exemple) ou indirect (on ajoute ces mises à jour dans les services fonctionnels).

Le lecteur attentif objectera à juste titre que l'ajout direct dans le modèle de processus des appels à un service CRUD (donc à grain fin) n'est pas conforme à l'objectif général de granularité « gros grain » des services orchestrés par le moteur de Processus. Mais positionner les appels à ce service dans les services fonctionnels risque de rendre ces services fonctionnels non réutilisables par d'autres applications composites. Cette décision est donc à prendre au cas par cas.

### 9.2.4 Conclusion : résumé méthodologique

L'approche méthodologique proposée pour modéliser un processus métier SOA est résumée par les principes suivants.

Tout processus repose sur l'orchestration de services métiers dont la granularité ne doit pas être trop fine (cf. la distinction orchestration de processus et orchestration de services décrite précédemment).

Un processus sans exception n'existe pas : le modélisateur doit, pour chaque activité du processus, se poser la question des exceptions possibles.

Un processus entièrement automatisé n'existe pas : la gestion des exceptions implique en général de « mettre l'homme dans la boucle ».

Un processus métier peut en cacher un autre (par exemple, un processus de recyclage des exceptions).

Si un processus fait appel à un service externe à la solution métier dont il fait partie, le fournisseur de ce service doit s'engager formellement sur la qualité de service (performances notamment).

Si un processus risque de durer longtemps (à l'échelle humaine), il peut être nécessaire de mettre en place : (a) au sein du processus, une mise à jour de l'état du processus après chaque activité effectuée et (b) une application composite complémentaire permettant au demandeur d'interroger l'état du processus.

## 9.3 MODÉLISER UNE ARCHITECTURE BPM DANS UN CADRE SOA

L'intervention d'un acteur humain dans un processus e-Business est un problème clef, qui conditionne clairement la validité de l'approche SOA dans ce domaine.

Sur le plan métier, cette intervention comme on l'a vu est le plus souvent indispensable, car il n'est pas possible d'automatiser tous les traitements, en particulier le traitement des exceptions.

Sur le plan informatique, se pose la question de la redondance entre workflow « traditionnel », et gestion de processus SOA. Autrement dit, peut-on utiliser une infrastructure SOA comme unique outil de gestion de processus, que ces processus soient « *human driven* » ou « e-business » c'est-à-dire automatisés ? Or le standard SOA actuel (BPEL) souffre d'un péché originel : la non prise en compte de la dimension humaine[1].

Ce qui suit propose un modèle de solution BPM complétant l'infrastructure BPEL pour qu'elle puisse prendre en compte les interventions humaines dans des processus

1. Cf. la partie 4 pour une présentation de BPEL sur le plan technique.

métier. Il s'agit donc de modéliser les principaux composants de cette solution et de montrer les interactions entre ces composants.

On remarquera au passage que certaines offres BPEL du marché (Oracle, IBM, BEA, etc.) commencent à offrir des composants prenant en compte la dimension humaine : ce qui suit peut être aussi utilisé comme guide de comparaison de ces offres.

Les composants à modéliser doivent permettre de répondre à deux questions clefs :

- Comment l'infrastructure SOA connaît-elle l'acteur qui doit intervenir sur le processus ?
- Comment l'acteur concerné sait-il qu'il a quelque chose à faire ?

### 9.3.1 Comment déterminer l'acteur devant intervenir sur un processus ?

Pour répondre à cette question, l'infrastructure BPM SOA (c'est-à-dire l'orchestrateur de service) fait appel à un service de routage. Ce service de routage s'appuie sur un service d'organisation et un service de gestion de l'état des processus, comme l'illustre la figure 9.6.

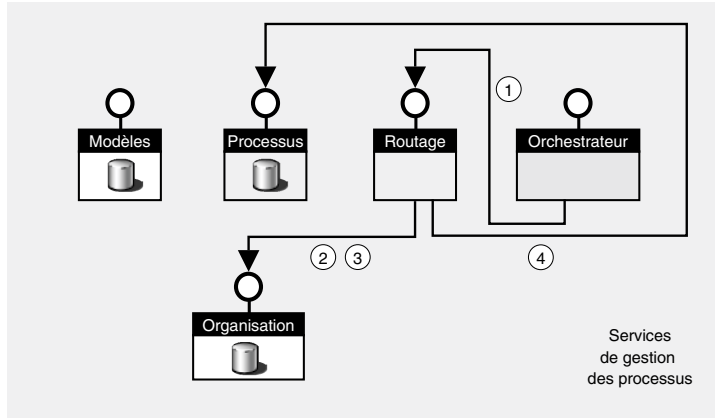



**Figure 9.6** — Router le processus

#### *Le service de routage*

Ce service a pour objectif de déterminer quel acteur doit traiter l'exception ou, plus généralement, quel acteur doit intervenir sur le processus. On parle d'aiguiller ou de « router » le processus vers l'acteur devant intervenir, d'où le nom du service.

Tout d'abord, on remarquera que, le plus souvent, le routage s'effectuera non pas vers un acteur nominatif, mais vers un groupe d'acteurs ayant le même profil et réunis dans une même cellule de travail (une cellule pouvant être : une agence commerciale, une direction régionale, une cellule de gestion au sein du siège de l'entreprise, etc.).

Pour effectuer ce routage, le service de routage s'appuie sur deux concepts fondamentaux permettant de répartir le travail au sein d'une organisation :

- Le concept de portefeuille de processus.
- Le concept de portefeuille d'objets métier.

Le portefeuille de processus d'une cellule de gestion est l'ensemble des modèles de processus sur lesquels la cellule a le droit d'intervenir. Par exemple, toute cellule « agence commerciale départementale » peut intervenir sur un processus de demande de prestation et notamment en traiter les exceptions.

Le portefeuille d'objets métier est l'ensemble des objets métier (ou plus exactement des instances métier) sur lesquels la cellule a le droit d'intervenir. Par exemple, la cellule « agence commerciale du département "59" » peut intervenir sur tout ce qui concerne les « clients » localisés dans ce département. On utilisera souvent des portefeuilles de « clients » (notion classique dans le monde de la gestion), mais on peut également avoir des portefeuilles « contrats » par exemple.

Pour router un processus instancié sur le modèle de processus X et traitant un événement métier initié par le client Y, le service de routage procède en trois temps, comme le montre la figure 9.6 :

- le service de routage calcule l'intersection entre l'ensemble des cellules pouvant intervenir sur le modèle de processus X, et l'ensemble des cellules ayant Y dans leur portefeuille client. Pour cela, il interroge le service Organisation pour avoir les informations nécessaires (étape 2 de la figure) ;
- lorsque le service de routage a déterminé la ou les cellules susceptibles d'intervenir, le service détermine le ou les acteurs ayant le profil pour intervenir. Pour cela, il interroge à nouveau le service Organisation (étape 3 de la figure) ;
- enfin, le service de routage met à jour le référentiel gérant l'état des processus en associant au processus l'identité du ou des acteurs pouvant intervenir (étape 4 de la figure).

Par exemple, seuls les acteurs de profil « consultant commercial senior » et appartenant à la cellule « agence commerciale départementale "59" » pourront intervenir sur une exception associée à une demande de prestation émise par un client de ce département.

### *Le service d'organisation*

Les informations nécessaires au fonctionnement du routage sont gérées par un référentiel dédié, le référentiel Organisation. Le service Organisation est le service de type CRUD permettant d'interroger et de mettre à jour ce référentiel Organisation.

Les informations gérées via ce référentiel sont en premier lieu les informations classiques : acteurs, hiérarchie organisationnelle des cellules, profils des acteurs. Le référentiel contient également pour chaque cellule le portefeuille d'activité et le portefeuille client/contrat gérés par cette cellule.

Le référentiel peut également contenir d'autres types d'information, permettant de gérer un routage plus précis, comme par exemple les informations sur la disponibilité des acteurs (acteur en vacances, acteur en arrêt maladie, intérimaire, stagiaire, etc.).

#### *Le service de gestion des processus*

Le référentiel gérant l'état des processus permet de savoir, pour une instance de processus donnée, dans quel état est ce processus, et, si le processus doit faire intervenir un acteur humain, quel(s) est(sont) le (ou les) acteur(s) concerné(s).

Le service de gestion des processus est le service de type CRUD permettant d'interroger et de mettre à jour ce référentiel Processus.

### 9.3.2 Comment un acteur sait-il qu'il doit intervenir ?

L'infrastructure BPM SOA met à disposition de chaque acteur humain une application composite appelée Corbeille, traduction libre de Worklist souvent utilisé dans le monde anglo-saxon (on parlera aussi d'Agenda, ou de ToDoList).

La fonction première du composant corbeille est d'interroger le service de gestion des processus pour connaître les processus sur lesquels l'acteur doit intervenir. La figure 9.7 illustre ce fonctionnement.

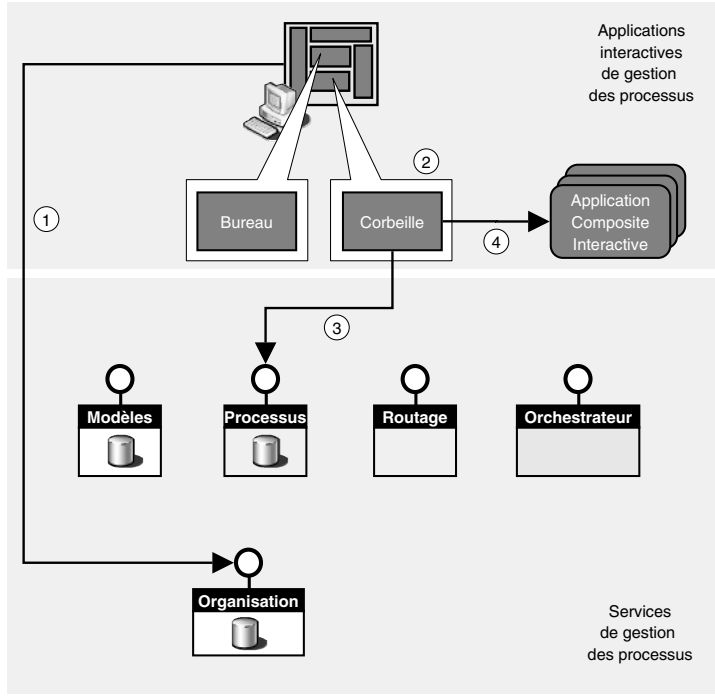



**Figure 9.7** — Intervenir sur un processus

Après avoir récupéré le profil de l'acteur associé (étape 1 de la figure), le poste de travail affiche la corbeille dédiée à cet acteur (étape 2). La corbeille utilise ensuite ce profil pour récupérer et afficher la liste des (instances de) processus sur lesquels l'acteur doit intervenir : elle accède pour cela au service de gestion de l'état des processus (étape 3). En sélectionnant un processus dans cette liste, l'acteur invoque ipso facto l'application qui permet d'intervenir sur ce processus (étape 4).

**Tableau 9.1** — Les différents types de corbeille

| Type de corbeille | Acteur utilisant ce type de corbeille | Principales fonctions de ce type de corbeille |
|---|---|---|
| Corbeille individuelle | Tout acteur appartenant à l'entreprise (par exemple : gestionnaire, responsable commercial, télé-acteur, etc.). | > Afficher la liste des activités affectées nominativement à l'acteur (activités à traiter, activités en cours de traitement).<br>> Offrir des fonctions de tri et de filtre sur la liste (par exemple : afficher uniquement les activités concernant un client X). |
| Corbeille de groupe | Acteurs appartenant à une même cellule de gestion. Ils auront donc deux corbeilles à leur disposition : corbeille individuelle et corbeille de groupe. | > Afficher la liste des activités affectées à la cellule de l'acteur concerné.<br>> Autoriser un acteur à s'auto-affecter une activité non encore affectée nominativement. Une activité affectée nominativement disparaît de la corbeille de groupe.<br>> Offrir des fonctions de tri et de filtre. |
| Corbeille de Supervision | Responsable d'une cellule de Gestion ou d'un ensemble de cellule (service, département…). | > Afficher la liste des activités affectées à la cellule ou aux cellules supervisées par le responsable.<br>> Autoriser le responsable à changer l'affectation d'une activité.<br>> Offrir des fonctions de tri et de filtre.<br>> Offrir une fonction d'historique.<br>> Permettre la mise en place d'alerte (une alerte est un filtre spécial permettant de détecter qu'une activité n'a pas été traitée avant un temps T, T paramétrable). |
| Corbeille simplifiée | Acteur externe à l'entreprise (par exemple, client émettant des demandes vers l'entreprise). | > Afficher la liste des événements métier émis par l'acteur externe.<br>> Permettre à l'acteur de demander l'état du processus traitant un des événements métier de la liste. |

Ce fonctionnement est hélas trop simple pour être utile sur le terrain : l'expérience montre que les besoins des différents acteurs concernés impliquent d'aller nettement plus loin en matière de fonctionnalité de la corbeille. Plusieurs questions se posent en effet :

- Comme on l'a vu, un processus peut être routé non pas vers un acteur mais vers une cellule : dans ce cas, comment se passe la répartition du travail entre les acteurs de la cellule ?
- La liste des instances de processus peut être fort longue : comment l'acteur peut-il organiser son travail via sa corbeille ?
- Un responsable de cellule doit être capable de superviser le travail de sa cellule : comment peut-il faire ?

On retiendra que répondre à ces questions implique de mettre en place une typologie de corbeilles, en fonction du type d'acteur concerné. Le tableau 9.1 esquisse une telle typologie.

## 9.4 LA SOLUTION MÉTIER REVISITÉE

La solution métier décrite dans le chapitre 7 voit donc son périmètre s'élargir notablement, en lui ajoutant les composants suivants :

- l'application composite d'affichage et de correction des exceptions;
- le processus de recyclage des demandes corrigées (et éventuellement les services fonctionnels associés);
- les services de gestion de « l'homme dans la boucle » : service de routage, service d'organisation;
- les référentiels associés : base organisation, base des instances de processus;
- les applications composites « corbeille individuelle » et « corbeille simplifiée » (et, si besoin, les applications « corbeille de groupe » et « corbeille de supervision »).

Il est important de noter que certains de ces composants sont génériques, c'est-à-dire réutilisables dans n'importe quel processus métier SOA : il s'agit des services et application (corbeilles) orientés « l'homme dans la boucle ». Une fois développés pour une solution métier, ces composants seront donc fortement réutilisables pour une autre solution métier. L'investissement à consentir sera donc amorti rapidement.

### En résumé

L'un des intérêts de l'approche SOA repose sur la possibilité de mettre en place des processus métier de type « e-business » : c'est en tout cas un avantage clef aux yeux des directions métier, qui permet de justifier les investissements nécessaires à une telle approche.

La modélisation de ces processus repose sur un dialogue étroit entre maîtrise d'ouvrage et maîtrise d'œuvre. Ce dialogue doit être guidé par les quelques questions fondamentales que pose cette modélisation : granularité des services, traitement des exceptions métier, traitement des exceptions techniques et qualité des services, etc.

Cette modélisation conduit à s'interroger sur la place fondamentale des acteurs humains dans ces processus automatisés. La mise en place de services et d'applications « human driven » complétant l'infrastructure SOA (BPEL) permet de répondre efficacement à cette interrogation.

# 10

# Modéliser les applications composites interactives

## Objectif

L'objectif de ce chapitre est de fournir des pistes concrètes pour la modélisation des applications composites interactives.

La phase d'expression de besoin de ces applications interactives repose sur les outils classiques que sont la description des cas d'utilisation et le maquettage d'écran. Le cadre UML, et les méthodes associées à UML, telle que UP (*Unified Process*) – c'est-à-dire les outils utilisés jusqu'alors pour les développements d'objets logiciels – ce cadre convient pour cette phase.

L'impact de SOA sur la modélisation des applications interactives concerne donc principalement les phases d'analyse et surtout de conception de ces applications interactives.

Depuis 1980, le modèle MVC (Modèle – Vue – Contrôleur) est le modèle de référence : l'objectif principal du chapitre est donc de revisiter ce modèle, après en avoir rappelé les caractéristiques fondamentales. On introduit notamment le concept de Contexte local à l'application composite.

Le chapitre introduit également le concept de Transaction Longue, en expliquant en quoi ce concept a non seulement une dimension technique mais également une dimension métier.

# 10.1 LE MODÈLE MVC

Depuis les années 80, le modèle MVC est le modèle architectural de référence pour concevoir des applications graphiques interactives. Ce modèle distingue trois grands types de composants :

- le Contrôleur : ce composant reçoit les demandes en provenance de l'utilisateur final et contrôle le comportement de l'application interactive pour répondre à ces demandes;
- le Modèle Métier : ce composant contient la logique et les informations métier nécessaires pour répondre aux demandes de l'utilisateur;
- la Vue : ce composant affiche les informations attendues par l'utilisateur final.

Ce modèle d'architecture est très lié à l'approche orientée Objet : le Contrôleur et la Vue sont des objets techniques, le Modèle Métier étant vu comme un ensemble d'Objets Métier activés par le Contrôleur.

La figure 10.1 illustre la dynamique générale de cette architecture.

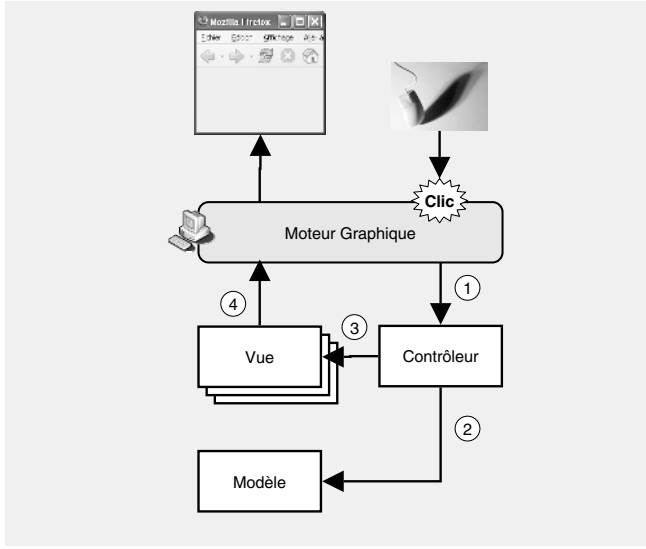



**Figure 10.1** — Le modèle MVC classique

Cette dynamique se décompose en 4 grandes étapes :

- étape 1 : via un moyen d'interaction quelconque (clavier, souris, voix…) géré par le moteur graphique, l'utilisateur envoie une demande d'action au Contrôleur;
- étape 2 : le Contrôleur reçoit cette demande, l'interprète et en déduit le ou les actions à demander au Modèle Métier. Le Modèle Métier est activé, et il doit renvoyer un objet « résultat »;

- étape 3 : une fois le résultat produit, le Contrôleur choisit la Vue qui doit afficher ce résultat à l'utilisateur final ;
- étape 4 : une fois activée, la Vue élabore le flux graphique (page HTML, écran Windows, animation Flash, etc.) à partir des informations de l'objet « résultat », puis envoie ce flux vers le moteur graphique (navigateur Web, Windows, interpréteur Flash, etc.) – ledit moteur devra interpréter ce flux pour afficher la page ou la fenêtre graphique.

## 10.2 SOA : LE MODÈLE MVC REVISITÉ

L'impact le plus évident de l'approche SOA sur cette architecture MVC concerne le Modèle. Celui-ci n'est plus local à l'application, mais encapsulé sous forme de services, comme on l'a vu dans les chapitres précédents.

De ce fait, le Contrôleur n'active plus directement ce Modèle Métier, mais fait appel à un ou plusieurs Services. Le Contrôleur devient ainsi fondamentalement un Coordinateur d'appel de services.

La figure 10.2 illustre la dynamique générale de l'architecture MVC revisitée.

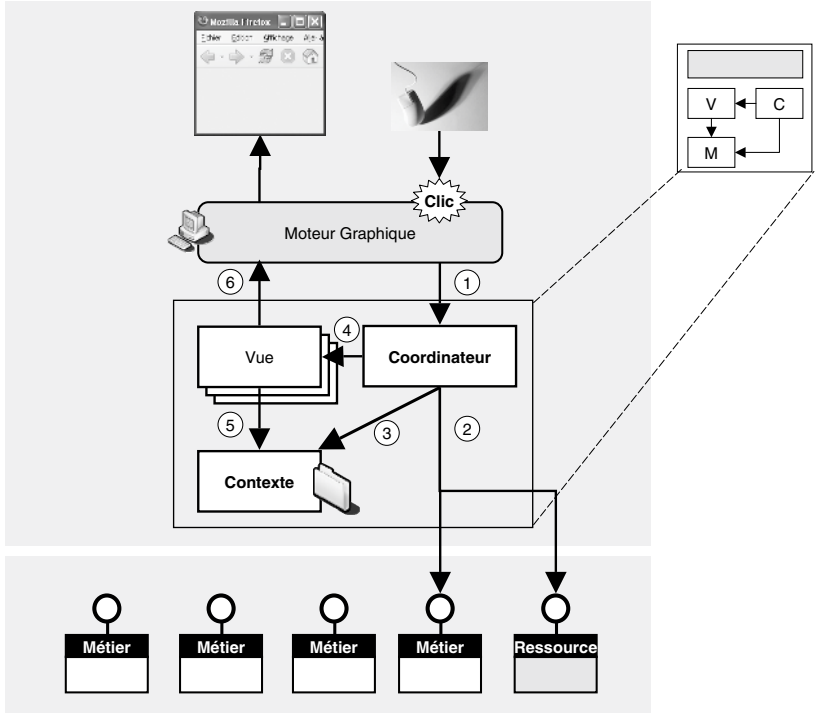



**Figure 10.2** — Le modèle MVC revisité dans un contexte SOA

Cette dynamique se décompose en 6 grandes étapes (en gras, les étapes qui apparaissent ou qui évoluent par rapport au modèle MVC classique) :

- étape 1 : l'utilisateur final envoie une demande à l'application;
- **étape 2 :** le Coordinateur reçoit cette demande, l'interprète et en déduit le ou les services métier et/ou techniques à appeler;
- **étape 3 :** au fur et à mesure de l'appel de chaque service, le Coordinateur récupère des informations, qu'il va stocker provisoirement dans un Contexte local pour élaborer le résultat à afficher. Le Contexte a pu également intervenir lors de l'étape 2, en fournissant au Coordinateur des paramètres d'appel des services;
- étape 4 : une fois le résultat obtenu par appel au(x) service(s) et stocké dans le Contexte, le Coordinateur choisit la Vue qui doit afficher ce résultat à l'utilisateur final;
- **étape 5 :** la Vue récupère dans le Contexte les informations à afficher;
- étape 6 : la Vue élabore le flux graphique à renvoyer vers le moteur graphique.

La suite du chapitre revient sur trois questions fondamentales de cette dynamique :

(1) Comment gérer un Contexte local à une application composite ?

(2) Quelle la conséquence de l'existence de plusieurs contextes locaux en parallèle pouvant accéder au même objet métier ?

(3) Quelle est l'architecture de ce que nous avons appelé le Coordinateur dans le modèle MVC/SOA ?

Le chapitre conclut sur ce que pourrait devenir une Vue dans ce contexte SOA.

### 10.2.1 Le concept de Contexte

#### *Pourquoi ce concept de Contexte ?*

Lorsqu'il utilise une application composite, l'utilisateur final entame une véritable session de travail. Dans cette session, l'utilisateur n'accède pas (en général) à un seul objet métier isolé, mais il navigue dans un graphe d'objets métier.

Par exemple, dans le « fil rouge », lorsque l'utilisateur final active l'application composite « consulter et rectifier les Relevés de Consommation », il souhaite naviguer du Fournisseur aux Clients desservis par ce fournisseur, d'un Client à ses Sites de fourniture[1], d'un Site aux Points de Distribution desservis sur ce site, d'un Point de Distribution à l'historique des Relevés de Consommation, etc. La figure 10.3 présente un exemple de session de travail.

1. On suppose ici que le Client est une entreprise qui possède plusieurs Sites géographiques (usines, entrepôts…), et que chaque Site peut posséder plusieurs Points De Distribution (donc plusieurs compteurs) correspondant par exemple à plusieurs types d'énergie fournie.

Le **Contexte** est le composant dans lequel l'application composite va stocker pendant la durée de la session de travail les objets métier dont elle a besoin pour répondre aux demandes de l'utilisateur associé à cette session. Il y a donc une instance de Contexte par Session de travail.

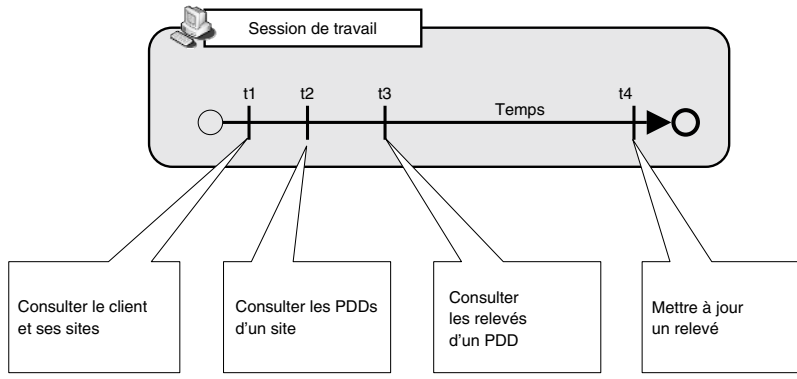



**Figure 10.3** — Session de travail

Le Contexte est l'équivalent de l'objet `HttpSession` proposé par J2EE, si l'on veut, mais c'est un concept d'abord architectural, indépendant de toute technologie, donc utilisable aussi bien en architecture client léger que client lourd[1].

On notera que ce concept de Contexte s'applique aussi bien aussi aux applications composites interactives, qu'aux processus métier évoqués au chapitre précédent. Dans le cas des Processus Métier, on parlera également de gestion de Dossier (le concept de Dossier est identique au concept de Contexte ici décrit).

### *Contexte et lecture des informations métier*

Il est important de noter que le remplissage, ou chargement, du Contexte avec les informations métier, peut se faire selon deux grands types de stratégie :

- stratégie par anticipation : l'application charge en une seule fois l'ensemble des informations métier nécessaires dans le contexte, c'est-à-dire qu'elle appelle le service CRUD pertinent avec un scénario de chargement « en profondeur » (cf. chapitre 8, § « zoom sur les services CRUD »);
- stratégie « paresseuse » : l'application charge une information dans le contexte, uniquement quand l'utilisateur demande cette information. Cela revient à charger les informations une par une.

1. De plus, et sans rentrer dans le détail technique, si on souhaite consulter en parallèle les informations de deux clients A et B, via deux onglets du navigateur, la `httpSession` sera unique, alors que l'application devra créer deux contextes bien distincts, l'un dédié au client A et l'autre au client B.

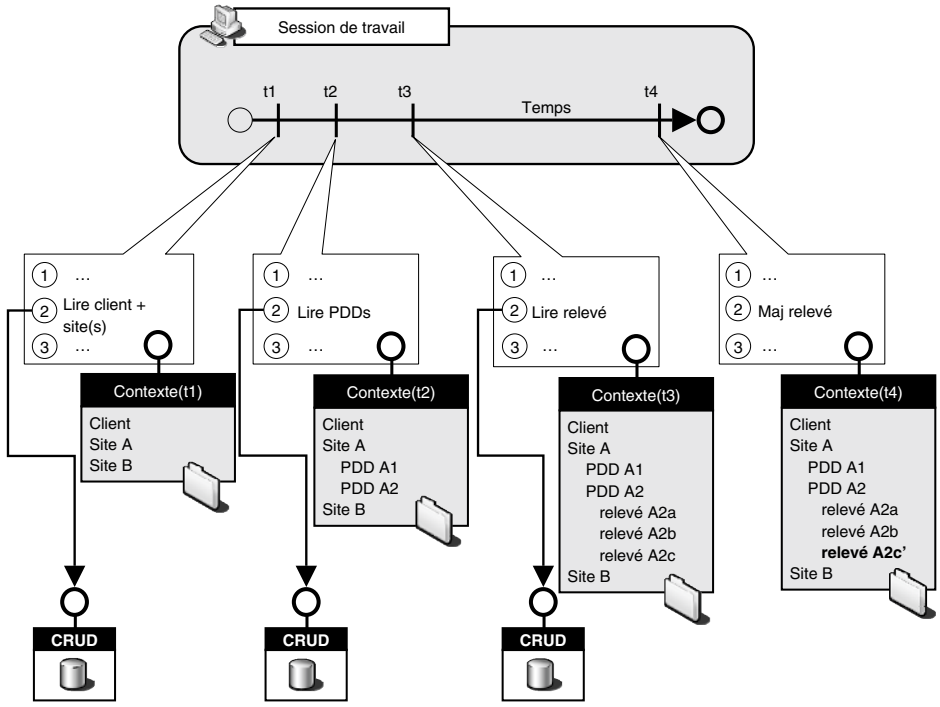



**Figure 10.4** — Session de travail, contexte, chargement paresseux

La figure 10.4 présente l'évolution du contexte dans la session de travail présentée plus haut, avec l'application d'une stratégie « paresseuse ».

Le choix de la stratégie à adopter dépend des besoins métier, car chaque stratégie a ses avantages et ses inconvénients.

La stratégie par anticipation autorise une navigation fluide entre les informations, puisque ces informations sont déjà contenues dans le contexte, qui joue ici un rôle de cache. Mais l'anticipation présente également des inconvénients indéniables : certaines informations seront inutiles (c'est-à-dire jamais utilisées pendant la session de travail de l'utilisateur), et surtout le temps de chargement peut être prohibitif – l'application risque en effet de souffrir du syndrome « j'ai besoin d'un objet métier racine, et en fait je charge toute la base via les relations entre objets métier ».

La stratégie « paresseuse » multiplie les chargements : l'avantage principal est que chaque chargement individuel est plus performant que dans la stratégie précédente, notamment le premier chargement, mais la navigation sera moins fluide du point de vue de l'utilisateur final, surtout si les services CRUD d'accès aux informations sont des web services distribués.

Cette analyse du choix de stratégie de chargement conforte la mise en place de la notion de scénario de chargement, introduite au chapitre « modéliser les services » (Paragraphe « zoom sur les services CRUD »). Cette notion permet en effet de moduler le choix « tout anticiper ou rien anticiper », en fonction du besoin métier.

La stratégie « paresseuse » pose de plus un problème de gestion des doublons dans le Contexte. Pour expliquer ce problème de doublon, on s'appuie sur l'exemple du graphe d'objets métier (diagramme de classe UML) de la figure 10.5.

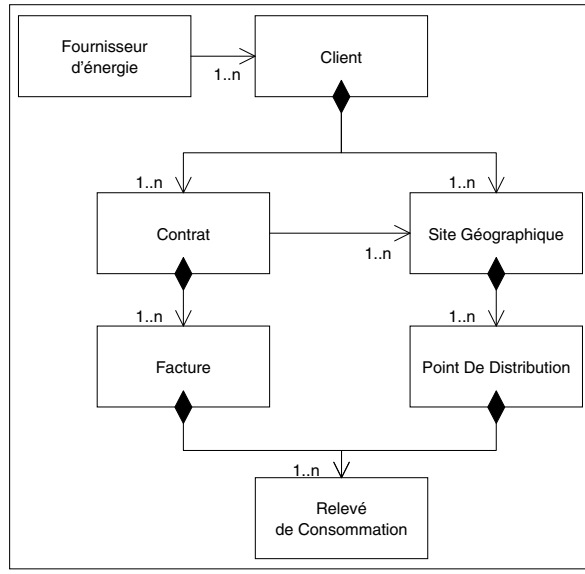



**Figure 10.5** — Modèle métier et gestion de doublon

Admettons que l'application charge d'abord un objet de la classe « client » et les objets « site », « PDD » et « Relevé » qui lui sont rattachés, puis que l'application effectue un chargement complémentaire avec les objets « contrats », « factures », et les objets qui leur sont rattachés.

Sans précaution particulière, les services de chargement vont créer des « doublons » de chaque objet « relevé », c'est-à-dire deux instances de la classe « relevé » (une par chargement) correspondant à la même information, alors qu'il ne devrait y avoir qu'une instance. Le graphe d'objet est alors incohérent, et posera des problèmes notamment lors de la sauvegarde[1].

Une solution à ce problème générique est de déléguer au Contexte le soin de détecter l'apparition de doublons (par comparaison des identités métier ou des identités techniques).

### *Contexte et invocation des services métier*

La question abordée ici porte sur l'utilisation du Contexte comme paramètre d'appel d'un service métier.

1. Le moteur de mapping objet/relationnel refusera de sauvegarder deux objets correspondant à la même ligne d'une table SQL.

Pourquoi, en poussant le raisonnement, ne pas considérer que le Contexte est le seul et unique paramètre d'appel de tous les services métier, en entrée et en sortie ? D'abord parce que cette simplification radicale se ferait au détriment de la robustesse de l'invocation d'un service. En effet, les outils sous-jacents (middleware web service, par exemple) ne pourraient plus contrôler le typage des paramètres d'entrée, alors que cette capacité de contrôle est un des atouts de ces outils.

Ensuite, parce que l'utilisateur d'un service, c'est-à-dire le concepteur de l'application composite, ne peut plus, à la seule lecture du contrat de service, comprendre ce que le service va produire comme résultat.

Cette simplification se ferait donc au détriment de la réutilisation et de la robustesse des services métier, et est donc à proscrire.

### *Contexte et sauvegarde des informations métier*

Lors d'une session de travail, l'utilisateur ne fait pas (en général) que consulter des informations métier : il les met à jour, ou il en crée de nouvelles. Comme l'utilisateur doit avoir un « droit à l'erreur », c'est-à-dire la possibilité de revenir en arrière ou de modifier plusieurs fois la même information dans la même session de travail, les mises à jour qu'il effectue ne seront pas immédiatement répercutées vers le ou les référentiels concernés. L'application attendra la fin de session[1] pour sauvegarder ces mises à jour ou création.

La conception de cette mécanique de sauvegarde du Contexte n'est pas triviale. Cette mécanique devra tout d'abord parcourir un ou plusieurs graphes d'objet métier, plus ou moins complexes (c'est-à-dire un graphe avec des relations u-naires et n-aires entre les objets). À chaque nœud du graphe, elle devra décider si le (ou les) objet(s) correspondant(s) doi(ven)t être sauvegardé(s) ou non (ce qui correspond soit au fait que l'objet a été modifié, soit au fait qu'il vient d'être créé). Enfin, la mécanique devra faire appel, à chaque nœud à sauvegarder, à un service CRUD d'écriture dans un référentiel, en gérant les aspects transactionnels.

Pourquoi compliquer cette mécanique de sauvegarde en faisant la différence entre objet à sauvegarder et objet qu'il n'est pas utile de sauvegarder ? Pourquoi ne pas faire de sauvegarde du Contexte « en bloc » ? Réponse : pour se prémunir contre la situation décrite par le tableau 10.1.

Supposons que travaillent simultanément deux utilisateurs avec la même application composite, et sur les mêmes objets métier (le même client, etc.), et supposons que la mécanique de sauvegarde travaille « en bloc » : le tableau 10.1 illustre les différentes étapes du travail de chacun des utilisateurs.

1. Détecter la fin d'une session de travail est un problème classique dans le cas des sites Internet. C'est pourquoi il sera toujours demandé explicitement à l'utilisateur de signaler cette fin de session. L'application ne peut pas décider seule (via un time out) de sauvegarder les informations saisies par cet utilisateur.

**Tableau 10.1** — Inconvénient de la sauvegarde « en bloc » du Contexte

| étape | Travail de l'utilisateur U1 | Travail de l'utilisateur U2 |
|---|---|---|
| 1 | U1 charge [a] dans son Contexte les objets O1 « pour consultation » et O2 « pour modification ». O1 est dans une version initiale v1, nous l'écrirons $O1_{v1}$. | |
| 2 | | U2 charge dans son Contexte l'objet $O1_{v1}$ « pour modification ». |
| 3 | | U2 modifie l'objet O1, qui devient l'objet $O1_{v2}$. |
| 4 | | U2 sauvegarde via le Contexte l'objet $O1_{v2}$. |
| 5 | U1 modifie O2, puis sauvegarde « en bloc » son Contexte : c'est la version $O1_{v1}$ qui est sauvegardée, donc il y a perte du travail de l'utilisateur 2 ! | |

a. « U1 charge O1… » est un raccourci commode pour « U1 demande à son exemplaire de l'application d'afficher O1, et pour cela l'application (i) interroge le référentiel concerné pour récupérer les données (SQL ou autre) nécessaires, (ii) crée une instance de l'objet O1, (iii) remplit cette instance avec ces données (utilisation des `get` et des `set`), et (iv) met cette nouvelle instance dans le Contexte dédié à U1 ».

### *Vers le concept de « Service de Gestion de Contexte »*

Tout ce qui précède montre que le Contexte n'est pas un « simple » container d'objets métier : on peut considérer le Contexte comme un véritable service, proposant à une Application Composite le contrat de service contenant les opérations suivantes :

- créer un nouveau Contexte ;
- écrire un objet dans un Contexte – cette opération détecte et évite les doublons ;
- récupérer un objet présent dans un Contexte ;
- sauvegarder un Contexte – cette opération ne sauvegarde dans le ou les référentiels concernés que les objets modifiés ou nouvellement créés dans ce Contexte, et prend en charge l'aspect transactionnel de l'écriture dans ces référentiels ;
- supprimer un Contexte.

Certaines architectures SOA vont plus loin dans l'utilisation du Contexte : il est en effet possible de faire du Contexte un intermédiaire entre d'une part les Applications Composites, et d'autre part les services CRUD d'accès au(x) référentiel(s). L'idée de base d'une telle architecture est qu'une Application Composite délègue à son Contexte le soin de récupérer les informations métier et d'appliquer la stratégie de chargement adéquate. L'application Composite se contente ensuite d'interroger le Contexte pour récupérer les informations dont elle a besoin (pour afficher un écran ou pour appeler un service métier de calcul), le Contexte effectuant si besoin

des chargements complémentaires, c'est-à-dire invoquant si besoin les services CRUD associés.

L'avantage principal de cette orientation est que le concepteur/développeur de toute Application Composite n'a plus à se préoccuper de l'appel des services CRUD et du remplissage du Contexte ; il se contente d'utiliser les services offerts par le Contexte. L'inconvénient est évidemment de complexifier la mise en place du service de gestion de Contexte.

#### *Conclusion : vers la normalisation du Contexte ?*

En conclusion, cette vision du Contexte en tant que service à part entière est partagée par la communauté SOA, qui de ce fait a entrepris récemment un effort de standardisation autour des normes WS-CAF (*Composite Application Framework*), et plus particulièrement de la norme WS-Context (cf. partie 4). Cet effort est cependant loin d'être achevé car un tel standard implique de proposer des réponses « universelles » aux questions non triviales décrites ici.

### 10.2.2 Concept de transaction longue SOA

#### *Pourquoi ce concept de transaction longue ?*

La discussion sur la sauvegarde du Contexte a mis en évidence une question importante pour la modélisation d'une application composite : que se passe-t-il quand deux utilisateurs finaux travaillent sur le même objet métier (sur le même client, sur la même facture…) ?

L'utilisateur, lorsqu'il entame une session de travail – et crée donc implicitement un Contexte de travail –, va en effet manipuler les informations de ce Contexte pendant un laps de temps « long », par opposition à la durée de vie d'une transaction atomique de mise à jour d'un référentiel. On parle de « transaction longue », parce que la Session de travail se compte en minutes, en dizaine de minutes, voire en heures.

Ce terme de « transaction longue » a donc été adopté pour bien faire la différence avec ce qu'on nomme usuellement sous le terme de Transaction. Une Transaction, que l'on devrait appeler Transaction « Courte », doit respecter les critères de base ACID : la transaction doit être Atomique, Cohérente, Isolée et Durable. Or deux Sessions de Travail ne peuvent plus être considérées comme isolées l'une de l'autre, car l'échelle de temps est ici l'échelle humaine et non plus l'échelle de temps du micro processeur sous-jacent (cf. partie 1).

En bref, il y a un risque bien réel de collision entre deux utilisateurs. Ce problème n'est pas apparu avec l'émergence de SOA, mais l'adoption d'une démarche SOA fera immanquablement apparaître cette problématique.

#### *Les problèmes à résoudre*

Il faut d'abord se prémunir contre le cas de figure où les deux sessions de travail modifient effectivement le même objet métier. Cas de figure illustré par le tableau 10.2.

**Tableau 10.2** — Conflit entre deux écrivains

| Étape | Travail de l'utilisateur U1 | Travail de l'utilisateur U2 |
|---|---|---|
| 1 | U1 charge dans son Contexte l'objet O1 « pour modification ». O1 est dans une version initiale v1, nous l'écrirons $O1_{v1}$. | |
| 2 | | U2 charge dans son Contexte l'objet $O1_{v1}$ « pour modification ». |
| 3 | | U2 modifie l'objet O1, qui devient l'objet $O1_{v2}$. |
| 4 | | U2 sauvegarde via le Contexte l'objet $O1_{v2}$. |
| 5 | U1 modifie O1, qui devient $O1_{v3}$, puis sauvegarde son Contexte : c'est la version $O1_{v3}$ qui est sauvegardée, donc il y a perte du travail de l'utilisateur 2 ! | |

Mais ce n'est pas suffisant : il faut aussi se prémunir contre le cas de figure où une des sessions de travail se contente d'utiliser un objet métier, sans nécessairement le modifier. Cas de figure illustré par le tableau 10.3.

**Tableau 10.3** — Conflit entre un écrivain et un lecteur

| Étape | Travail de l'utilisateur U1 | Travail de l'utilisateur U2 |
|---|---|---|
| 1 | U1 charge dans son Contexte l'objet O1 « pour consultation ». O1 est dans une version initiale v1, nous l'écrirons $O1_{v1}$. | |
| 2 | | U2 charge dans son Contexte l'objet $O1_{v1}$ « pour modification ». |
| 3 | U1 travaille sur les objets de son Contexte. Ce travail peut dépendre de la valeur de $O1_{v1}$. | U2 modifie l'objet O1, qui devient l'objet $O1_{v2}$. |
| 4 | | U2 sauvegarde via le Contexte l'objet $O1_{v2}$. |
| 5 | U1 continue à travailler mais il ne sait pas que O1 est modifié : son travail peut en devenir incohérent ! | |

Comment alors se prémunir contre ces risques ?

### *Une solution possible : le verrouillage « métier »*

Une première solution possible repose sur l'utilisation d'un verrouillage « optimiste »[1]. Mais ce type de solution ne protège pas contre le cas de figure présenté par le tableau 10.3.

Pour protéger complètement le travail d'un utilisateur contre les effets d'un travail parallèle d'un autre utilisateur, un mécanisme de verrouillage « métier » doit être mis en place. L'objectif de cette solution est simple : permettre à une application composite de prévenir un de ses utilisateurs qu'un autre utilisateur est déjà en train de travailler (en lecture ou en écriture) sur un objet métier.

Une application composite, lorsqu'elle va charger dans un Contexte un Objet Métier, pose un verrou « métier » sur cet objet métier. Ce verrou peut être un verrou « pour Consultation » (l'application ne modifiera pas cet objet métier), ou un verrou « pour Modification ».

Une autre session de travail, souhaitant également charger le même Objet Métier, vérifiera d'abord systématiquement la présence ou non d'un verrou sur cet objet. S'il n'y a pas de verrou, alors le chargement peut avoir lieu.

Ce verrou est géré comme un véritable objet métier, avec ses attributs (objet verrouillé, identité de l'utilisateur posant le verrou, date de verrouillage, type de verrou…), et son référentiel dédié de persistance.

### *Verrouillage métier et dialogue avec la Maîtrise d'Ouvrage*

Mais que se passe-t-il si une application composite détecte la présence d'un verrou sur l'objet métier qu'elle souhaite charger ? Doit-elle bloquer la session de travail, c'est-à-dire empêcher l'utilisateur d'aller plus loin (si on souhaite une priorité à l'intégrité des données) ? Doit-elle simplement alerter l'utilisateur, sans le bloquer ?

Il n'y a pas de réponse « universelle » à cette question, seule la maîtrise d'ouvrage sera à même de répondre, en fournissant un tableau tel que le tableau 10.8, proposé à titre indicatif.

On notera que sur le plan technique, la mise en place de cette solution de verrouillage ne repose sur aucun mécanisme technique de verrouillage de bas niveau, lié par exemple à la Base de Données Relationnelles. En cas de plantage du système d'information, la gestion du déverrouillage peut se faire de façon très simple, en supprimant l'ensemble des verrous présents dans le référentiel dédié.

---

1. Le verrouillage optimiste repose sur l'association à tout objet métier d'une date de dernière mise à jour. Lorsqu'une application charge dans son contexte un objet métier, elle charge également cette date. Au moment de sauvegarder l'objet métier, l'application relit la date. Si la valeur obtenue à cet instant diffère de la première valeur, cela veut dire qu'une autre session de travail a entre temps modifié en parallèle cet objet métier, et qu'il y a conflit. Le verrouillage est dit « optimiste » car on suppose que les conflits sont rares. Le problème est que si on n'a pas besoin de sauvegarder l'objet parce que celui-ci n'est pas modifié, il n'y a pas de vérification de la date de mise à jour.

**Tableau 10.4** — Résolution des conflits de verrouillage

| Étape | Il existe déjà un verrou déjà posé « pour Consultation » par l'utilisateur U2 (ou par le processus P2[a]) | Il existe déjà un verrou déjà posé « pour Modification » par l'utilisateur U2 (ou par le processus P2) |
|---|---|---|
| L'application veut poser un verrou « pour Consultation » sur un objet métier = l'utilisateur U1 veut uniquement consulter cet objet métier. | A priori, c'est un cas passant : deux utilisateurs ne faisant que consulter un objet métier, peuvent travailler en parallèle sur cet objet. Cela peut cependant impliquer de devoir gérer plusieurs verrous « pour Consultation » sur le même objet métier (relation 1.N entre 1 objet métier et ses verrous), afin de pouvoir alerter tous les utilisateurs consultant cet objet lorsqu'un verrou « en écriture » sera posé (cas ci-dessous). | L'application doit au minimum alerter l'utilisateur voulant lire l'information : « attention ! Mr U2 (ou : le processus P2) travaille déjà sur cette information métier et risque de modifier l'information en parallèle ». |
| L'application veut poser un verrou « pour Modification » sur un objet métier = l'utilisateur U1 veut probablement modifier cet objet métier. | L'application doit au minimum alerter l'utilisateur risquant de modifier l'information « attention ! Mr U2 travaille déjà sur cette information, vous risquez de gêner son travail ». Si l'application ne bloque pas l'utilisateur U1, elle doit alors tracer l'action de U1 et/ou envoyer un mail à l'utilisateur U2. | L'application doit au minimum alerter l'utilisateur risquant de modifier l'information « attention ! Mr U2 risque également de modifier cette information, vous risquez de perdre vos travaux respectifs ». Si l'application ne bloque pas l'utilisateur U1, elle doit cependant tracer l'action de U1 et/ou envoyer un mail à l'utilisateur U2. |

a. Le service de verrouillage peut être utilisé par une application Composite Interactive ou par un Processus Métier.

### *Conclusion : vers un service de gestion des verrous*

La nécessité de mettre en place une gestion de verrou métier dépend de l'analyse des risques de collision entre utilisateurs, analyse à mener avec la maîtrise d'ouvrage.

La vision SOA du Système d'Information conduit à augmenter ce risque, puisque les référentiels sont de plus en plus intégrés et accédés en temps réel. La mise en place d'un service technique de gestion des verrous métier peut donc s'avérer indispensable.

Ce service offre aux applications composites le contrat défini par les opérations suivantes :

- `isLocked(businessObject)` : cette opération permet de savoir si un verrou est déjà posé sur l'objet métier.

- `getLocks(businessObject)` : cette opération permet de récupérer l'ensemble des verrous posés sur un objet métier.
- `removeLock(user, businessObject, lockMode)` : cette opération permet de supprimer un verrou.
- `setLock(user, businessObject, lockMode)` : cette opération permet de poser un verrou du type défini sur l'objet métier.

### 10.2.3 Architecture du coordinateur d'une application composite SOA

Il est possible de préciser le comportement d'un coordinateur dans une architecture tenant compte des concepts de Contexte et de Transaction longue. La figure 10.6 détaille l'architecture du coordinateur du modèle MVC revisité.

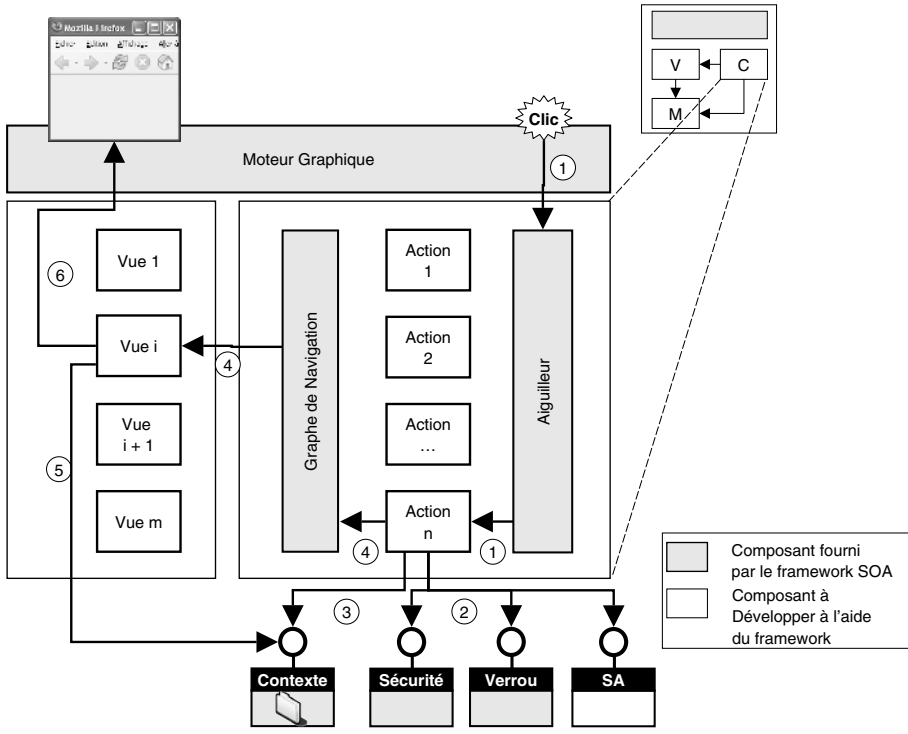



**Figure 10.6** — Architecture d'un coordinateur d'application composite

Le composant Aiguilleur récupère la demande émise par l'utilisateur et récupère si besoin les informations transmises dans la demande via un formulaire.

Un composant Action, une fois sélectionné puis activé par le composant Aiguilleur, a un comportement relativement générique : (i) il contrôle le droit de l'utilisateur à utiliser cette action et/ou à accéder aux objets métier concernés (accès au service de sécurité) (ii) il verrouille si besoin un objet métier (accès au service de ver-

rouillage métier) (iii) il active le ou les Services Applicatifs nécessaires (iv) il met à jour le contexte (accès au service de gestion de contexte).

Les composants Aiguilleur et Gestionnaire du graphe de navigation sont fournis par un framework, baptisé CAF (*Composite Application Framework*) dans la littérature.

**Règle :** mettre en place un framework CAF, regroupant les outils (composants génériques MVC) et services (verrou, contexte) permettant de développer les applications interactives.

### 10.2.4 Et la Vue ?

Chaque vue affiche un ou plusieurs objets métier contenus dans le Contexte. La vue devient un assemblage de panneaux, chaque panneau est associé à un objet métier. Une tendance actuelle est de modéliser une vue à l'aide d'un formalisme XML. Ce formalisme permet de :

- lister les différents panneaux;
- pour chaque panneau, décrire les composants graphiques à afficher. Dans le cas d'un formulaire classique, il s'agira essentiellement de champs de saisie;
- lister les activateurs affichés par la vue (activateur = bouton, hyperlien, menu et item de menu…);
- pour chaque champ de saisie, spécifier le lien entre le champ et l'attribut de l'objet métier concerné;
- pour chaque activateur, spécifier l'action à appeler.

Une fois modélisée, cette description peut être soit interprétée par un plug-in de navigateur ou un moteur dédié (attention aux problèmes de performance), soit utilisée pour générer le code de la Vue dans une cible appropriée (client léger ou client lourd). Ceci fera également partie du framework CAF.

On notera que de tels formalismes existent déjà (on citera XFORMS du W3C, XUL de la fondation MOZILLA, XAML de MICROSOFT).

## En résumé

La mise en place d'une démarche de modélisation d'une architecture SOA ne se réduit pas à l'émergence d'une bibliothèque de services réutilisables : l'impact de SOA sur les applications interactives ne doit pas être sous-estimé.

Cela conduit à revisiter le modèle classique MVC. Ce travail fait émerger naturellement les concepts de Transaction Longue et de Gestion de Contexte. Ces concepts ne sont pas triviaux, mais l'expérience montre que la vraie difficulté est de prendre

conscience que ces problèmes existent et doivent d'une façon ou d'une autre être pris en compte.

Enfin, l'utilisation d'un framework orienté application composite (CAF) permet d'homogénéiser la mise en œuvre de l'architecture proposée sur l'ensemble des solutions métier.

# 11

# Organiser un projet SOA : démarche, acteurs, outils

## Objectif

L'objectif de ce chapitre est de présenter des principes généraux de méthode et d'organisation dès lors qu'une direction informatique s'engage avec l'appui de ses maîtrises d'ouvrage dans une orientation SOA.

Le chapitre présente en premier lieu le concept de Programme SOA et la démarche méthodologique associée à ce concept. Cette démarche se présente sous la forme d'étapes, éventuellement déclinées en sous-étapes.

Chaque étape de la démarche est mise en œuvre par des acteurs de l'organisation regroupés en équipes projet et équipes transverses. Ces équipes se coordonnent via des comités périodiques. Enfin ces acteurs utilisent des outils. Acteurs, Structures et Outils sont décrits dans les différents sous-chapitres qui leur sont dédiés.

## 11.1 PLANIFIER

La démarche méthodologique décrite dans ce chapitre permet d'élaborer le planning d'un projet en le décomposant en étapes. Cette démarche est présentée par la figure 11.1 (en gris foncé les étapes probablement déjà réalisées en totalité ou partiellement au moment de l'adoption de SOA).

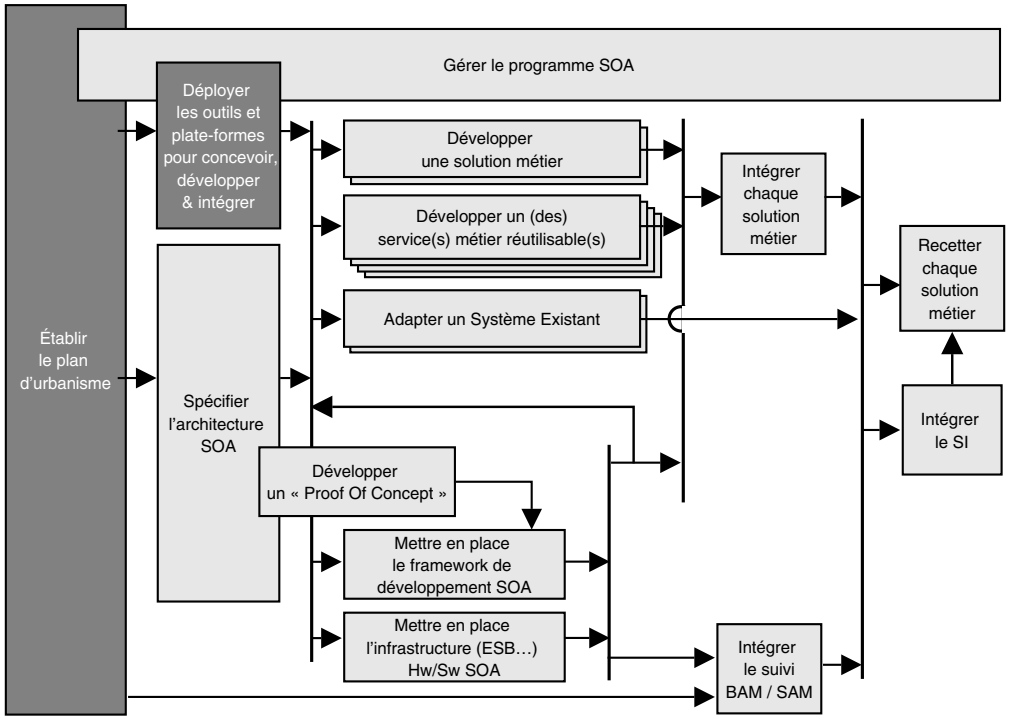



**Figure 11.1** — Démarche méthodologique

### 11.1.1 Utilisation de la démarche

La figure 11.1 présente un enchaînement macroscopique et chronologique des différentes étapes.

> Cet enchaînement chronologique doit être adapté en fonction du contexte propre à chaque projet SOA.

Autrement dit, l'important ici est la liste des étapes, car elles sont génériques. En revanche l'enchaînement chronologique dépend du contexte de chaque projet : il est en effet probable que le développement des solutions et services se fera par versions successives, ce que la figure 11.1 ne fait pas apparaître.

Par exemple, un service pourra d'abord avoir une version 0 sous forme de service « bouchon », afin de permettre le démarrage au plus tôt des tests des applications clientes du service. Une version 1 de qualification fonctionnelle permettra ensuite à la MOA de valider concrètement le service vis-à-vis de l'expression de besoin et à la MOE de corriger les bogues. Une version 2 de déploiement sur un site « pilote » donnera un retour d'expérience suffisant à la MOA pour rectifier une expression de besoin incomplète ou inadaptée. En cas de rectification, une version 3 sera réalisée pour le déploiement généralisé du service.

### Le concept de Programme SOA

La figure 11.1 repose par ailleurs sur l'hypothèse qu'il y aura plusieurs solutions métier à développer en parallèle, compte tenu de l'ampleur du projet. D'où l'introduction du concept de « Programme SOA », et la mise en place d'une structure dédiée à la gestion de ce programme, afin d'en assurer la coordination au quotidien, notamment sur le plan de la réutilisation.

De ce point de vue, un tel Programme verra la mise en place d'une ou plusieurs équipes dédiées à la réalisation de Services Métier réutilisables, en particulier les services métier CRUD dédiés à l'accès aux objets métier « racines » (client, fournisseur, demande de prestation, PDD, etc.).

**Remarque importante :** lorsqu'on parle dans ce chapitre d'acteur et d'équipe, il s'agit de rôle et de responsabilité, pas de personnes physiques.

Un acteur ou une équipe peuvent donc désigner une personne à temps partiel sur quelques jours, aussi bien que plusieurs personnes à temps plein sur plusieurs mois voire sur plusieurs années.

### Application à l'exemple Fil Rouge

Si l'on reprend l'exemple « fil rouge », on peut penser que l'étape « plan d'urbanisme » puis l'étape « architecture SOA » feront apparaître une charge de travail prévisionnelle importante. D'où la nécessité de mettre en place non pas une solution métier unique, mais un programme SOA coordonnant trois équipes orientées solutions métier et quatre équipes orientées services métier :

- Équipe solution métier « traitement et suivi d'une demande de prestation » : c'est le « cœur de cible », justifiant la mise en place d'une démarche SOA. C'est la solution métier initialement prévue.
- Équipe solution métier « recherche d'information sur les Points de Distribution, les Relevés de Consommations, les Contrats… ». Cette solution émerge pour éviter que l'équipe précédente ne devienne trop pléthorique.
- Équipe solution métier « portail d'Accueil et Gestion de l'information », basée sur la mise en œuvre d'un outil de gestion de contenu et d'un portail. Cette solution émerge car elle requiert des compétences spécifiques (compétence en gestion de contenus non structurés et outils associés, compétence en web design, etc.).
- Équipe dédiée au service de gestion des Demandes de Prestation et au service de gestion des Fournisseurs émettant ces demandes.
- Équipe dédiée aux services de gestion des Clients, de leurs Contrats, et de leurs Sites Géographiques.
- Équipe dédiée aux services d'accès aux PDD et aux relevés de consommation associés.
- Équipe dédiée au service de gestion de l'organisation (description des utilisateurs de la solution, de leur profil, etc.).

On décrit maintenant chaque étape de la démarche de façon synthétique par l'un des paragraphes ci-dessous, chaque description comprenant les rubriques suivantes :

- objectif de l'étape dans la démarche ;
- description générale ;
- éléments en entrée ;
- livrable(s) de l'étape (en précisant si nécessaire à quel(s) acteur(s) s'adresse la livraison) ;
- acteur(s) responsable(s) ou participant(s) à l'élaboration des livrables ;
- outil(s) ;
- points clefs.

### 11.1.2 Établir (ou faire évoluer) le plan d'urbanisme

#### *Objectif*

Établir la stratégie d'évolution du Système d'Information.

#### *Description générale*

L'objectif général est d'établir ou de faire évoluer le plan d'Urbanisme du SI en appliquant l'approche « processus e-Business » de l'orientation SOA.

#### *Élément(s) en entrée*

- stratégie métier d'évolution de l'entreprise.

#### *Livrable(s)*

- plan d'urbanisation :
  - principes d'évolution des processus métier : spécifier le modèle des événements métier pris en compte et l'architecture générale des processus associés ;
  - principes d'évolution des fonctions métier : en particulier, préciser les échanges entre les différentes fonctions ;
  - principe d'évolution des informations métier : Spécifier le modèle unifié (ou modèle pivot) des objets métier. Préciser l'architecture générale des référentiels associés : décider de répliquer ou non certaines informations, décider d'agréger certaines informations pour obtenir l'information « pivot » ;
  - principes d'évolutions technologiques du SI ;
- indicateurs de performance (KPI : *Key Performance Indicators*) :
  - performance métier de chaque processus ;
  - performance technique générale des applications, des services et des référentiels ;

- planification générale des évolutions :
  - livraison de tous ces livrables vers l'équipe Programme SOA (pour qu'elle puisse débuter ses travaux d'organisation et de planification), et vers la Direction Générale pour vérification de l'adéquation avec la stratégie de l'entreprise.

#### *Acteur(s)*

- Équipe Urbanisme.
- Équipe programme SOA (dès sa mise en place).

#### *Points clefs*

1 – Cette étape utilise une démarche moderne d'urbanisation du SI. Cette démarche s'appuie sur un référentiel méthodologique d'urbanisation général[1], et applique les recommandations du présent livre, notamment le chapitre 4 « SOA et Urbanisation » et le chapitre 9 « Modélisation des Processus Métier ».

### 11.1.3 Spécifier l'architecture SOA

#### *Objectif*

Spécifier chaque solution métier et chaque service.

#### *Description générale*

Cette étape a pour **premier objectif** d'établir la répartition des travaux à mener en une ou plusieurs solutions métier, une ou plusieurs équipes d'adaptation du SI existant, et/ou en une ou plusieurs équipes de développement de service. Cette répartition dépend des paramètres classiques en gestion de projet :

- charge de travail et dimensionnement « à taille humaine » des équipes;
- délai envisagé;
- compétences nécessaires.

Le **deuxième objectif** est de préciser le périmètre de chaque solution métier, et de fournir les moyens de tester chaque solution.

Le **troisième objectif** est de spécifier les services métier réutilisables à développer.

Le **quatrième objectif** est de préciser si et comment les solutions métier coopèrent et s'intègrent entre elles.

Le **cinquième objectif** est de spécifier les services techniques attendus du framework SOA.

1. Tel que celui décrit dans [Le Projet d'urbanisation du SI/C Longépé/DUNOD éditeur/2004].

*Élément(s) en entrée*

- Plan d'urbanisation.

*Livrable(s)*

- Périmètre des solutions métier :
  - pour chaque solution métier : modèle du ou des processus métier impliqués, use cases des applications composites interactives, fiche de description des Services Applicatifs associés aux applications;
  - pour chaque solution métier : liste des contraintes non fonctionnelles (NFR) pesant sur les processus et les applications.
- Liste des services métier réutilisables :
  - pour chaque service métier, fiche de description du service;
  - création et mise à jour de la matrice de réutilisation des services.
- Liste des adaptations du SI existant :
  - pour chaque service métier légataire, fiche de description du service.
- Liste des fonctionnalités attendues du framework SOA, en particulier liste des services techniques :
  - pour chaque service technique, fiche de description du service.
- Pour chaque Solution Métier, cahier de recette métier et jeux de tests associés.

*Acteur(s)*

- Équipe Urbanisme.
- Équipe programme SOA.
- Équipe Architecture Applicative.

*Points clefs*

1 – La matrice de réutilisation des services liste les services, et en face de chaque service, positionne les solutions et/ou les services de plus haut niveau utilisant ce service. Cette matrice permet de suivre le niveau de réutilisation globale des services, et met en évidence des anomalies à analyser (un service utilisé par aucune solution, ou une solution n'utilisant que des services qui lui sont dédiés).

### 11.1.4 Gérer le programme SOA

*Objectif*

Coordonner l'ensemble des équipes et assurer le contrôle Qualité du projet.

*Description générale*

L'objectif de coordination inclut les tâches suivantes :

- Planifier l'ensemble des travaux, définir les tâches (WBS) et définir les livrables.

- Organiser les différentes équipes de réalisation, affecter les tâches.
- Préciser le cycle de vie logiciel et synchroniser les équipes :
  - Organiser les réunions (comités de suivi, réunions de travail…).
  - Communiquer (mise à jour de l'intranet projet, modérer les forums sur le wiki, publier une lettre d'information, etc.).
  - Suivre les livraisons des différentes équipes vers l'équipe Intégration.
- Gérer le budget du programme SOA (si nécessaire, gérer la sous-traitance).
- Organiser le changement culturel, notamment mettre en place un plan de formation adapté au contexte (les compétences existantes au sein de la direction informatique et au sein de la MOA) et à la cible (les compétences à acquérir).
- Organiser le déploiement des outils et plates-formes nécessaires à la modélisation, au développement, à l'intégration et à la qualification des solutions, services et framework SOA.
- Contrôler la qualité :
  - Organiser les revues orientées qualité (revues d'architecture, de code, de documentation).
  - Organiser les revues orientées réutilisation.
  - Organiser les recettes (recettes fonctionnelles, qualification de pré production).
  - Organiser le suivi des Fiches de Fait Technique (c'est-à-dire les corrections d'anomalies et les demandes d'évolution).
- Gérer et animer la communication avec toutes les parties concernées.

#### *Élément(s) en entrée*

- Les différents livrables des étapes du projet.

#### *Livrable(s)*

- Planning tenu à jour.
- Budget tenu à jour.
- Plan qualité tenu à jour (ce document précise l'organisation, les points de synchronisation, le cycle de vie, les moyens et outils de contrôle de la qualité, etc.).
- Lettre d'information périodique.

#### *Acteur(s)*

- Équipe programme SOA.

#### *Outil(s)*

- Intranet projet (publication des documents, accès aux outils de gestion des anomalies, accès aux rapports de tests, etc.).

- Wiki de dialogue entre les différents intervenants.
- Outil de planification de projet (permettant le suivi du reste à faire).
- Outil de suivi budgétaire.

#### *Points clefs*

1 – La description qui précède repose sur la règle suivante :

> **Règle :** le responsable MOE du programme SOA doit avoir la responsabilité globale de l'architecture SOA, des solutions métier, des services, de l'architecture applicative et de l'infrastructure spécifique SOA.

L'expérience montre que, trop souvent, pour des raisons historiques ou politiques, certaines équipes ne sont pas rattachées au responsable du programme, or ces dérogations sont toujours sources de désagrément, car elles favorisent les classiques parties de ping-pong « ce n'est pas ma responsabilité, mais la sienne ». Pour combattre ce facteur d'entropie, il y a au moins une règle à respecter :

> **Règle :** l'équipe Intégration doit être rattachée au responsable du programme SOA, pour lui permettre de vérifier le bon avancement des travaux de tous les intervenants dans le programme.

2 – Autre point clef important, qui est également une source potentielle de désagrément (point non spécifique à SOA, mais que SOA rend plus critique) : le suivi du « reste à faire » n'est pas uniquement le suivi des charges nécessaires pour terminer les développements, mais aussi le suivi du « reste à industrialiser ». Autrement dit, une solution métier ou un service ne sont considérés comme terminés qu'une fois intégrés, qualifiés et débogués.

### 11.1.5 Développer une solution métier

#### *Objectif*

Assurer le développement et la livraison d'une solution métier.

#### *Description générale*

L'objectif principal est de s'assurer que l'ensemble des composants de la solution métier sera livré en respectant délais, coûts et adéquation aux besoins. Le responsable de la solution métier est donc un chef d'orchestre s'assurant que chaque fournisseur de composant livre en temps et en heure.

La réalisation de certains de ces composants (services métier, framework SOA) est déléguée à d'autres équipes, pour maximiser la réutilisation de ces composants et l'utilisation des compétences.

D'autres composants peuvent être directement pris en charge par l'équipe solution :

- Application composite.
- Processus métier.
- Certains services métier, soit parce qu'ils sont spécifiques de la solution (Services Applicatifs) soit parce qu'ils ne sont pas encore développés (services Fonctionnels) – cf. ci-dessous le paragraphe « points clefs de l'organisation ».

### *Élément(s) en entrée*

- Spécification du périmètre de la solution.
- Services métier réutilisables (pour chaque service : spécification puis service « bouchon » dans un premier temps, service « opérationnel » dans un second temps).
- Framework SOA et outillage associé.
- Formation spécifique sur le framework et l'outillage.
- Plate-forme de développement, plate-forme de tests.
- Fiche(s) de Fait Technique concernant la solution, émise(s) par l'équipe Intégration (lors du déroulement du programme).

### *Livrable(s)*

- L'ensemble des composants de la solution. Les composants sont assemblés, testés, packagés, documentés. Livraison à l'équipe Intégration avec un bordereau de livraison récapitulant les composants livrés et leur numéro de version, les Fiches de Fait Technique prises en compte dans la livraison, etc.
- Fiche(s) de Fait Technique concernant les services métier ou le framework SOA ou l'infrastructure SOA : livraison aux équipes concernées et à l'équipe Programme SOA (pour impact budget/planning).

### *Acteur(s)*

- Équipe Solution Métier.
- Équipe(s) Services Métier (fournisseur(s) de composants métier).
- Équipe Architecture Applicative (fournisseur du framework SOA et de ses composants techniques, et des outils associés).

### *Points clefs*

1 – Le développement d'une solution métier peut parfaitement réutiliser les méthodes et outils classiques dans le monde objet (SOA ne réinvente pas la roue sur ces sujets) :

- Contrôle des risques et de la qualité totale avec CMMI.
- Démarche itérative avec UP.
- Modélisation avec UML, complété avec BPMN pour les processus.

### 11.1.6 Développer un service réutilisable

*Objectif*

Assurer le développement et la livraison d'un service métier.

*Description générale*

L'objectif principal est de concevoir, coder et tester le service métier et ses opérations.

*Élément(s) en entrée*

- Fiche de description du service à réaliser.
- Framework SOA et outillage associé.
- Formation spécifique sur le framework SOA et l'outillage.
- Fiche(s) de Fait Technique concernant le service, émise(s) par les équipes Solution utilisatrices ou l'équipe Intégration (lors du déroulement du programme).

*Livrable(s)*

- Description WSDL du service : livraison à l'équipe d'urbanisme et à l'équipe d'intégration pour vérifier le respect du contrat de service, livraison aux solutions métier pour préparer et/ou vérifier la conception de leurs composants.
- Implémentation « bouchon » : livraison aux solutions métier pour leurs tests.
- Implémentation « opérationnelle » : livraison aux solutions métier pour leurs tests, livraison à l'équipe intégration.
- Fiche(s) de Fait Technique concernant le framework SOA ou l'infrastructure SOA : livraison à l'équipe concernée et à l'équipe Programme SOA (pour impact budget/planning).

*Acteur(s)*

- Équipe « service métier ».
- Équipe urbanisme (contrôle voire participation à la conception du service).

*Points clefs*

**Règle :** livrer un service « bouchon » aussi rapidement que possible aux solutions métier utilisatrices, si possible en même temps que la publication officielle du contrat de service WSDL.

### 11.1.7 Adapter un Système Existant

*Objectif*

Adapter un système existant à son utilisation dans une solution métier SOA.

### Description générale

L'objectif est de fournir un service d'accès à un système existant (cf. chapitre 7.4 « zoom sur les systèmes existants »). Cela implique dans certains cas de faire évoluer le système lui-même. L'urbanisation puis l'étude d'architecture SOA se seront assurées de la faisabilité (technique et économique) de ces évolutions.

### Élément(s) en entrée

- Plan d'urbanisme et architecture SOA (spécification des adaptations).
- Fiche de description du service à réaliser (pour concevoir le service).
- Framework SOA et outillage associé (pour réaliser le service).
- Formation spécifique sur le framework SOA et l'outillage.
- Fiche(s) de Fait Technique concernant le service, émise(s) par les équipes Solution utilisatrices ou l'équipe Intégration (lors du déroulement du programme).

### Livrable(s)

- Description WSDL du service : livraison à l'équipe d'urbanisme pour vérifier le respect du contrat de service, livraison aux solutions métier pour préparer et/ou vérifier la conception de leurs composants.
- Implémentation « bouchon » : livraison aux solutions métier pour leurs tests.
- Implémentation « opérationnelle » : livraison à l'équipe intégration.
- Fiche(s) de Fait Technique concernant le framework SOA ou l'infrastructure SOA : livraison à l'équipe concernée et à l'équipe Programme SOA (pour impact budget/planning).

### Acteur(s)

- Équipe de maintenance du système existant.
- Équipe urbanisme (contrôle voire participation à la conception du service).

### Points clefs

1 – La livraison de service « bouchon » est encore plus sensible dans ce contexte d'accès à un système existant. Il est en effet fréquent que, vu le coût d'installation de ces services, les solutions métier attendent la phase d'intégration pour se confronter aux services dans leur version opérationnelle.

2 – Ces adaptations sont confiées aux équipes chargées de la maintenance des systèmes existants. Compte tenu des technologies en jeu, c'est souvent la seule façon de faire, et c'est de plus une source de motivation pour ces équipes, à condition qu'elles soient correctement formées et accompagnées sur les nouvelles technologies et nouvelles architectures. Elles peuvent en retour apporter leur longue expérience des processus d'industrialisation du logiciel.

### 11.1.8 Mettre en place le framework SOA

#### *Objectif*

Livrer aux équipes de réalisation un framework SOA, ses services techniques et ses outils.

#### *Description générale*

L'objectif est d'élaborer et de livrer aux développeurs un framework permettant de réaliser les applications et services SOA. Cet objectif implique soit de développer soit de choisir (en les adaptant si besoin) des composants chez les éditeurs ou dans le monde Open source, et d'intégrer ces composants dans un tout cohérent.

Cet objectif implique également de développer ou de choisir des outils de productivité pour faciliter la mise en œuvre du framework.

#### *Élément(s) en entrée*

- Spécification du framework (au démarrage du programme).
- Fiche(s) de Fait Technique concernant le framework émise(s) par les autres équipes, en particulier l'équipe Intégration (lors du déroulement du programme).

#### *Livrable(s)*

- Framework SOA.
- Outillage associé.
- Guide de mise en œuvre et exemples de code.
- Formation spécifique sur le framework et l'outillage.
- Fiche(s) de Fait Technique concernant l'infrastructure SOA : livraison à l'équipe concernée et à l'équipe Programme SOA (pour impact budget/planning).

#### *Acteur(s)*

- Équipe Architecture Applicative (responsable).
- Équipe(s) « solution » et/ou équipe(s) « service », si l'équipe leur délègue la réalisation ou l'adaptation de certains services techniques du framework (cf. paragraphe organisation ci-dessous).
- Équipe Infrastructure SOA.

#### *Points clefs*

1 – Le framework est la clef de voûte des développements : sa qualité est donc l'une des conditions du succès de tout programme SOA. Il est nécessaire de minimiser les risques dès le démarrage de ce programme.

La mise au point du framework et de l'infrastructure, mais aussi l'apprentissage sur le terrain des bonnes pratiques SOA, le test des performances de l'infrastructure sous-jacente constitue autant de raisons en faveur du développement d'un POC.

**Règle :** pour minimiser le risque « framework », il faut développer un prototype, ou POC (*Proof Of Concept*), sous forme d'une solution métier simplifiée (mais réaliste sur le plan métier) centrée sur la mise en œuvre du framework et de l'infrastructure logicielle et matérielle.

Ce développement doit, si possible, se faire en avance de phase par rapport au démarrage des solutions métier.

### 11.1.9 Mettre en place l'infrastructure SOA

#### *Objectif*

Installer une infrastructure logicielle et matérielle SOA.

#### *Description générale*

L'objectif est de choisir, installer, paramétrer, qualifier et préparer l'exploitation d'une infrastructure SOA complète, aussi bien sur le plan logiciel (ESB, EII…, cf. partie 6) que matériel (*load balancing*, cluster, etc.).

Une telle infrastructure peut, pour des projets importants, inclure de nombreux composants. Cet objectif implique donc un travail important, d'autant que ces outils sont nouveaux et nécessitent donc au préalable une appropriation culturelle intensive.

#### *Élément(s) en entrée*

- Plan d'urbanisation (notamment orientations technologiques, listes des NFR).
- Spécification du framework (pour mettre en évidence des besoins particuliers en matière d'infrastructure : connecteurs particuliers, par exemple).
- Fiche(s) de Fait Technique concernant l'Infrastructure SOA émise(s) par les autres équipes, en particulier l'équipe Intégration.

#### *Livrable(s)*

- Infrastructure SOA installée et paramétrée sur les différentes plates-formes concernées.
- Outils (guide de mise en œuvre…) et formation à l'usage de l'équipe Architecture Applicative et des équipes de développement.

#### *Acteur(s)*

- Équipe infrastructure SOA.

#### *Points clefs*

1 – L'installation de cette infrastructure concerne l'ensemble des plates-formes du programme SOA : tests, intégration, qualification fonctionnelle, qualification technique, et bien sûr production.

2 – La mise au point de cette infrastructure potentiellement complexe milite également en faveur d'un POC (cf. paragraphe précédent).

### 11.1.10 Intégrer une solution métier

#### *Objectif*

Intégrer et évaluer une solution métier.

#### *Description générale*

L'objectif est d'intégrer chaque solution métier, en effectuant les différentes catégories de test nécessaires, comme illustré par la figure 11.2.

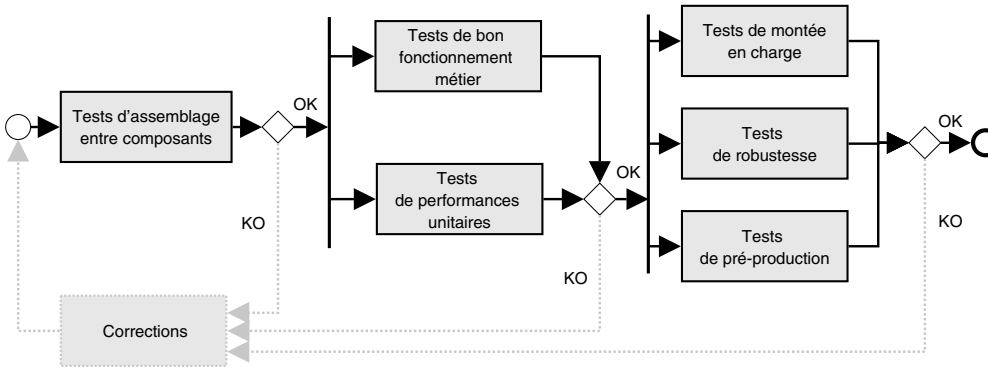



**Figure 11.2** — Démarche méthodologique d'intégration

Les tests d'assemblage permettent notamment de vérifier le respect des contrats de service par le client (la solution métier) et par le fournisseur (le service).

Les tests de Bon Fonctionnement métier concernent :

- Pour les nouveaux composants : la vérification du respect des exigences métier (pré-recette de préparation de la recette MOA).
- Pour les composants déjà intégrés : la vérification de la non régression fonctionnelle.

#### *Élément(s) en entrée*

- Solution métier installée sur la plate-forme d'intégration, avec bordereau de livraison.
- Infrastructure SOA installée et paramétrée sur la plate-forme d'intégration.
- Cahier de recette métier et jeux de tests associés.

#### *Livrable(s)*

- Fiche(s) de Fait Technique concernant la solution métier, les services métier utilisés, le framework SOA ou l'Infrastructure SOA : livraison aux équipes concernées et à l'équipe Programme SOA (pour impact budget/planning).

#### *Acteur(s)*

- Équipe Intégration.
- Équipes de développement (assistance à la mise en œuvre de la solution et à l'analyse des problèmes).
- Équipe Urbanisme (assistance aux tests métier).

#### *Points clefs*

1 – La solution métier à intégrer peut inclure des services métier « bouchons » lorsqu'il s'agit de services d'accès aux systèmes existants ou de services de communication avec une autre solution métier. L'étape d'intégration globale « niveau SI » prendra en charge la vérification de ces accès « en vraie grandeur ». Ceci afin d'éviter une synchronisation trop étroite entre les différents développements concernés. Cette décision d'accepter ou non des services « bouchons » est à prendre au niveau Programme SOA.

### 11.1.11 Intégrer le suivi BAM/SAM

#### *Objectif*

Installer, paramétrer, compléter une solution BAM (*Business Activity Management*) + SAM (*Service Activity Management*).

#### *Description générale*

L'objectif est de mettre en place une solution de type BAM et SAM, à partir d'outils BAM/SAM, en paramétrant ces outils (notamment en terme de profil d'accès), et éventuellement en réalisant des développements complémentaires.

#### *Élément(s) en entrée*

- Plan d'urbanisation (notamment définition des KPI à suivre).
- Infrastructure SOA installée et paramétrée sur les différentes plates-formes concernées (les outils de BAM/SAM s'appuieront sur des composants de cette infrastructure).

#### *Livrable(s)*

- Infrastructure BAM et SAM installée et paramétrée sur les différentes plates-formes concernées.
- Outils (guide d'utilisation) et formation à l'usage des utilisateurs finaux et des équipes de production.

*Acteur(s)*

- Équipe Solution BAM/SAM – cette équipe peut ou non faire partie de l'équipe architecture applicative.
- Équipe Infrastructure.

*Points clefs*

1 – On parle ici de solutions BAM/SAM, et non pas uniquement d'outils BAM/SAM. L'expérience montre en effet que les outils disponibles dans l'infrastructure SOA doivent être en général complétés/adaptés : par exemple, un outil de BAM doit, pour des raisons ergonomiques, être intégré dans la Corbeille des responsables (cf. tableau 9.1). Autre exemple, il peut être nécessaire de mettre en place un véritable portail orienté SAM, permettant de consulter en temps quasi réel la performance de chaque service.

### 11.1.12 Intégrer le Système d'Information

*Objectif*

Vérifier que toutes les solutions métier coopèrent ensemble et avec les systèmes existants.

*Description générale*

Le premier objectif est de vérifier que les solutions métier coexistent sans effet de bord au sein du SI.

Le deuxième objectif est de vérifier que chaque solution métier accède correctement aux systèmes existants via les services concernés (c'est-à-dire qu'on teste les solutions métier non plus avec les services bouchons mais avec les vrais services d'accès).

Le troisième objectif est de vérifier que les solutions métier devant communiquer entre elles (via l'ESB) le font correctement.

Le quatrième objectif est de vérifier le bon fonctionnement de certains services techniques transverses au niveau SI, ce qui implique notamment :

- Les tests d'évaluation des outils de sécurité (cf. partie 4 pour une présentation des normes et outils de la sécurité dans un contexte SOA).
- Les tests d'évaluation des solutions de réplication.

*Élément(s) en entrée*

- Solutions métier installées sur la plate-forme d'intégration, avec bordereau de livraison.
- Infrastructure SOA installée et paramétrée sur la plate-forme d'intégration.
- Cahier de recette métier et jeux de tests globaux.

*Livrable(s)*

- Fiche(s) de Fait Technique concernant les solutions métier, les services utilisés, ou l'Infrastructure SOA : livraison aux équipes concernées, à l'équipe Urbanisme, et à l'équipe Programme SOA (pour impact budget/planning).

*Acteur(s)*

- Équipe Intégration.
- Équipes de développement (assistance à la mise en œuvre des solutions et à l'analyse des problèmes).
- Équipe Urbanisme (assistance aux tests métier et à l'analyse des problèmes globaux).

## 11.2 ORGANISER

### 11.2.1 Acteurs

Les principales équipes impliquées sont (liste récapitulative) :

- Équipe(s) Solution métier (dont équipe BAM/SAM) (SOL).
- Équipe(s) Services métier (SER).
- Équipe(s) Maintenance des Systèmes Existants (MAIN).
- Équipe Architecture applicative et ateliers de mise en œuvre (ARCH).
- Équipe Architecture infrastructure (INF).
- Équipe Administration des composants (ADM).
- Équipe Intégration (INT).
- Équipe Programme SOA (PROG).
- Équipe Urbanisme (URBA).
- Sponsor Maîtrise d'Ouvrage (SPON).
- Équipe Maîtrise d'Ouvrage (MOA).

### 11.2.2 Responsabilités

Le tableau 11.1 fournit le diagramme RACI de l'organisation. Pour chaque étape de la démarche SOA, le diagramme précise qui est Responsable, Acteur, Coopérant ou Informé. Le Responsable est celui qui contrôle en dernier ressort les livrables de l'étape, l'Acteur est celui qui produit les livrables, le Coopérant participe activement à l'élaboration des livrables, l'Informé est informé des travaux et fournit si besoin des informations.

**Tableau 11.1** — Diagramme RACI

| Étapes | SPON | MOA | URBA | PROG | SOL | SER | MAIN | ARCH | INF | ADM | INT |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Établir le plan d'urbanisme | R | C | **A** | I | | | | | | | |
| Spécifier l'architecture SOA | | C | **A** | R | | | | C | | | |
| Gérer le programme SOA | R | I | I | **A** | C | C | C | C | C | C | C |
| Développer une solution métier | | | C | I | **RA** | C | C | C | C | I | I |
| Développer les services métier | | | C | I | I | **RA** | C | C | C | I | I |
| Adapter un Système existant | | | C | I | I | | **RA** | C | C | I | I |
| Mettre en place l'architecture et le framework SOA | | | I | I | I | I | I | **RA** | C | I | I |
| Mettre en place l'infrastructure SOA | | | I | I | I | I | I | C | **RA** | I | I |
| Intégrer une solution métier | | | C | I | C | C | C | C | I | C | **RA** |
| Intégrer le suivi BAM/SAM | | | C | I | C | I | I | I | C | I | **RA** |
| Intégrer le Système d'Information | | | C | I | C | C | C | I | I | I | **RA** |

### 11.2.3 Coordination

La coordination implique classiquement la mise en place de comités périodiques :

- Comité de Programme SOA : gestion de la synchronisation des plannings et allocation de ressources, prise de décision concernant les points durs.
- Comité d'urbanisation SOA : notamment suivi de la matrice solutions/services.
- Comité de suivi « solution métier » : notamment suivi de l'intégration progressive de la solution.
- Comité de suivi « architecture » : notamment suivi de la mise en place et de la mise au point du framework SOA et de l'infrastructure SOA.

### 11.2.4 Communication

L'un des objectifs clefs de l'utilisation de l'approche SOA est de maximiser la réutilisation de services (métier et technique). Cela passe par le bon fonctionnement de la relation client ↔ fournisseur entre les équipes « solution métier » (les clients), et les équipes « services métier » et « architecture applicative/framework » (les fournisseurs).

Or réutiliser ne se décrète pas : si une équipe Solution n'a pas confiance dans les composants d'un de ses fournisseurs, elle ne les réutilisera pas, quelles que soient les

obligations qui pèseront sur elle. C'est normal : on ne peut pas demander à un responsable de Solution de s'engager sur des délais et des coûts, et en même temps lui imposer des freins ou ce qu'il perçoit comme tel.

Ce manque de confiance arrive très souvent, car les informaticiens ne sont pas des adeptes naturels de la communication intensive : cette absence de communication, si elle n'est pas combattue, impliquera que chacun interprète dans son coin les spécifications, ce qui conduit inévitablement à des incompréhensions[1].

Cette confiance fonctionne dans les deux sens : si un fournisseur s'aperçoit qu'un client est de mauvaise foi (en étant atteint du syndrome NIH[2], par exemple, ou en ne faisant pas l'effort nécessaire d'appropriation du composant fourni), cela démotivera ce fournisseur.

Créer les conditions de la confiance, tel est le point clef de la réutilisation. Cela passe par :

- Créer les opportunités de communiquer entre équipes : les comités ne suffisent pas, il faut également des réunions d'information et d'échanges régulières.
- Organiser des « échanges culturels » : un architecte va par exemple être détaché quelques semaines dans une équipe « solution » pour adapter le framework de développement. Autre exemple, un développeur de l'équipe Solution S1 va être détaché dans une équipe Service Métier pour développer ou adapter un Service Métier dont la solution S1 est la première « cliente ». Ces échanges sont très efficaces, car ils n'empiètent pas sur les prérogatives des responsables d'équipes « fournisseur » tout en rassurant les responsables « clients » sur la bonne prise en compte de leurs besoins.
- Mettre en place des outils simples comme support de la communication, tel un WIKI permettant d'une part d'accéder simplement aux documentations disponibles, d'autre part de consulter la roadmap des composants et solutions, et enfin de garder une trace des FAQ.

Mais créer la confiance ne signifie pas laxisme. L'équipe Programme SOA doit mettre en place des revues périodiques d'architecture, permettant de contrôler cette réutilisation.

## 11.3 OUTILLER

### 11.3.1 Le framework SOA

Le framework SOA est l'outil de base pour concevoir et développer les solutions et les services : il concrétise l'architecture applicative SOA.

1. Citons le cas – caricatural mais hélas réel – de la perte de la sonde Mars Climate Orbiter en 1999, parce qu'une équipe logicielle avait raisonné en mètre, et une autre en miles !
2. NIH = *Not Invented Here*.

On rappellera qu'un framework comprend deux types de composants :

- Les composants génériques : un tel composant est un squelette de code à compléter par le développeur (dans le monde objet, on parlera de réutilisation par héritage) : comme exemple de composant générique, on citera le composant « squelette de Service Applicatif », le composant « squelette d'action », etc.
- Les services techniques, prêts à l'emploi après paramétrage : comme exemple, on citera le service « log », le service « impression de rapport », le service « verrouillage métier », etc.

Le framework SOA est lui-même composé de (sous) frameworks :

- Le framework CAF dédié aux Applications Interactives.
- Le framework dédié aux Processus Métier, basé sur un moteur d'orchestration BPEL.
- Le framework Services, notamment en ce qui concerne les services CRUD (orienté objet métier racine) et les accès aux référentiels.
- Éventuellement, un framework orienté Règles de Gestion IF, THEN, ELSE.

### 11.3.2 Les outils

Pour faciliter la mise en œuvre de ce framework, et garantir un niveau de productivité suffisant, il faut mettre en place un outillage complet.

La figure 11.3 présente un panorama général des ateliers logiciels à mettre en place au sein de ce qu'on peut baptiser de chaîne de fabrication SOA.

L'atelier de Modélisation comprend l'ensemble des éditeurs de modélisation graphiques (UML, BPMN, formalisme propriétaire.) ou textuels (éditeur XML).

L'atelier de Production comprend l'ensemble des outils de développement manuel et de génération automatique de code[1].

L'atelier de Composition comprend l'ensemble des outils permettant d'assembler les composants à partir d'autres composants.

L'atelier d'Homologation comprend l'ensemble des outils de tests permettant de tester et qualifier les composants SOA, en particulier les services.

### 11.3.3 Outils et productivité : SOA et MDA

La productivité obtenue à l'aide de cette chaîne de fabrication SOA est un élément clef du succès de l'approche SOA telle que décrite dans ce livre. Il ne saurait être question que les avantages d'une telle approche soient obtenus au détriment de cette productivité, à partir du moment où les investissements à consentir sont réalisés.

1. Exemples : ECLIPSE, RAD, NETBEANS, VISUAL STUDIO.NET, etc.

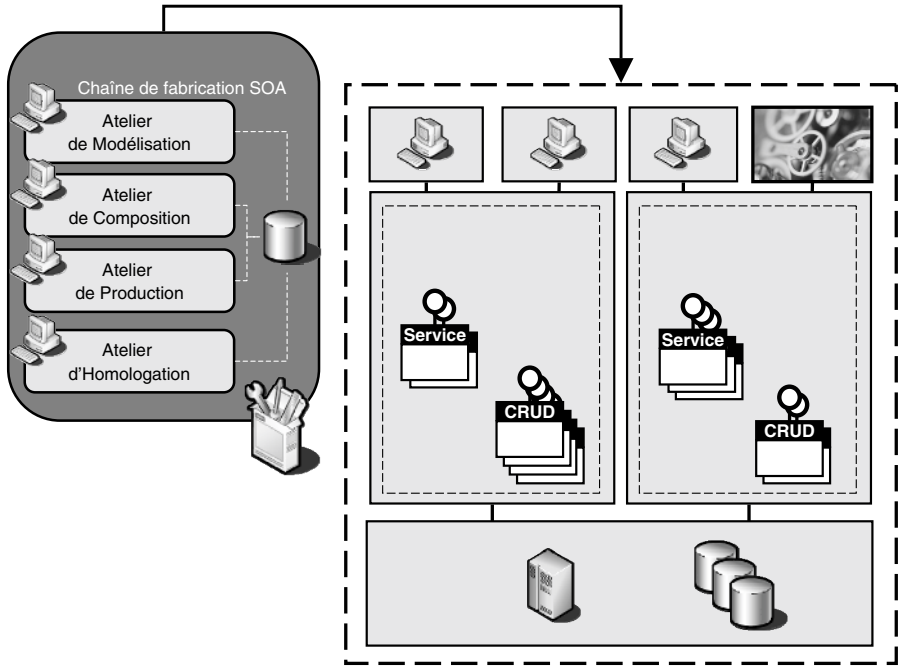



**Figure 11.3** — Rôle et place des outils du framework SOA

Il est donc logique que l'approche architecturale SOA se rapproche de l'approche MDA, *Model Driven Architecture*. Il ne s'agit pas ici de céder à la tentation du mariage futuriste de deux acronymes très à la mode en ce début de XXIe siècle, mais bien d'étudier comment, en restant pragmatique, il est possible d'orienter les idées MDA vers la génération de composants SOA.

> La démarche MDA prône la génération automatique de code à partir de modèles UML. Plus précisément, il s'agit, à partir de modèles UML représentant le métier à informatiser (modèles PIM : *Platform Independant Model*), de raffiner par étape ces modèles pour en faire des modèles PSM (*Platform Specific Model*) permettant in fine une génération de code complète, sans passer par l'étape « écriture de logiciel à la main ». Les modèles PIM sont indépendants de toute technologie, les modèles PSM sont associés à une technologie cible (Java/Swing/RMI, J2EE/Spring, J2EE/EJB, VB/.NET, etc.).

La figure 11.4 présente une démarche MDA adaptée au contexte SOA.

### *Modèles Graphiques (PIM)*

La modélisation « métier » débute sa phase de structuration « technique » grâce à une formalisation UML/BPMN des processus métier. Un tel formalisme du modèle de processus permet de générer le fichier BPEL associé, qui lui est directement exécutable

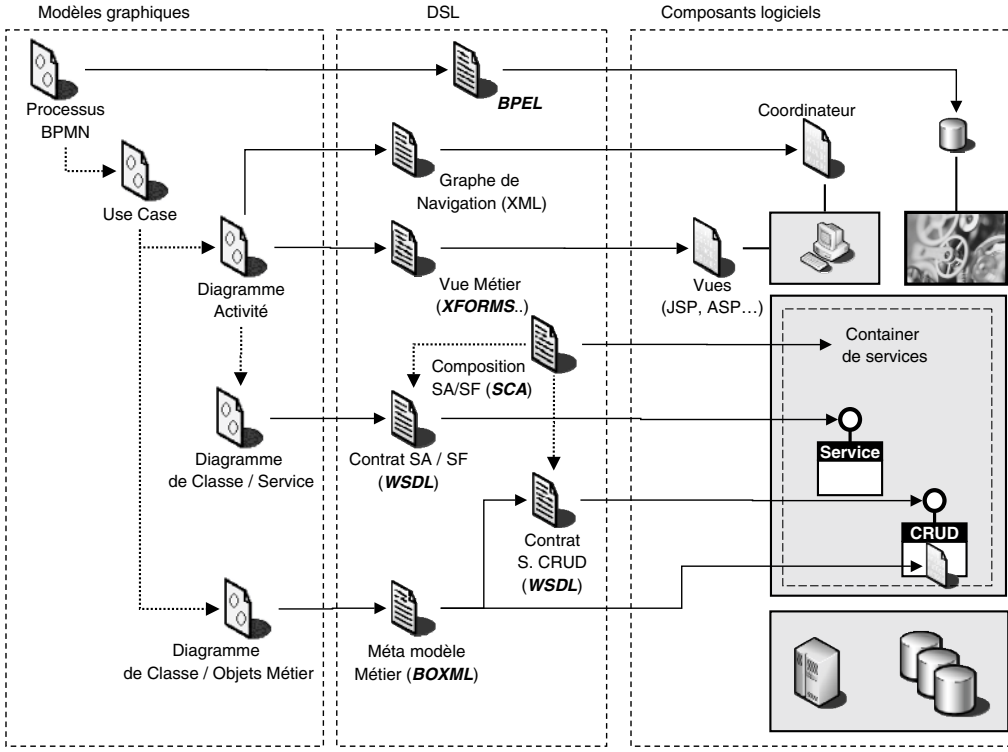



**Figure 11.4** — SOA et MDA : une approche possible

par le moteur d'orchestration de processus (BPM). D'autre part, il met en évidence les applications interactives nécessaires pour faire intervenir l'homme dans la boucle (cf. chapitre 9 et ses exemples de processus BPMN).

Toute application interactive est décrite à son tour, par un diagramme de Cas d'Utilisation, décrivant l'interaction entre l'utilisateur et la solution. Elle est associée à :

- Un diagramme d'activité décrivant le comportement du Coordinateur de l'application, c'est-à-dire la cinématique de navigation entre les vues de l'application et les services appelés (cf. chapitre 10) lors des interactions utilisateur ↔ solution.
- Un diagramme de classe listant et décrivant les services appelés (Services Applicatifs/Services Fonctionnels).

Les applications interactives d'une même solution sont également associées à un diagramme de classe modélisant les objets métier utilisés.

### *Modèles PSM*

À partir de ces modèles graphiques, il est possible de générer les modèles PSM. La démarche proposée est légèrement différente d'une approche MDA centrée sur UML :

en effet, le modèle PSM d'un composant n'est pas un modèle UML annoté, mais un document XML obéissant à une syntaxe spécifique du type de composant concerné[1]. Par exemple, chaque Vue pourra être décrite par un fichier au format XFORMS ou XUL, tandis que chaque Coordinateur sera décrit par un graphe de navigation XML au format `jsf-config.xml`.

Les diagrammes de services permettent de générer les contrats d'interface WSDL. À partir de ces contrats, il est également possible de composer des services de plus haut niveau, via un éditeur SCA.

Le modèle des Objets Métier quant à lui permet de générer les Services CRUD d'accès aux référentiels. Il permet également d'enrichir les Vues générées (le typage des attributs d'un objet métier permet de générer le contrôle de saisie correspondant)[2].

## 11.4 INDUSTRIALISER : VERS UN MODÈLE DE MATURITÉ SOA

Le problème récurrent de toute démarche informatique est tout simplement de l'appliquer, et si possible avec rigueur. Les bonnes résolutions ne suffisent pas.

Or l'application d'une telle démarche ne se fait pas instantanément, pour une raison simple : elle implique une appropriation culturelle par une équipe d'êtres humains. Il ne s'agit pas seulement de former cette équipe, mais également de lui permettre d'acquérir progressivement de l'expérience, c'est-à-dire de mûrir vis-à-vis d'un paradigme technique nouveau.

### 11.4.1 Les modèles de maturité

Cela explique l'apparition puis le succès de *modèle de maturité*, dont le plus connu est le modèle CMM-I (*Capability Maturity Model – Integrated*). Un modèle de maturité définit d'une part les capacités qu'une organisation (une direction informatique, une SSII, etc.) doit acquérir pour maîtriser un certain type de compétence informatique, et d'autre part définit les niveaux de maturité que cette organisation atteindra successivement, permettant ainsi d'en mesurer les progrès.

La définition d'un tel modèle dans le contexte SOA est d'autant plus nécessaire qu'il s'agit d'une évolution, voire une révolution, dans la façon d'approcher le Système d'Information d'une entreprise. Des travaux ont vu récemment le jour pour

1. Ces syntaxes spécifiques sont autant de DSL (Domain Specific Language). Dans la mesure où ces langages sont déclaratifs et non programmatiques, il s'agit cependant bien d'une démarche respectant l'esprit MDA. Il n'y a pas lieu d'opposer l'approche DSL (défendue par MICROSOFT) de l'approche MDA (défendue par le reste du monde regroupé dans l'OMG).
2. Le lien entre fichiers XML « Vue » et fichier XML « méta modèle métier » n'est pas représenté sur la figure 11.4 pour ne pas la surcharger.

définir un modèle de maturité appliqué au contexte SOA. On citera par exemple les travaux d'IBM[1] et de MICROSOFT[2].

L'expérience des auteurs les a conduits à définir un modèle de maturité SOA, baptisé PSAUMM : *Practical, Service-oriented Architecture, Unified Maturity Model*. Ce modèle est composé de 5 axes de progression, comme le met en évidence la figure 11.5.

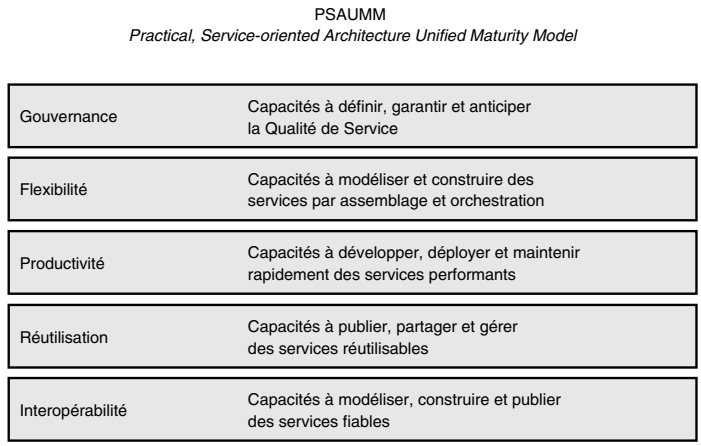



**Figure 11.5** — Modèle PSAUMM : les axes de progression

La progression se fait sur ces axes de façon indépendante, selon la situation initiale et les objectifs de l'entreprise.

### *PSAUMM et CMMI*

On notera que l'objectif de PSAUMM est d'évaluer la maturité de l'architecture SOA mise en place. Ce type de modèle est donc parfaitement complémentaire de CMMI, qui lui a pour objectif d'évaluer la maturité des équipes dans la maîtrise du cycle de vie logiciel.

Il est probable que le modèle CMMI aura une durée de vie supérieure à un modèle de maturité orienté Architecture, si l'on en juge par l'histoire de l'informatique depuis 30 ans. C'est d'ailleurs une raison suffisante pour distinguer un modèle de maturité orienté « génie logiciel » et un modèle de maturité orienté « architecture SOA »[3].

La suite du chapitre décrit les niveaux de maturité du modèle et les capacités associées.

1. Modèle SIMM = *Service Integration Maturity Model* – cf. par exemple : SOA compliance : initial steps in a longer journey/J. Falkl et alii/www.ibm.com/décembre 2005.
2. Modèle ESOMM = *Enterprise Service Orientation Maturity Model* – cf. par exemple : Enable the Service Oriented Enterprise/William Oellermann/The MS Architecture Journal, n° 7/avril 2006.
3. Certains modèles de maturité SOA essaient en effet de modifier directement CMMI en lui faisant prendre en compte des aspects architecturaux précis. Ce n'est clairement pas l'orientation de PSAUMM et des modèles de ce type.

## 11.4.2 Le modèle PSAUMM

Le modèle comprend 3 niveaux de maturité, comme le montre la figure 11.6. Cette figure liste également les capacités mises en jeu en fonction de l'axe de progression et du niveau de maturité.

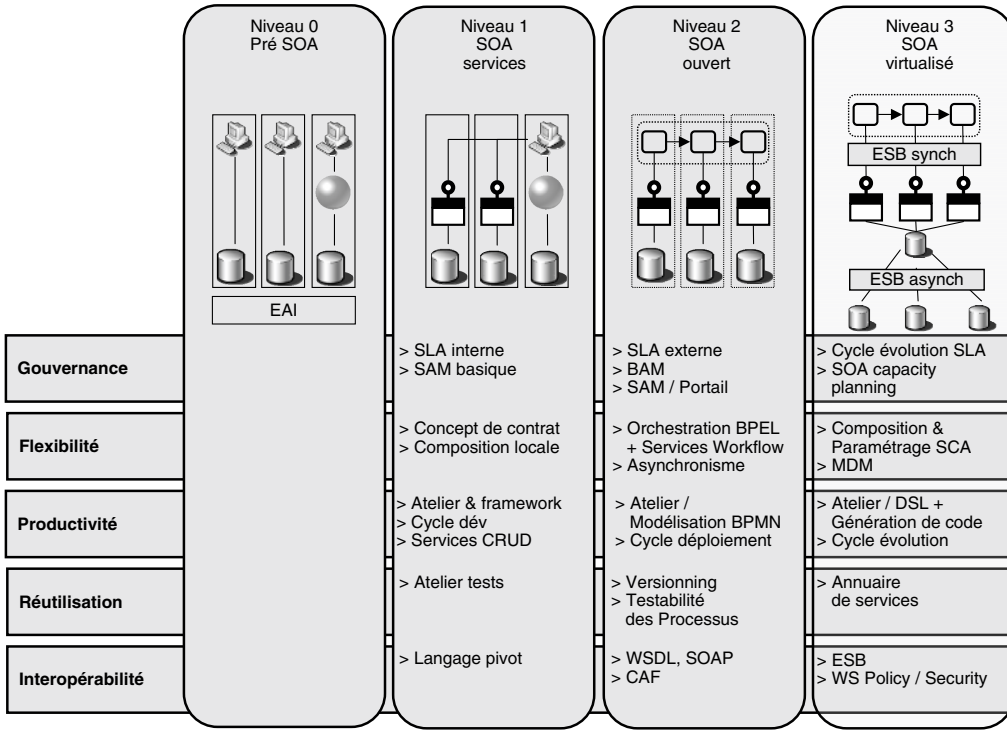



**Figure 11.6** — Modèle PSAUMM : niveaux de maturité et capacités associées

Le niveau 0 correspond à la situation initiale des entreprises ayant d'une part entrepris un effort d'urbanisation de leur système (EAI) et d'autre part ayant adopté les technologies Objet pour le développement de leurs nouvelles applications.

Le niveau 1 correspond à la situation d'émergence des services métier au sein des nouvelles applications, à la mise en place des capacités de développement rapide, et à la mise en place de la démarche décrite dans le présent chapitre.

Le niveau 2 correspond à la mise en place des processus e-business, à la mise en place des capacités d'intégration et de déploiement rapide, et à l'ouverture du système d'information.

Le niveau 3 correspond à la généralisation de l'approche SOA au système d'information, à la capacité d'évolution des services réutilisés, et à la capacité d'anticiper sur le dimensionnement du système en fonction des besoins métier.

### 11.4.3 Les capacités du modèle

On décrit dans ce sous-chapitre les différentes capacités à acquérir progressivement. On notera une règle intangible :

> **Règle :** l'acquisition des capacités PSAUMM implique la mise en place concomitante d'un véritable plan de formation et d'accompagnement.

#### *Axe Interopérabilité*

La première capacité à acquérir sur cet axe concerne la mise en place d'un premier ensemble de services CRUD : cet objectif implique la définition d'un langage « pivot », basé sur la modélisation des Objets Métier accédés via les Services.

La deuxième capacité concerne la maîtrise des Web Services, nécessaires d'une part à l'ouverture du système d'information, et d'autre part à la mise en œuvre des Processus e-Business. Cette capacité peut être nécessaire dès le premier niveau de maturité architectural, en fonction des objectifs généraux de l'entreprise.

La troisième capacité concerne la maîtrise de l'architecture des Applications Composites : cela implique la compréhension des concepts de Contexte et de Transaction longue, et la mise en place d'un framework (CAF).

La quatrième capacité concerne le déploiement d'un ESB complet, tel que décrit au chapitre 17.

La cinquième capacité concerne la maîtrise des aspects avancés de SOA, notamment dans le domaine de la sécurité.

#### *Axe Réutilisation*

La première capacité à acquérir sur cet axe concerne la mise en place d'un atelier d'homologation orienté vers les tests des services à développer : l'objectif est de garantir le niveau de qualité des services présentés comme réutilisables.

La deuxième capacité concerne la maîtrise du versionning des services.

La troisième capacité concerne la maîtrise des tests des Processus. Elle s'appuie sur la première capacité de cet axe, mais implique en sus un travail sur les plates-formes de test et d'intégration, les jeux de tests et les outils d'espionnage des Processus.

La quatrième capacité concerne la publication des services dans un annuaire de Services.

#### *Axe Productivité*

La première capacité à acquérir sur cet axe concerne la mise en place d'un atelier de fabrication, orienté objet. L'atelier est organisé autour d'un référentiel permettant de gérer les versions des composants et les configurations des solutions métier obtenues par assemblage des composants.

La deuxième capacité concerne la définition d'un cycle de développement des composants SOA. Ce cycle définit les modalités de travail en équipe (en lien avec la mise en place du référentiel évoqué précédemment).

La troisième capacité à acquérir sur cet axe concerne la possibilité de générer tout ou partie des services CRUD à partir d'une description des Objets Métier.

La quatrième capacité a pour objectif la mise en place d'un atelier permettant la modélisation de Processus et la génération du code BPEL associé. Cet atelier permet également de gérer le versionning des modèles.

La cinquième capacité concerne la définition d'un cycle d'intégration et de déploiement. Complément du cycle de développement, ce cycle définit les différentes étapes menant des tests unitaires au déploiement opérationnel. Il affine le cycle présenté par la figure 11.2.

La sixième capacité a pour objectif la mise en place d'un atelier de Composition de services et d'application composite interactive. Cet atelier inclut les éditeurs XML permettant d'éditer ou de modifier les fichiers DSL évoqués au sous-chapitre Outillage.

La septième capacité concerne la définition du cycle d'évolution (maintenance corrective, maintenance évolutive) des solutions métier et des services. Ce cycle précise également la procédure à utiliser en cas d'urgence (correction d'un bogue bloquant la production).

### *Axe Flexibilité*

La première capacité à acquérir sur cet axe concerne la maîtrise du concept de contrat de service. Il correspond à l'émergence d'un ensemble de services, en général les services CRUD (cf. axe Interopérabilité).

La deuxième capacité concerne l'apprentissage de la composition de services en utilisant des outils d'Injection de Dépendance tels que SPRING[1]. On parlera de composition « locale » de service puisque ce type d'outil ne sait composer que des services Java.

La troisième capacité concerne la mise en place d'une infrastructure de gestion de processus métier SOA, incluant la prise en compte de la dimension humaine.

La mise en place de processus métier faisant appel à des systèmes existants peut induire l'utilisation d'une messagerie asynchrone, ou MOM, ou l'interfaçage avec un EAI asynchrone existant. La quatrième capacité porte donc sur la maîtrise de l'asynchronisme dans une architecture SOA.

La cinquième capacité concerne la maîtrise de la composition généralisée de services, maîtrise permettant in fine la mise en place d'une taxonomie de services de grain plus ou moins fin. Cette maîtrise implique la mise en place d'un atelier de composition de services orienté SCA.

1. Cf. la présentation de la relation entre SPRING et SCA, partie 4, chapitre 14.

La sixième capacité concerne la mise en place d'un outil de réplication + agrégation de type EII ou MDM[1]. Cette capacité est optionnelle, car elle dépend de l'architecture SOA à mettre en place, elle-même dépendante de l'urbanisation du SI.

La septième capacité concerne l'usage d'un moteur de règles (BRMS[2]) basé sur le référentiel MDM et permettant de constituer une chaîne d'agilité intégrée depuis le BPM.

### *Axe Gouvernance*

La première capacité à acquérir sur cet axe concerne l'apprentissage de la notion de SLA. Cette notion sera déployée dans un premier temps au sein du Système d'Information de l'entreprise.

La deuxième capacité concerne la mise en place d'un premier niveau de SAM. Cela peut passer par l'obtention de traces de performance permettant d'analyser et corriger les problèmes survenus en production.

La troisième capacité concerne la mise en place de SLA entre l'entreprise et ses clients : à ce niveau, un SLA n'est plus uniquement un concept Technique mais également un concept Business.

La quatrième capacité concerne la mise en place d'un portail SAM. L'idée de base est de permettre un accès unifié à un ensemble de statistiques de performances. Un tel outil permet aux responsables d'anticiper sur d'éventuels problèmes en détectant des dégradations plus ou moins progressives des performances de production.

La cinquième capacité concerne la mise en place de capacités de BAM (intégrant un mode événementiel de capture de l'ensemble des indicateurs de performance métiers placés sur les processus métiers), afin d'avoir le même niveau de consolidation d'information que pour le SAM, mais cette fois pour les métiers. Ces capacités peuvent s'intégrer au portail SAM précédemment évoqué, en fonction des outils utilisés et des utilisateurs ciblés.

Un SLA peut évoluer au cours du temps. Par ailleurs un même service peut avoir simultanément plusieurs niveaux de SLA selon le client du service. La sixième capacité vise à maîtriser le cycle de vie des SLA notamment par l'évaluation systématique de l'impact d'un changement de SLA d'un service sur les solutions utilisatrices.

La généralisation de la démarche SOA, la multiplication des processus automatisés, l'accès aux référentiels ou la duplication des informations, impliquent une montée en charge de l'infrastructure SOA : la septième capacité a donc pour objectif de pouvoir anticiper sur cette montée en charge pour éviter tout problème en production. Cette capacité peut impliquer la possibilité de simuler les processus métier avant leur déploiement.

1. Cf. partie 3, chapitre 7, pour l'introduction à cette problématique, et partie 7, chapitre 20, pour la description de ces outils.
2. BRM (Business Rules Management System).

## En résumé

L'adoption d'une démarche SOA implique la mise en place d'un programme SOA, englobant Solution(s) Métier, Services, Architecture et Infrastructure. Une réflexion sur la planification et l'organisation de ce programme s'impose alors.

Cette adoption ne pourra se faire que progressivement : d'où l'utilité d'évaluer la maturation de cette adoption en utilisant un modèle de maturité basé sur les capacités : PSAUMM (*Practical, Service-oriented Architecture, Unified Maturity Model*).

Ce modèle décrit la maturité de l'architecture SOA progressivement mise en place, et est donc parfaitement complémentaire du modèle CMMI.

Enfin, il est nécessaire de réfléchir à la productivité des développements : il n'est pas possible de sacrifier la productivité pour rechercher les bénéfices de l'approche SOA en terme de flexibilité des processus métier et de réutilisation de l'existant.

# QUATRIÈME PARTIE

# La boîte à outils Web Services

Comme on l'a vu dans les précédentes parties, SOA est une approche d'architecture et ne fait pas d'hypothèse sur la technologie de mise en œuvre. En particulier, l'amalgame souvent fait entre SOA et les Web Services est une erreur de compréhension de SOA.

Cependant, la conception des spécifications Web Services a été menée dans l'objectif de répondre au mieux aux enjeux des architectures SOA. Ainsi, les Web Services sont une technologie très pertinente pour mener une démarche SOA.

L'objectif de cette partie est de parcourir les différentes spécifications qui pourront être mises en œuvre dans le cadre d'une architecture SOA basée sur les Web Services.

- Le chapitre 12 présente les spécifications de base des Web Services : SOAP, WSDL, UDDI.
- Le chapitre 13 présente les spécifications permettant de gérer la sécurité, la garantie d'acheminement, la garantie transactionnelle, le monitoring des services.
- Le chapitre 14 présente les spécifications permettant de gérer les aspects composition, orchestration et présentation.

# LA PILE WEB SERVICES

**Rappels sur les organismes de normalisation**

Dans l'histoire de la normalisation des Web Services, le consortium W3C (*World Wide Web Consortium*) a bâti les fondations : il a normalisé SOAP, WSDL et UDDI, normes présentées au chapitre 12.
La mise en œuvre des Web Service a alors fait apparaître des besoins de spécifications plus complètes afin de pouvoir adresser l'ensemble de la pile Web Service présentée dans le paragraphe suivant.
Le consortium W3C n'a pas voulu prendre en charge l'écriture de l'ensemble de ces spécifications, c'est pourquoi l'OASIS (*Organization for the Advancement of Structured Information Standards*), organisme de normalisation initialement chargé de la gestion de la norme documentaire SGML[a], a repris le flambeau. L'OASIS a pris en charge l'évolution de UDDI, tandis que le W3C conservait la gestion de SOAP et WSDL. Puis l'organisme a commencé à travailler sur l'écriture des nouvelles spécifications nécessaires à la complétion de la pile Web Services.
Sont alors apparus de nombreux regroupements d'acteurs de l'informatique qui écrivaient des spécifications, parfois contradictoires entre elles, et les proposaient parfois, mais pas toujours, à l'OASIS pour ratification. Ces initiatives individuelles expliquent le foisonnement qui règne autour des spécifications WS-*.
Parmi ces regroupements d'éditeurs, il faut signaler le partenariat d'IBM et Microsoft autour d'une vision commune de l'architecture des Web Services, appelée GXA, *Global XML Architecture*. Les propositions de GXA sont indépendantes de l'OASIS mais elles ont un poids important de part le soutien d'IBM et Microsoft. Par ailleurs, Microsoft a décidé de proposer une implémentation de référence de GXA dans un framework.NET appelé WSE, *Web Services Enhancements*.
Dans cette partie, on a pris le parti de présenter les spécifications issues du W3C, de l'OASIS et de GXA. On s'est aussi arrêté sur le travail de la Liberty Alliance, essentiel dans le cadre de la fédération d'identité.
On a ignoré de nombreuses spécifications plus exotiques et dont l'avenir est incertain. Leur dénombrement complet serait en effet une tâche titanesque et peu intéressante pour le lecteur.

a. SGML est l'ancêtre de XML. L'OASIS était donc pertinent sur le sujet Web Services.

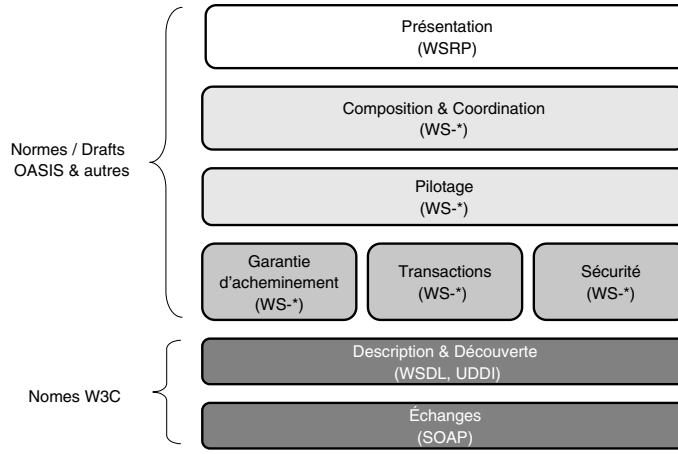
La figure suivante présente la pile des grammaires Web Services regroupées en couches.

Les cinq couches sont les suivantes :

- Les normes fondamentales pour la mise en œuvre des Web Services (SOAP, WSDL, UDDI).
- Les spécifications qui répondent aux exigences de sécurité, de garantie d'acheminement et d'intégrité transactionnelle.
- Les spécifications qui permettent le pilotage et la supervision des services.

- Les spécifications permettant la coordination ou la composition des services.
- Les spécifications qui permettent la présentation des services.

Cette représentation repose sur la vision GXA de Microsoft et IBM. Les couches gris clair (sécurité, garantie d'acheminement, intégrité transactionnelle, supervision, coordination/composition) de cette figure sont les plus récentes et les moins stabilisées.




La pile des grammaires Web Services

La figure fait apparaître les normes écrites par le Consortium W3C, aujourd'hui stabilisées et reconnues par tous les acteurs du monde informatique. Elle présente aussi les spécifications en cours d'écriture par divers groupes de normalisation pas toujours d'accord entre eux. Par conséquent, il est souvent risqué d'employer une de ces spécifications sans avoir évalué sa compatibilité avec les autres.

La maturité et le caractère opérationnel de chacune de ces spécifications seront évoqués dans les chapitres correspondants.

# 12

# L'infrastructure de base

## Objectif

Ce chapitre aborde les couches basses des Web Services (cf. figure 12.1), c'est-à-dire :

- La communication avec SOAP.
- La description d'un contrat de service avec WSDL.
- L'annuaire des services UDDI.

Pour chacune d'entre elles, il présente d'abord les objectifs qui ont piloté l'écriture des spécifications, puis l'évolution des usages consécutifs aux retours terrains.

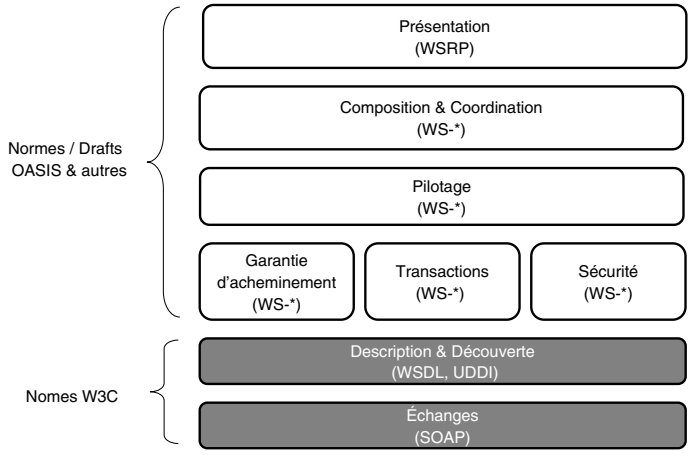



**Figure 12.1** — Les couches basses de la pile Web Services

# 12.1 IMPLÉMENTER LES COMMUNICATIONS SERVEUR À SERVEUR AVEC SOAP

## 12.1.1 XML-RPC, ancêtre de SOAP

XML-RPC[1] date de 1995. C'est une technologie d'invocation de service à distance basée sur XML et HTTP, donc indépendante du langage d'implémentation.

Son mode de fonctionnement consiste en l'invocation à distance d'une méthode de classe objet. Pour mener à bien cette invocation dans un environnement Internet, la signature de méthode est traduite en XML et les paramètres sont passés selon le même principe.

Si l'on considère le service de demande d'intervention présenté dans le fil rouge, la requête XML-RPC permettant de demander l'état d'avancement d'une demande prendra ainsi la forme suivante :

```
<?xml version="1.0" ?>
<methodCall>
     <methodName>ServiceIntervention.recupererEtatDemande</methodName>
     <params>
          <param>
               <value><i4>42</i4></value>
          </param>
          <param>
               <value><i4>108</i4></value>
          </param>
     </params>
</methodCall>
```

On retrouve entre les balises *<methodName>* le nom de la méthode permettant de récupérer l'état de la demande d'intervention et entre les balises *<value>* la valeur de l'identifiant de la demande (*<i4>* signifie que le paramètre est ici un entier).

XML-RPC connaît une dizaine de types très primitifs, ce qui en fait une grammaire simple à appréhender, mais très rudimentaire en terme de capacité d'invocation.

Par ailleurs, la technologie ne propose pas de notion de description de l'interface du service : ainsi les 2 parties impliquées dans l'invocation doivent se concerter pour se mettre d'accord sur l'adresse du service, ses méthodes, ses paramètres, etc.

De plus, il n'y a pas non plus de normalisation des erreurs dans XML-RPC.

Cette technologie est donc utilisée aujourd'hui dans des cas très simples, mais inadaptée à des architectures distribuées complexes.

1. RPC signifie *Remote Procedure Call*.

### 12.1.2 Les principes de SOAP

SOAP a été conçu à partir de 1998, afin d'étendre les possibilités de XML-RPC. L'acronyme signifie *Simple Object Access Protocol*, c'est-à-dire protocole simplifié pour l'accès à des objets distants. SOAP a été pensé d'une part pour être indépendant de l'implémentation technique du service, d'autre part pour passer au travers des firewalls, et ainsi permettre l'invocation de services situés dans des Systèmes d'Information partenaires.

SOAP gère l'échange de messages XML : c'est la couche basse des architectures Web Services. Ce protocole applicatif peut s'appuyer sur différents protocoles de transport pour des échanges synchrones ou asynchrones (cf. Partie 1) :

- **HTTP :** échanges synchrones;
- **Java RMI :** échanges synchrones;
- **.NET Remoting :** échanges synchrones;
- **SMTP :** échanges asynchrones;
- **FTP :** échanges asynchrones;
- **Java JMS :** échanges asynchrones;
- **.NET MSMQ :** échanges asynchrones;
- etc.

Par ailleurs, SOAP désigne les informations échangées dans une syntaxe correspondant aux fonctions applicatives, contrairement à XML-RPC qui utilise uniquement des nommages techniques et primitifs.

De plus, SOAP permet la sérialisation/désérialisation de tout objet métier, sous une forme XML utilisable quelle que soit la technologie d'implémentation. Enfin, SOAP normalise la gestion des erreurs.

### 12.1.3 Fonctionnement de SOAP

Le message SOAP est composé de deux parties :

- La partie **Header** porte des informations complémentaires pour le traitement des données (identification de l'émetteur du message, règles de sécurité pour la lecture du message, algorithme de chiffrement à utiliser pour la lecture du message, etc.).
- La partie **Body** porte les données propres au message, et matérialise la requête ou la réponse SOAP (structure de données spécifique).

L'exemple suivant présente, dans le cadre du fil rouge, la demande d'avancement sur une intervention en SOAP :

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
     <soap:Body>
          <recupererEtatDemande>
               <NumeroDistributeur>42</NumeroDistributeur>
               <NumeroDemande>108</NumeroDemande>
          </recupererEtatDemande >
     </soap:Body>
</soap:Envelope>
```

On constate dans l'exemple que la lecture du message SOAP est relativement aisée par un humain grâce à la souplesse du langage XML qui permet un nommage proche des fonctionnalités métiers.

La figure 12.2 récapitule le fonctionnement de SOAP.

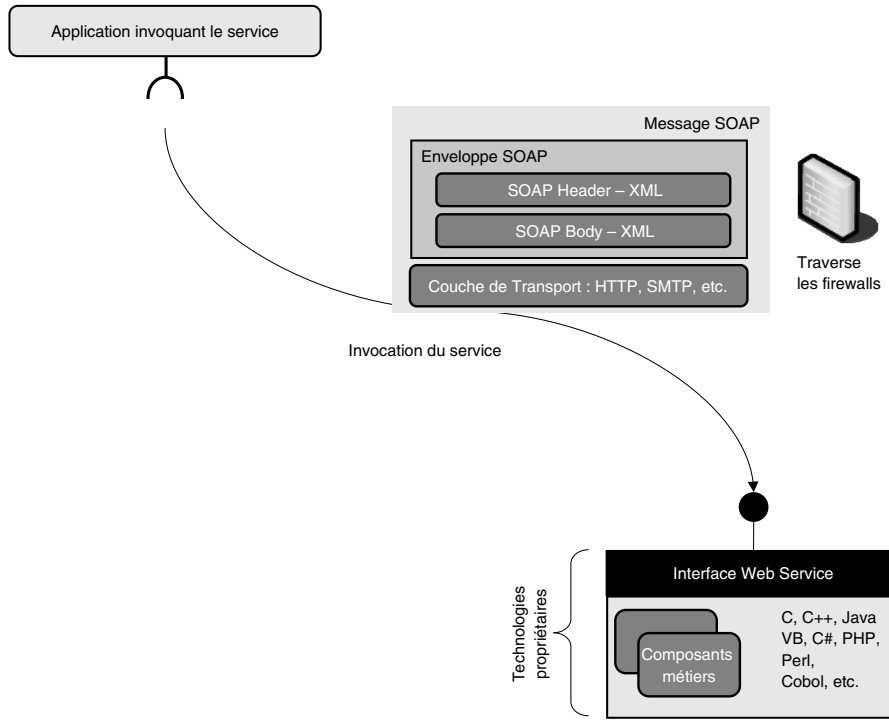



**Figure 12.2** — Fonctionnement de SOAP

## 12.1.4 Les limites de SOAP

Si les messages XML utilisés dans SOAP sont facilement lisibles par des humains, ils sont assez verbeux ce qui entraîne des problématiques en terme de performance :

- D'une part, le poids d'un message SOAP (codé en mode texte) est beaucoup plus important qu'un message codé en binaire, ce qui a un impact sur son temps d'acheminement.

- D'autre part, le travail de construction et de lecture du message par les applications impliquées dans la communication est gourmand en terme de temps processeur.

Lorsque HTTP est le protocole de transport utilisé avec SOAP, on peut résoudre cette problématique de performance avec des boîtiers accélérateurs XML. Le rôle de ces boîtiers est de décharger les middlewares d'intégration des tâches de gestion des flux XML. Compte tenu du coût d'une telle solution, on préfère souvent remplacer HTTP par un protocole propriétaire mais plus performant comme RMI avec Java. Cette stratégie n'est possible que lorsque la problématique d'interopérabilité n'est pas présente, c'est-à-dire à l'intérieur du SI.

Par ailleurs, certains développeurs reprochent à SOAP la complexité de sa syntaxe et se tournent vers du « Plain Old XML » ou XML simple et basique. Ces personnes mettent alors en œuvre XML-RPC ou REST.

### 12.1.5 REST : un concurrent de SOAP pour une intégration légère

REST signifie *REpresentational State Transfer*. Il désignait à l'origine un principe d'architecture basé sur la distribution de médias liés entre eux par des hypertextes (à la manière du Web). Aujourd'hui, le terme est employé pour désigner une architecture d'intégration de services allégée.

REST est en effet beaucoup plus simple que SOAP. Il utilise les différentes méthodes du protocole HTTP (GET, POST, PUT, et DELETE) pour consulter ou mettre à jour des données distantes. Ainsi le nombre de verbes (méthode du protocole applicatif) n'est pas extensible contrairement à SOAP mais directement lié aux verbes du protocole de transport.

REST n'est pas à proprement parler un protocole d'invocation à distance : il s'agit essentiellement d'un protocole destiné à des services CRUD (cf. Partie 3) utilisant des messages XML pour l'accès à des ressources distantes.

Par ailleurs, REST affecte une adresse URI[1] à chacune des ressources à accéder. De ce fait, les URL sont particulièrement lisibles et les requêtes sont fortement simplifiées.

L'exemple suivant présente, dans le cadre du fil rouge, la demande d'avancement sur une intervention en REST :

```
http (methode GET)://www.portos.org/ServiceIntervention/recupererEtatDemande/
NumeroDistributeur/42
NumeroDemande/108
```

REST est largement utilisé pour les intégrations au niveau de l'interface utilisateur dans des services sur Internet. Ces services comme Flickr ou del.icio.us[2] sont

1. URI signifie Uniform Resource Identifier. Il s'agit d'un format d'adressage sur le Web.
2. On les regroupe parfois sous l'appellation Web 2.0.

destinés à permettre à des internautes de partager des informations entre plusieurs applications. Leurs besoins en intégration sont donc très simples.

REST est donc destiné à des développeurs de sites Web : en aucun cas, cette technologie ne peut être utilisée pour des compostions de services au sein d'applications métiers complexes. Ce n'est donc pas un concurrent sérieux pour SOAP dans le cadre des architectures SOA.

Enfin, REST, de même que XML-RPC, ne répond pas au besoin de description de l'interface du service (cf. Partie 2).

# 12.2 DÉFINIR UN CONTRAT DE SERVICE AVEC WSDL

## 12.2.1 Fonctionnement de WSDL

WSDL signifie *Web Service Description Langage*. Comme son nom l'indique, il s'agit d'un langage XML normalisé pour décrire le mode de fonctionnement d'un Web Service.

Il permet ainsi de décrire les modalités d'invocation distante d'un Web Service, en particulier :

- Les opérations possibles au sein du service.

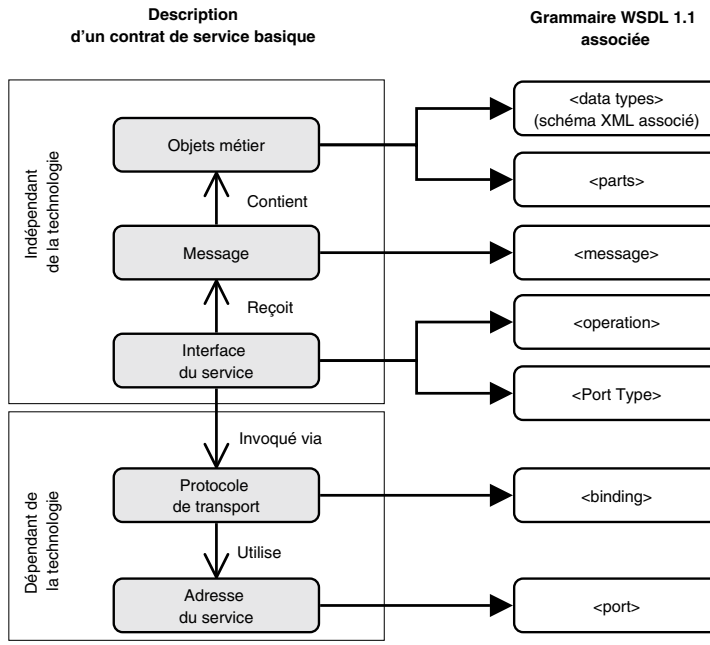



**Figure 12.3** — Les éléments de la grammaire WSDL

- Les paramètres d'entrée et sortie de ces opérations.
- Le typage de ces paramètres.
- Les points d'entrée (URL) des opérations.

La figure 12.3 montre comment WSDL traduit ces notions dans sa grammaire normalisée.

Les concepts sous-jacents sont présentés en détail dans la seconde partie de cet ouvrage. WSDL ne couvre que le contrat « minimal » décrit dans cette partie, c'est-à-dire la seule méthode d'invocation et non un engagement sur la qualité de service.

WSDL constitue un contrat pour l'invocation technique de service Web, sur lequel les applications clientes peuvent s'appuyer pour connaître les opérations et leurs arguments. Ainsi chaque application pourra paramétrer de manière automatique cette invocation de service.

L'exemple suivant présente, dans le cadre du fil rouge, la définition en WSDL 1.1 du service de demande d'avancement sur une intervention. On considère que le service, intitulé *ServiceIntervention*, a comme paramètre d'entrée un numéro de demande entier, et comme paramètre de sortie un état d'avancement de la demande sous la forme d'une chaîne de caractères.

```
     <?xml version="1.0"?>

<!— root element wsdl:definitions defines set of related services —>

<wsdl:definitions xmlns:si="http://www.portos.com/ServiceIntervention"
xmlns:bo="http://www.portos.com/bo" xmlns:soap="http://schemas.xmlsoap.org/
wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" targetNamespace="http://www.portos.com/
ServiceIntervention" name="ServiceIntervention">
     <wsdl:types>
          <xsd:schema targetNamespace="http://www.portos.com/bo">
               <xsd:element name="recupererEtatDemande">
                    <xsd:annotation>
                         <xsd:documentation>le type wrapped de recupération de
l'état de la demande</xsd:documentation>
                    </xsd:annotation>
                    <xsd:complexType>
                         <xsd:sequence>
                              <xsd:element name="NumeroDemande" type="xsd:int"/>
                              <xsd:element name="NumeroDistributeur" type="xsd:int"/>
                         </xsd:sequence>
                    </xsd:complexType>
               </xsd:element>
               <xsd:element name="recupererEtatDemandeResponse">
                    <xsd:annotation>
                         <xsd:documentation>le type wrapped de la réponse</xsd:
documentation>
                    </xsd:annotation>
```

```
                    <xsd:complexType>
                         <xsd:sequence>
                              <xsd:element name="EtaDemande" type="xsd:string"/>
                         </xsd:sequence>
                    </xsd:complexType>
               </xsd:element>
          </xsd:schema>
     </wsdl:types>
     <wsdl:message name="recupererEtatDemande">
          <wsdl:part name="parameters" element="bo:recupererEtatDemande"/>
     </wsdl:message>
     <wsdl:message name="recupererEtatDemandeResponse">
          <wsdl:part name="parameters" element="bo:recupererEtatDemande-
Response"/>
     </wsdl:message>
     <wsdl:portType name="ServiceInterventionPort">
          <wsdl:operation name="recupererEtatDemande">
               <wsdl:input message="si:recupererEtatDemande"/>
               <wsdl:output message="si:recupererEtatDemandeResponse"/>
          </wsdl:operation>
     </wsdl:portType>
     <wsdl:binding name="ServiceInterventionBinding" type="si:ServiceIntervention-
Port">
          <soap:binding style="document" transport="http://schemas.xmlsoap.org/
soap/http"/>
          <wsdl:operation name="recupererEtatDemande">
               <wsdl:input>
                    <soap:body use="literal"/>
               </wsdl:input>
               <wsdl:output>
                    <soap:body use="literal"/>
               </wsdl:output>
               <soap:operation soapAction="urn:#recupererEtatDemande"/>
          </wsdl:operation>
     </wsdl:binding>
     <wsdl:service name="ServiceIntervention">
          <wsdl:port name="ServiceInterventionPort" binding="si:Service-
InterventionBinding">
               <soap:address location="http://services.portos.com/wsdl/
ServiceIntervention"/>
          </wsdl:port>
     </wsdl:service>
</wsdl:definitions>
```

**Nota** : l'exemple présenté respecte la spécification WSDL 1.1 et non la spécification WSDL 2.0. En effet, WSDL 1.1 fait partie des normes d'interopérabilité WS-I (cf. partie 5).

L'exemple montre bien le caractère verbeux de WSDL. Cependant, il est dans la pratique assez rare d'écrire le corps du fichier à la main : la plupart des environnements de développement savent en effet générer le fichier WSDL sur la base des caractéristiques du service développé. Certains autres permettent de le construire sur une base visuelle (cf. figure 12.4).

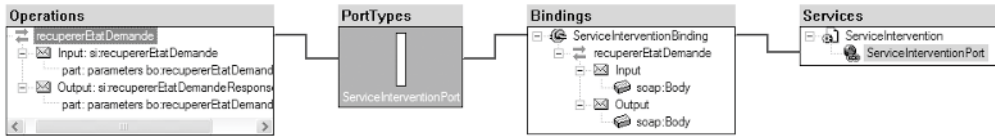



**Figure 12.4** — Construction de contrat WSDL au travers d'un outil WYSIWYG

### 12.2.2 Les limites de WSDL

Comme précisé dans le paragraphe précédent, WSDL ne décrit dans sa version actuelle que le contrat « minimal », c'est-à-dire la mécanique d'invocation du service. Il ne permet pas de décrire (cf. partie 2) :

- La plaquette commerciale du service.
- Les engagements vis-à-vis du service (SLA).
- La tarification éventuelle du service.
- Etc.

Un certain nombre de spécifications additionnelles en cours d'écriture visent à combler ces manques :

- **WS-agreement (initiée par IBM et soumise au W3C) :** cette grammaire XML vise à définir les garanties s'appliquant sur un contrat de service. Elle aborde la description des parties en présence, du niveau de service rendu, des garanties apportées (disponibilité, temps d'inactivité, etc.), du mode de contrôle de la disponibilité des services.
- **WSMO, *Web Service Modeling Ontology* (ESSI[1]) :** cette initiative vise à créer des grammaires XML pour décrire des services dans un domaine métier particulier, afin de pouvoir comparer les offres de divers vendeurs (par exemple décrire les offres de vente de musique en ligne).

La pérennité des spécifications citées ci-dessus n'est pas assurée : elles sont intéressantes uniquement pour avoir une vision prospective des axes d'évolution des contrats de service.

Enfin, il existe par ailleurs une spécification plus généraliste et plus mature qui peut être utilisée pour décrire les directives s'appliquant sur un contrat de service : il s'agit de WS-Policy. Elle sera présentée dans le chapitre 13.

1. *European Semantic Systems initiative*.

# 12.3 DÉCOUVRIR DES SERVICES AVEC LE REGISTRE UDDI

## 12.3.1 Principes de UDDI

UDDI signifie *Universal Description, Discovery, and Integration*. Comme son nom l'indique, il s'agit d'un système d'annuaire qui permet à un fournisseur de décrire son Service, puis au client de découvrir le Service et de s'y connecter. UDDI est à l'origine une norme du W3C écrite en 2000. L'OASIS a repris sa gestion et a sorti la version 3 en 2004.

Un annuaire UDDI est construit sur la base des normes WSDL et SOAP. Ainsi les services y sont décrits en WSDL et on accède à l'annuaire via des requêtes SOAP. On y trouve trois types d'informations :

- **Les Pages Blanches**, qui décrivent les fournisseurs de services.
- **Les Pages Jaunes**, qui décrivent les catégories de services.
- **Les Pages Vertes**, qui décrivent les contrats WSDL.

UDDI complète donc le contrat WSDL par des méta-données sur le fournisseur du service.

UDDI a été conçu pour permettre la découverte et l'intégration automatique d'un service. Par exemple, un portail paramétré pour intégrer un service de météorologie peut, si son service habituel ne répond pas, aller en chercher un autre dans un annuaire UDDI et s'y connecter sans aucune intervention humaine (cf. figure 12.5).

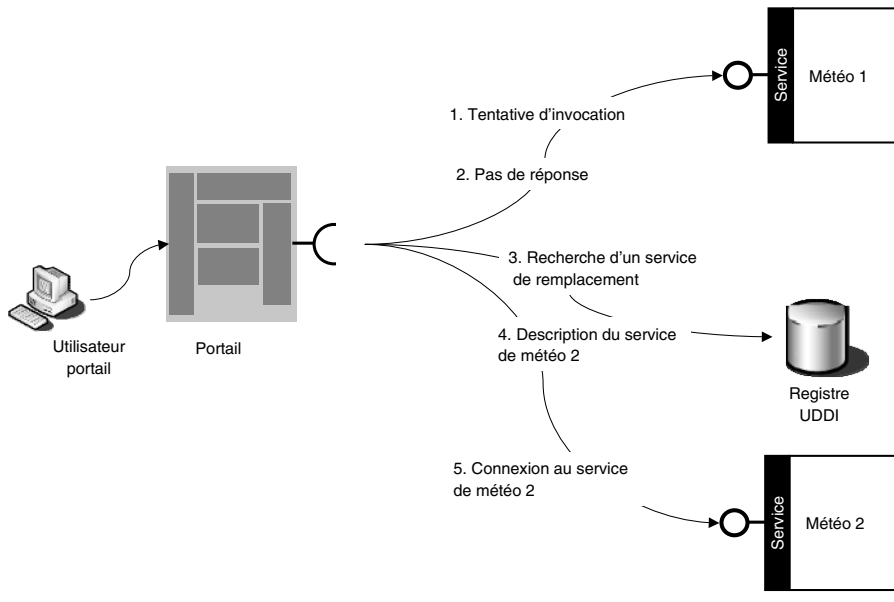



**Figure 12.5** — Exemple de découverte et intégration automatique

Ces aspects sont décrits en détail dans la partie 2 de cet ouvrage.

Les annuaires UDDI peuvent être autonomes ou se fédérer en réseaux de registres à la manière des DNS ou des autorités de certification. Ils peuvent être privés, publics ou semi-publics, c'est-à-dire accessibles uniquement par des partenaires.

On distingue ainsi trois types de registres :

- **Registre d'entreprise :** ces annuaires sont situés sur un réseau privé, et servent de référentiels des services disponibles en interne. C'est aujourd'hui le principal usage des annuaires UDDI.
- **Registre fédéré avec des partenaires :** ces annuaires peuvent avoir une gestion déléguée à un tiers ou bien être synchronisés avec d'autres référentiels. Leur architecture doit être soignée pour assurer leur sécurité : ils peuvent en effet être la cible de pirates.
- **Registre public :** ces annuaires devraient théoriquement être liés les uns aux autres à la manière de DNS afin de constituer un registre mondial des services invocables depuis Internet. Dans la pratique, ils sont inexistants : seuls quelques annuaires de tests sont proposés par Microsoft, IBM et Xmethods.

### 12.3.2 Aspects techniques

Le fournisseur de service publie son contrat WSDL dans l'annuaire UDDI en l'associant à la description de la société et à la catégorie de service concernée.

La description des catégories de services n'est pas normalisée : elle est laissée à l'appréciation du fournisseur de registre, ce qui limite la possibilité de registre interopérable à l'échelle d'Internet. On retrouve ici la problématique d'ontologie, évoquée dans le paragraphe 12.2.2 Les limites de WSDL.

Chaque service de l'annuaire est identifié par une clef unique, sorte de clef primaire du registre.

UDDI normalise la notion d'affiliation entre les registres, permettant la mise en œuvre d'architectures fédérées entre annuaires de plusieurs partenaires.

Avec la version 3 de la norme, il est possible de signer électroniquement un contrat WSDL lors de sa publication afin de certifier qu'il émane bien d'une société connue et non d'un pirate malveillant.

### 12.3.3 Les limites de UDDI

On fait aujourd'hui un constat d'échec sur la découverte et l'intégration automatiques qui étaient un des fondements de la conception de UDDI. Les concepteurs de la norme avaient en effet sous-estimé la nécessité de négociation financière et de contractualisation qui empêchent toute idée d'automatisation de l'accès à un service payant. De plus, avec les nouvelles réglementations de type Sarbanes-Oxley, il n'est pas envisageable de se connecter à un prestataire dont on ignore tout. L'objectif

initial de mettre en œuvre un annuaire de services à l'échelle d'Internet n'a donc pas été atteint.

Cependant, dans le contexte d'un système d'information s'orientant vers une architecture SOA, il est tout à fait pertinent de mettre en œuvre un registre interne des services. Ce registre constitue un référentiel d'entreprise au même titre qu'un annuaire LDAP ou une base de clients. Il permet de décrire et localiser les services avec leurs différentes versions, jouant un rôle complémentaire à l'intranet de la DSI souvent utilisé pour décrire les infrastructures techniques de l'entreprise.

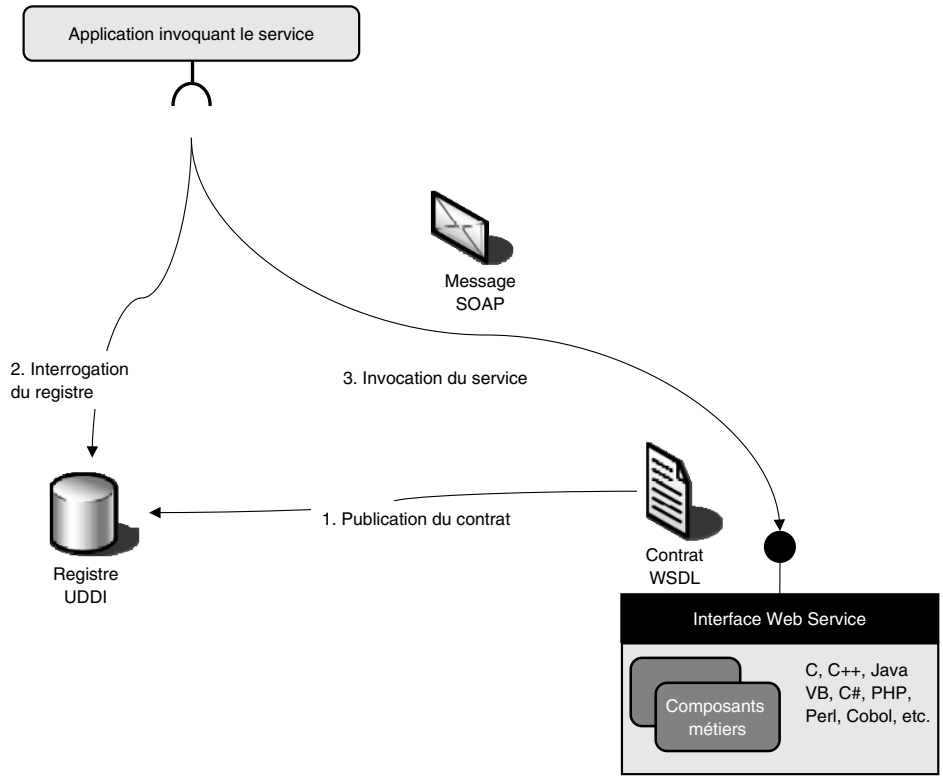



**Figure 12.6** — L'infrastructure de base des Web Services

De plus, il existe, dans le cadre du framework WS-Policy, une spécification appelée WS-PolicyAttachment qui permet d'ajouter une politique d'accès à un service donné (cf. paragraphe sur WS-Policy).

L'annuaire UDDI peut donc, en intégrant cette description, gérer la politique d'accès aux services de l'entreprise. Il prend ainsi un rôle structurant dans l'infrastructure de sécurité de l'entreprise. Certains analystes y voient le vrai intérêt des registres UDDI.

## En résumé

La figure 12.6 propose un récapitulatif des normes présentées dans ce chapitre. Elle positionne :

- SOAP, pour l'échange des messages d'invocation des services ;
- WSDL, pour la description de contrat de services, quelle que soit la technologie d'implémentation sous-jacente ;
- UDDI, comme référentiel des services disponibles.

# 13

# Les réponses aux exigences techniques

## Objectif

Ce chapitre aborde la seconde couche des spécifications Web Services présentées en introduction de cette partie (cf. figure 13.1).

Il décrit les spécifications permettant :

- La sécurité sur la base du socle WS-Security.
- La garantie d'acheminement.
- La garantie transactionnelle.
- Le monitoring des services.

Ces spécifications correspondent au cahier des charges technique de la première partie.

### *En introduction, une norme généraliste*

#### *WS-Policy et WS-Policy Attachment (GXA)*

Le framework WS-Policy permet de définir toute politique associée à un service. Il offre une grammaire normalisée pour décrire des politiques de sécurité, de garantie de service (SLA) ou autre. Il est donc destiné à enrichir la description des services au sein des annuaires UDDI.

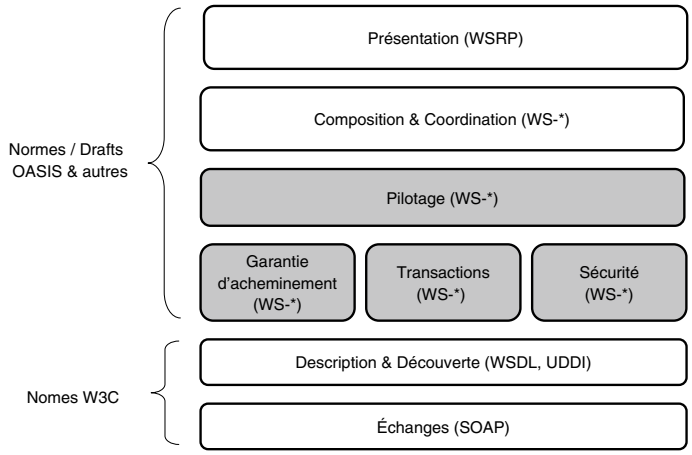



**Figure 13.1** — La seconde couche de la pile Web Services

Cette spécification ne rentre donc pas dans l'un des sous groupes de ce chapitre (sécurité, transaction, etc.). Par contre, on y fera référence dans plusieurs d'entre eux.

WS-Policy ne précise pas comment lier la politique à un service donné. A contrario, WS-Policy Attachment se présente sous la forme d'un fichier séparé qui permet de lier la politique au service.

# 13.1 GÉRER LA SÉCURITÉ[1]

## 13.1.1 La sécurité dans le cadre des Web Services

La sécurisation des Web Services est possible avec les méthodes traditionnelles notamment la mise en œuvre de tunnels SSL pour assurer la confidentialité des appels Web Services ou l'utilisation d'identifiant/mot de passe pour assurer l'authentification d'une application cliente d'un web service. Elle est toute indiquée pour des échanges point à point.

Cependant, dans le cadre d'échange de messages SOAP entre plus de deux services, le nombre de tunnels SSL et points d'authentification devient exponentiel : en effet, pour échanger des messages entre quatre services, il faudra déployer quatre certificats SSL et authentifier chacun des trois autres services sur chaque service invoqué (voir figure 13.2). Ce mode de fonctionnement devient vite ingérable, c'est pourquoi on préfère intégrer la gestion de la sécurité au sein des messages SOAP.

Ce chapitre a pour objectif de présenter les spécifications permettant la sécurisation des Web Services au travers de méthodes basées sur les messages SOAP eux-mêmes.

1. Pour un rappel général sur les notions de sécurité, voir partie 1.

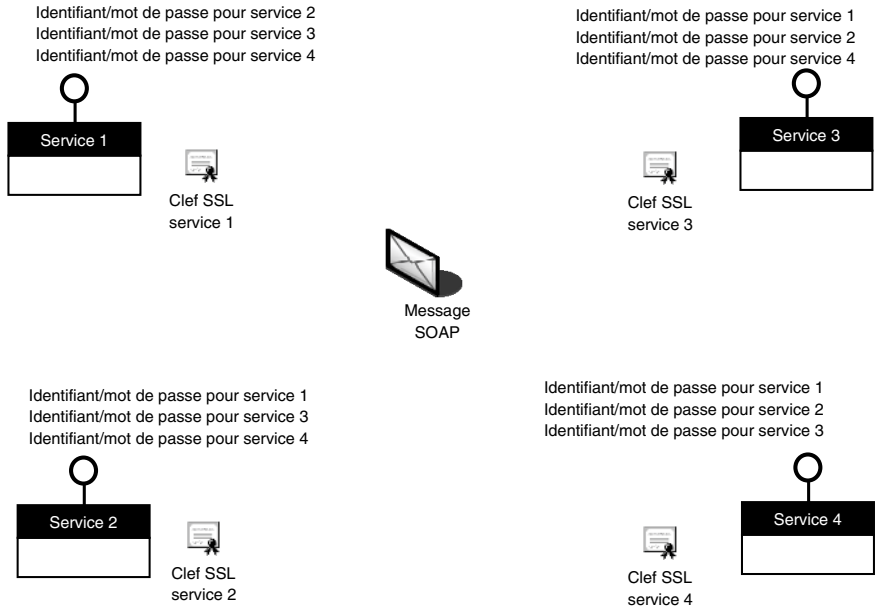



**Figure 13.2** — Sécuriser les Web Services avec les méthodes traditionnelles

## 13.1.2 Le framework WS-Security

WS-Security ou WSS constitue un framework de base pour la sécurisation des Web Services. Il s'appuie sur un certain nombre de sous-normes, présentées dans la suite de ce paragraphe.

### *XML Encryption (W3C)*

*XML Encryption* est une spécification du W3C qui permet d'assurer la confidentialité des informations en chiffrant un sous-ensemble d'un message XML. Son mode de fonctionnement est le suivant : on applique des mécanismes de chiffrement à clef secrète ou à clef publique sur une partie du message SOAP.

Il devient ainsi possible, lors de la transmission de messages XML, d'assurer la confidentialité d'une partie des données vis-à-vis de certains services.

XML *Encryption* fournit une grammaire XML standardisée pour décrire les méthodes de chiffrement des données. Elle permet de mettre en œuvre des procédures automatisées pour assurer la confidentialité des échanges.

### *XML Signature (W3C)*

XML Signature est une autre spécification du W3C qui permet d'assurer la non-répudiation d'un message SOAP en appliquant une signature numérique sur un sous-ensemble d'un flux XML.

Il devient ainsi possible, lors de la transmission de messages SOAP, d'authentifier l'expéditeur, d'assurer l'intégrité des données et d'assurer la non-répudiation des échanges.

L'utilisation de XML Signature suppose bien entendu la mise en œuvre de procédures pour distribuer des certificats numériques aux services et de routines pour vérifier les signatures.

### *SAML (OASIS)*

SAML signifie *Security Assertion Markup Language*. C'est une grammaire XML normalisée par l'OASIS qui permet d'attester l'authentification d'un client souhaitant accéder à plusieurs services, en évitant ainsi à ce client de s'authentifier autant de fois qu'il y a de services invoqués. SAML permet ainsi d'assurer le *Single Sign On*, l'authentification étant assurée par un service technique dédié à cette tâche.

Une fois l'authentification effectuée, ce service de sécurité distribue aux autres services des *assertions* ou *jetons de sécurité* SAML, au travers desquels il certifie avoir bien mené l'authentification des accédants et s'engage sur leur identité. Il utilise pour cela des mécanismes de signature numérique et de chiffrement basés sur *XML Encryption* et *XML Signature*.

L'authentification au sein du service de sécurité peut être basée sur des identifiants/mots de passe, des tickets Kerberos, des certificats X509, des certificats PGP, etc.

### *WS-Security (OASIS) : un framework basé sur les spécifications précédentes*

WS-*Security* est initialement une proposition de Microsoft, IBM, et Verisign reprise par l'OASIS. C'est un framework de sécurisation des Web Services basé en particulier sur les standards décrits plus haut : *XML Encryption*, *XML Signature*, et SAML.

*WS-Security* propose de sécuriser de manière intrinsèque les messages SOAP :

- Assurer l'intégrité d'un fragment du message SOAP en le signant avec XML Signature.
- Assurer la confidentialité d'un fragment du message SOAP avec XML Encryption.
- Certifier l'identité de l'accédant auprès du serveur SOAP avec SAML ou Kerberos[1].

WSS décrit ainsi des scénarios d'échanges entre Web Services en y associant des pratiques de sécurisation. La figure 13.3 présente le framework WS-Security.

Ce framework est aujourd'hui standardisé et implémenté dans les serveurs d'application du marché en technologie JEE et .NET. Il est donc possible de le mettre en œuvre dans le cadre d'architecture SOA.

1. Kerberos n'est pas présenté dans ce chapitre : il s'agit un protocole permettant de déléguer l'authentification à un tiers à la manière de SAML. Cependant il n'est pas basé sur XML mais sur des messages spécifiques.

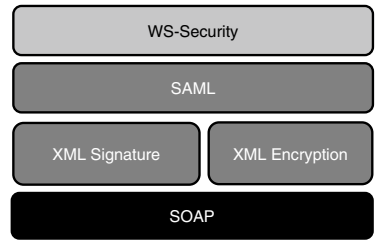



**Figure 13.3** — Le framework WS-Security

**Remarque :** WS-Security est aussi parfois appelé WSS ou encore SOAP Message Security. De fait, les variations de nommage des spécifications Web Services sont un facteur important de confusion pour leurs utilisateurs.

### 13.1.3 Les spécifications de sécurité moins abouties

Les spécifications présentées dans ce paragraphe s'appuient sur le framework WS-Security. Elles sont généralement en cours d'écriture et peu stabilisées.

Elles émanent de consortiums d'éditeurs : elles ne sont donc prises en charge par aucun organisme de normalisation. Par conséquent, leur implémentation est assurée par une minorité de serveurs d'applications.

#### *XACML (OASIS)*

XACML est une spécification gérée par l'OASIS et stabilisé depuis quelques années. Elle n'a donc pas tout à fait sa place dans la rubrique des spécifications inabouties. Cependant, elle n'est pas implémentée aussi fréquemment que les précédentes. Son usage est plutôt marginal, d'où sa présence dans ce chapitre.

XACML signifie *eXtensible Access Control Markup Language*. Son objectif est de compléter SAML en précisant les habilitations de l'accédant. Elle décrit donc des droits pour des accédants ou des groupes d'accédants vis-à-vis de services.

#### *SPML (OASIS)*

SPML signifie *Service Provisioning Markup Language*. Cette spécification a été proposée par WaveSet, éditeur racheté par Sun, puis ratifiée par l'OASIS.

Elle porte sur le *provisioning* des annuaires de sécurité, c'est-à-dire sur la gestion des comptes utilisateurs dans différentes applications disposant de bases d'identifiants/mot de passe autonomes.

SPML peut traiter trois cas d'usage classiques de provisioning :

- **Un collaborateur est embauché dans une société :** il faut alors créer ses comptes pour Windows, la messagerie, l'intranet, etc.

- **Un collaborateur change de statut dans la société :** il faut pouvoir modifier ses droits ou ses comptes vis-à-vis des applications.
- **Un collaborateur quitte la société :** il est nécessaire de désactiver l'ensemble de ses comptes applicatifs.

SPML est donc une grammaire XML normalisée qui permet d'échanger des messages de création/modification/suppression de compte entre des services.

Avec l'usage de SPML, on considère que la centralisation des comptes de sécurité n'est pas nécessaire et que l'on ne va pas à l'encontre de la multiplication des annuaires de sécurité. Cette approche est opposée mais complémentaire à celle de la fédération d'identité, présentée dans la suite de ce chapitre.

### *WS-SecurityPolicy (GXA)*

WS-SecurityPolicy est une déclinaison de WS-Policy dans le cadre des politiques de sécurité. WS-SecurityPolicy est utilisable dans le cadre de WS-Security, WS-SecureConversation et WS-Trust (cf. suite de ce chapitre).

WS-SecurityPolicy indique :

- Quels modes d'authentification sont acceptés par le service.
- Si le service délègue son authentification à un tiers, quels types d'assertions de sécurité il accepte (tickets SAML, Kerberos, ou autres).
- Si le service manipule des données confidentielles et exige des échanges sur un mode chiffré.

L'exemple ci-dessous présente le formalisme d'une politique de sécurité exigeant une authentification par identifiant/mot de passe (mot clef : UsernameToken).

```
<wsse:SecurityTokenReference>
<wsse:Reference
URI="#SecurityToken-c4dc7e55-7c3d-43ed-b615-0b62a14009c7" ValueType="http://
docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#UsernameToken"/>
</wsse:SecurityTokenReference>
```

### *WS-Trust (GXA)*

Le principe de WS-Trust est de demander à un service technique, le Security Token Service, de servir de relais de confiance entre deux services qui n'utilisent pas les mêmes méthodes d'authentification ou les mêmes typologies d'assertions de sécurité. Ainsi le Security Token Service doit être capable de :

- Traduire les assertions de sécurité d'un formalisme (par exemple SAML) à un autre (par exemple Kerberos) pour les services en présence.
- Servir de relais de confiance entre les services en présence, car ces derniers lui font tous deux confiance.

Le Security Token Service joue ainsi un rôle de tiers d'intermédiation entre deux univers qui ne parlent pas le même langage et où les règles de sécurité sont différentes.

Pour les lecteurs familiers avec la PKI, le Security Token Service est comparable à l'autorité d'enregistrement qui joue le rôle d'intermédiaire entre l'autorité de certification et les demandeurs de certificats.

Pour donner une image compréhensible par tous, on pourrait considérer deux pays (Israël et la Palestine) qui n'ont aucune relation de confiance, mais qui de façon ponctuelle peuvent s'appuyer sur la France, qui a des relations diplomatiques avec ces deux pays, pour se transmettre des informations fiables.

### *WS-SecureConversation (GXA)*

WS-SecureConversation définit une extension de WS-Security qui permet de mener une session sécurisée pour les échanges entre deux services. C'est en quelque sorte l'équivalent d'une session SLL en peer-to-peer entre Web Services.

WS-SecureConversation commence par une phase d'initialisation au cours de laquelle les deux partenaires se mettent d'accord sur les clefs de chiffrement à utiliser tout au long de la session (cf. SSL Handshake pour les connaisseurs en sécurité). Lors de longues sessions chiffrées, le protocole permet d'économiser de la complexité syntaxique par rapport à WS-Security.

De manière synthétique, WS-SecureConversation permet de simplifier les messages WS-Security lorsqu'on a affaire à des échanges longs et répétés entre deux services.

**À noter :** WS-SecureConversation peut s'appuyer sur WS-Trust.

### *WS-Privacy (GXA)*

WS-Privacy est un projet en cours d'écriture par un consortium d'éditeurs. Cette spécification aura pour objectif de définir une politique sur la gestion des données privées des clients/utilisateurs. Il s'agit de se mettre au clair vis-à-vis des utilisateurs et du législateur (la CNIL en France) sur la problématique épineuse de la conservation des données privées.

### *WS-Authorization (GXA)*

WS-Authorization est de même un projet en cours d'écriture. La grammaire XML sous-jacente n'est donc pas disponible.

Il semblerait que ce soit une spécification concurrente de XACML, donc vouée à définir des droits d'accès sur des services. Elle est proposée par IBM et Microsoft pour être plus cohérente avec l'ensemble des spécifications présentées dans ce chapitre.

### 13.1.4 Les spécifications pour la fédération d'identité

#### *Introduction à la fédération d'identité*

La fédération d'identité est une approche d'architecture qui propose d'établir des liens de confiance de manière distribuée entre des Services Applicatifs et des référentiels utilisateurs, appelés *serveurs d'identité*. Le serveur d'identité est une sorte d'annuaire LDAP amélioré, capable d'authentifier des accédants pour le compte d'applications internes ou externes à l'entreprise sur la base des technologies Web Services. Il fournit aussi des fonctions de *Single Sign On*.

Le principal objectif de la fédération d'identité est de faciliter les échanges entre partenaires sans avoir à dupliquer les référentiels utilisateurs et donc en faisant de la délégation de gestion des utilisateurs sur la base de liens de confiance entre entreprises.

La figure 13.4 illustre ce mode de fonctionnement :

- Lorsqu'un utilisateur A interne à l'entreprise A accède à un service A1, l'authentification est basée sur le serveur d'identité A interne.
- Lorsqu'un utilisateur B externe à l'entreprise A accède à un service A2, l'authentification est basée sur le serveur d'identité B de son entreprise.
- Lorsqu'un utilisateur B accède à un service A2 externe à son entreprise et un service B1 interne à son entreprise, les deux services d'appuient sur le serveur d'identité B de son entreprise. L'utilisateur B bénéficie ainsi de Single Sign On entre deux services émanant de deux structures différentes.

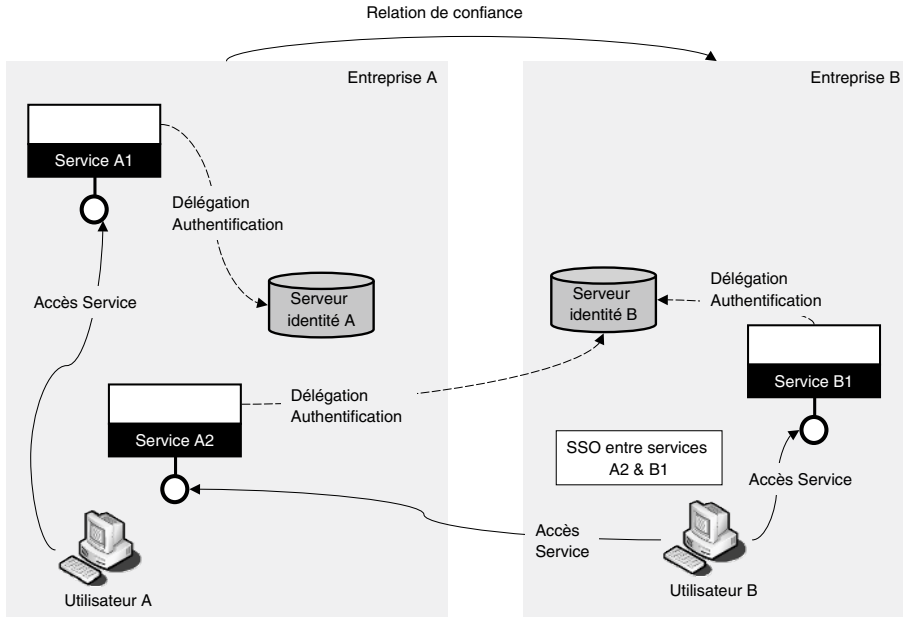



**Figure 13.4** — Fonctionnement de la fédération d'identité

Les modes d'interactions entre services et serveurs d'identité peuvent suivre de multiples cas d'usage (plus complexes que ceux de la figure 13.4) afin de faire face à tous les scénarios de délégation d'identité.

La fédération d'identité est un principe un peu déroutant au premier abord, mais il se révèle précieux lorsqu'on souhaite ouvrir ses services à des partenaires sans avoir à gérer leurs utilisateurs. Par ailleurs, il permet à l'utilisateur de ne mémoriser qu'un seul couple d'identifiant/mot de passe pour les applications internes et externes à l'entreprise.

### *Le projet Liberty*

Le *projet Liberty* est la première proposition de spécification pour assurer la fédération d'identité. La spécification Liberty est gérée par un consortium spécifique appelé *Liberty Alliance*, regroupant la majorité des acteurs de l'informatique à l'exception notable de Microsoft.

La spécification est stabilisée depuis 2003 et implémentée par de nombreux produits du marché. Par contre, les entreprises utilisant les principes de fédération d'identité sont encore rares, ces concepts étant encore jeunes et plutôt complexes à appréhender.

La spécification Liberty est en réalité un vaste framework composé de nombreuses sous-normes que nous ne décrirons pas en détail dans cet ouvrage. Elle s'appuie sur le framework WS-Security, mais pas sur les spécifications inabouties présentées dans le chapitre précédent (cf. figure 13.5). Par conséquent, elle est stable, mais ne bénéficie pas de la modularité des spécifications précédemment citées, contrairement à WS-Federation présenté dans le paragraphe suivant.

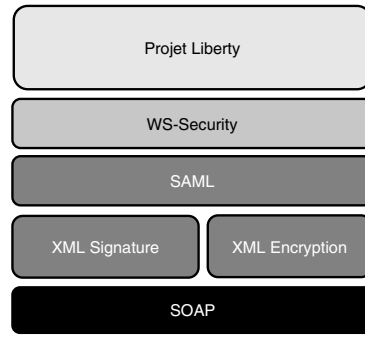



**Figure 13.5** — La pile de fédération d'identité Liberty

### *WS-Federation (GXA)*

WS-Federation est une spécification concurrente à Liberty et beaucoup plus récente proposée par un consortium d'éditeurs intégrant Microsoft.

Elle est encore peu stabilisée et ne dispose que de très peu d'implémentations. Elle s'appuie en revanche sur les spécifications précédentes : WS-Security, WS-Trust, WS-SecurityPolicy. Elle apparaît globalement moins complexe à mettre en œuvre que Liberty.

De plus elle bénéficie du soutien des grands éditeurs, y compris Microsoft, ce qui lui donne une chance de supplanter Liberty.

**Bilan sur la sécurisation des Web services**

Pour conclure ce sous-chapitre sur la sécurité, il faut retenir que le framework WS-Security forme une base stabilisée sur laquelle on peut s'appuyer pour mettre en œuvre une architecture SOA. Il est donc aujourd'hui possible d'adresser les bases fondamentales de la sécurité : confidentialité, authentification, intégrité, non répudiation. Les normes Liberty sont quant à elles stabilisées et implémentées par les grands éditeurs, mais elle risque de souffrir de la concurrence de WS-Federation. Ainsi, pour parer à toutes les éventualités, la société Ping Identity, éditeur majeur de la fédération d'identité, a décidé d'implémenter les deux spécifications.
La pérennité des spécifications GXA peut sembler plus aléatoire, mais elles bénéficient du support d'IBM et Microsoft. Ces derniers devraient travailler dans les prochaines années à sa pérennisation. Il est cependant un peu tôt pour les mettre en œuvre. La figure 13.6 suivante présente le modèle GXA pour la sécurité.

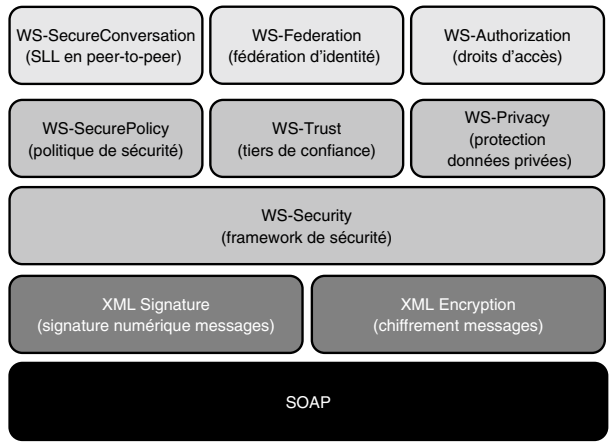



**Figure 13.6** — La pile de sécurité selon GXA

# 13.2 GÉRER LA GARANTIE D'ACHEMINEMENT[1]

## 13.2.1 Les spécifications disponibles

### *WS-Reliability (OASIS)*

WS-Reliability est une grammaire XML normalisée par l'OASIS depuis 2004. Elle offre aux messages SOAP les garanties d'acheminement et de traitement suivantes :

1. Pour un rappel général sur la garantie d'acheminement, voir partie 1.

- **Garantie que le message a été reçu au moins une fois**, et notification de cette réception.
- **Garantie que le message n'a été reçu qu'une seule fois.** Détection et élimination des duplications.
- **Garantie sur l'ordre de réception des messages :** ils seront reçus dans leur ordre d'émission.

WS-Reliability définit pour cela des commandes XML à insérer dans les en-têtes et les corps des messages SOAP. La spécification permet ensuite de notifier les parties en présence des résultats d'acheminement.

### *WS-ReliableMessaging (GXA)*

Cette spécification a été écrite en parallèle à WS-Reliability par un groupe piloté par IBM et Microsoft sans concertation avec l'OASIS. Elle offre le même périmètre que WS-Reliability, mais est incompatible avec cette dernière. Face à cette divergence de normalisation, certains éditeurs comme BEA ont décidé d'implémenter les deux spécifications.

Selon IBM et Microsoft, WS-ReliableMessaging est plus adapté pour fonctionner avec WS-Security, WS-SecureConversation et plus généralement avec GXA.

Depuis mai 2005, des travaux sont en cours pour fusionner les deux spécifications et donner la gestion de la norme résultante à l'OASIS.

### *WS-RM Policy (GXA)*

WS-RM Policy est une déclinaison de WS-Policy au même titre que WS-SecurityPolicy. Cette grammaire XML permet donc de définir une politique sur la garantie d'acheminement. Elle décrit en particulier :

- Le fait qu'un service exige une garantie d'acheminement basée sur WS-ReliableMessaging.
- Les temps d'acheminement maximum attendus par le service.
- Le délai avant réacheminement d'un message.
- Le délai d'attente maximum du message de confirmation de l'acheminement.
- Etc.

La spécification a été proposée à l'OASIS en mai 2005, elle est en cours de ratification et devrait donc passer rapidement dans le giron de l'OASIS.

### *Synthèse sur la garantie d'acheminement*

La figure 13.7 récapitule les grammaires XML de la garantie d'acheminement.

Il semble que l'OASIS devrait à court terme récupérer la gestion des spécifications de garantie d'acheminement (gestion de l'acheminement et politique sur la garantie d'acheminement) et mettre fin au flou qui règne depuis quelques années autour de cette problématique dans les Web Services.

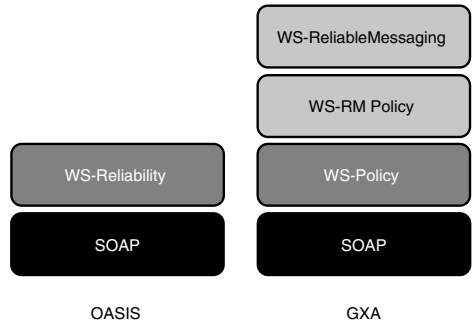



**Figure 13.7** — Les piles de garantie d'acheminement selon OASIS et GXA

# 13.3 GÉRER LES TRANSACTIONS DISTRIBUÉES[1]

Les spécifications présentées dans ce chapitre ont été proposées par un groupe d'acteurs mené par IBM et Microsoft, et qui comprend notamment BEA et IONA. Elles ont ensuite été proposées à l'OASIS, et sont en cours de ratification.

## 13.3.1 Les spécifications disponibles

### *WS-Coordination (GXA)*

Cette spécification propose une grammaire XML pour coordonner des transactions distribuées. Elle fonctionne sur la base d'un service de coordination jouant le rôle de moniteur transactionnel.

Les services désirant participer à la transaction doivent tout d'abord s'inscrire auprès d'un service d'enregistrement en utilisant un formalisme de messages normalisés.

WS-Coordination offre ensuite un système de propagation du contexte transactionnel entre les services[2].

Les messages échangés peuvent être sécurisés via WS-Security, WS-Trust et WS-SecureConversation (cf. chapitre 13.1).

### *WS-AtomicTransaction (GXA)*

WS-AtomicTransaction s'appuie sur WS-Coordination. La spécification précise les mécanismes à utiliser pour gérer les transactions atomiques et permet de traiter les transactions two-phase commit (variantes Durable et Volatile) dans un contexte Web Service.

1. Pour un rappel général sur les transactions distribuées, voir partie 1.
2. Le concept de Contexte est présenté dans la partie 3, chapitre 9 de cet ouvrage. Le contexte offert par WS-Coordination est plus restrictif que ce dernier.

#### *WS-BusinessActivity (GXA)*

WS-BusinessActivity s'appuie sur WS-Coordination. La spécification précise les mécanismes à utiliser pour gérer des transactions longues et les processus de compensation.

### 13.3.2 Synthèse sur les transactions distribuées

La figure 13.8 récapitule les grammaires XML de gestion des transactions.

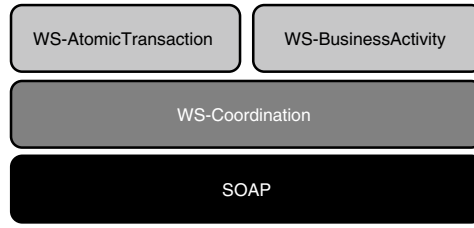



**Figure 13.8** — La pile de gestion transactionnelle selon GXA

## 13.4 SUPERVISER LES SERVICES

### 13.4.1 WS-DM (OASIS)

WS-DM signifie *Web Services Distributed Management*. Cette spécification a été écrite par l'OASIS sur la base d'une norme plus ancienne : *WS-Manageability*.

WS-Manageability était une proposition d'IBM, Computer Associates et HP soumise à l'OASIS en 2003. Cette spécification était conçue pour assurer la supervision de serveurs sur la base d'un langage normalisé et donc indépendant des agents propriétaires (SNMP ou autres) qui rendent la gestion de parcs de serveurs complexe et coûteuse.

L'OASIS a élargi le scope de WS-Manageability pour proposer un framework plus complet : WS-DM s'applique aux appareils électroniques professionnels ou grand public. Elle permet par exemple à une imprimante de signaler que son encre est épuisée, à un projecteur vidéo de signaler une ampoule en fin de vie.

WS-DM décrit deux sous-normes :

- **MUWS** (*Management Using Web Services*), qui permet de superviser tout appareil présentant une interface ad hoc sous la forme de Web Service.
- **MOWS** (*Management of Web Services*), qui permet de superviser les Web Service.

Dès lors qu'une ressource publie son état sous la forme de Web Service, WS-DM permet :

- d'inspecter son état (en attente, occupé, saturé, arrêté, déficient, etc.),
- de suivre ses métriques de fonctionnement,
- de récupérer des alertes.

La grammaire WS-DM décrit en particulier comment :

- Un client accède aux informations de monitoring de la ressource.
- Un client gère les paramètres de la ressource ou s'abonne aux alertes publiées la ressource.
- Un client informe la ressource d'un événement de gestion.

### 13.4.2 WS-Management

WS-Management est une proposition concurrente de Microsoft et des principaux constructeurs de matériel informatique. La spécification est orientée gestion de parc et des ressources réseaux. Elle propose de superviser des PC, serveurs, terminaux mobiles de tous types, et accessoirement des Web Services.

Elle a été soumise à la DMTF[1], un organisme de normalisation qui traite plus particulièrement de la gestion des ressources en réseau d'entreprise.

*WS-Management Catalog* est une sous-spécification de WS-Management. Elle a pour objectif de proposer un référentiel des ressources disponibles (une sorte de registre UDDI orienté réseau). Ce catalogue décrit pour chaque ressource son emplacement, sa version, son fournisseur, ses opérations accessibles, ses propriétés.

La grammaire XML proposée par WS-Management est beaucoup moins complète que WS-DM. De plus, elle semble encore en cours d'écriture. Elle offre cependant l'avantage d'être plus simple à implémenter que WS-DM pour des terminaux simples. Par ailleurs, la force de frappe de Microsoft ainsi que sa maîtrise des systèmes d'exploitation en entreprise pourrait bien l'imposer face à WS-DM.

## En résumé

Pour conclure ce chapitre, il faut retenir que :

- Pour sécuriser une architecture SOA, on peut s'appuyer sur le socle WS-Security les autres spécifications présentées étant sujettes à évoluer. Dans le cadre spécifique de la fédération d'identité, on préférera les spécifications Liberty aujourd'hui opérationnelles et implémentées par les éditeurs.

1. *Distributed Management Task Force*.

- La gestion de la garantie d'acheminement reste encore immature.
- Il en est de même pour la gestion des transactions, d'autant plus que WS-CAF, présenté dans le chapitre suivant, entre en partie en conflit avec les spécifications décrites dans ce chapitre.
- Pour ce qui concerne la supervision, WS-DM est sans aucun doute la norme la plus aboutie aujourd'hui. Mais l'avenir pourrait faire pencher la balance du côté de WS-Management.

# 14

# La composition de services

## Objectif

On décrit ici les spécifications permettant :

- La composition interactive de services au sein d'un portail.
- La composition automatisée des services au sein d'un orchestrateur.
- L'assemblage de services dans l'objectif de créer des applications composites.

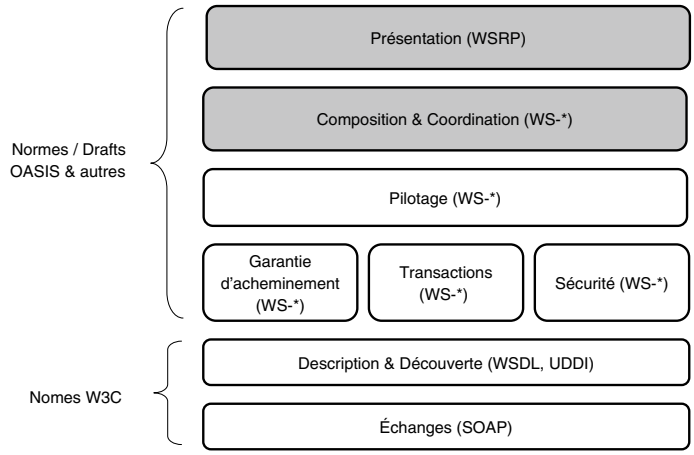



**Figure 14.1** — Les couches supérieures de la pile Web Services

Ce chapitre explicite, dans le cadre des Web Services, les notions présentées dans la partie 2 de cet ouvrage.

Il sort un peu du cadre strict des Web Services car il aborde SCA, une spécification qui s'applique à d'autres technologies comme Java, .NET, etc.

# 14.1 AGRÉGER LES SERVICES AU SEIN D'UNE INTERFACE

## 14.1.1 WSRP (OASIS)

WSRP signifie *Web Services for Remote Portlets*. Cette spécification issue de l'OASIS décrit comment intégrer facilement des services au sein d'une interface de portail.

WSRP permet à un service d'associer aux messages SOAP un modèle de présentation qui pourra être consommé par un portail compatible WSRP. La plupart des portails actuels sont compatibles avec cette spécification : ils peuvent donc intégrer la portlet[1] sans aucun développement.

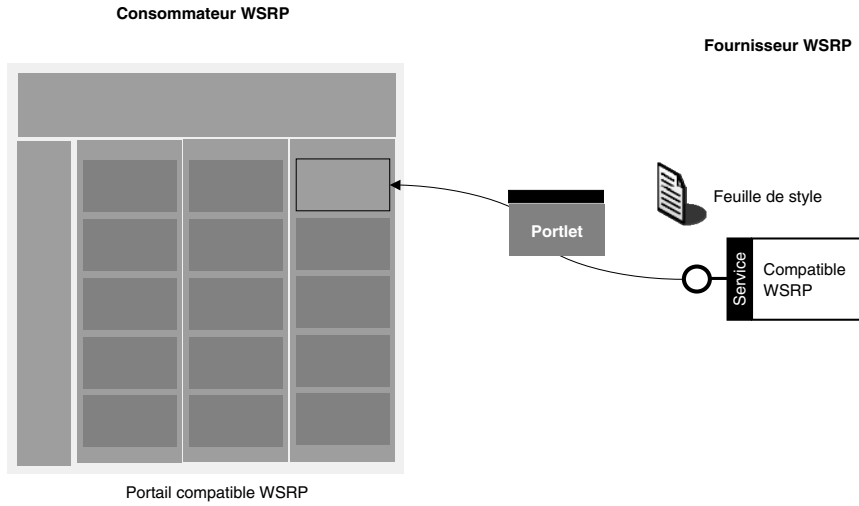



**Figure 14.2** — La consommation de portlet avec WSRP

Un service WSRP peut être consommé par une application interactive qui ne serait pas un portail. Ainsi le développeur d'application composite interactive n'a plus besoin de construire l'interface utilisateur sur la base de l'invocation du service : celle-ci est prédéfinie et auto-générée.

1. Une portlet est un fragment de page dans une interface de portail.

Le protocole permet de sérialiser les données non plus au niveau des messages, mais directement au niveau de la couche de présentation (la session, les URLS ou boutons d'actions, ainsi que le fragment de rendu lui-même).

WSRP permet aussi la publication et la découverte des Portlets au travers d'un registre UDDI.

Enfin, une prochaine évolution du standard devrait intégrer les problématiques de dialogue inter-portlets (cf. normes de gestion de contexte).

# 14.2 COORDONNER LES SERVICES

## 14.2.1 WS-BPEL (OASIS)

BPEL signifie *Business Process Execution Language*. La spécification a été initialement proposée par Microsoft, IBM et BEA en 2002 : elle était intitulée BPEL4WS (*Business Process Execution Language for Web Services*). Sa gestion a ensuite été reprise par l'OASIS et son nom a été mis en conformité avec la notation WS-*. Cependant, on parle le plus souvent de BPEL et non de WS-BPEL.

### *BPEL 1.1 : une première version*

BPEL a pour objectif de décrire un langage de programmation de « haut niveau », par opposition à des langages considérés comme de « bas niveau » : Java, .NET, COBOL, etc. Tandis que les langages de « bas niveaux » font appel à des composants et des API, BPEL invoque des services.

BPEL dispose d'une algorithmique simplifiée qui rappelle celle des langages de « bas niveau » :

- Boucles *while*, *switch* (disponible aujourd'hui avec la version 1.1).
- Boucles *if-then-else*, *repeatUntil*, *forEach* (disponibles dans la future version 2.0).

La composition des services s'appuie sur les contrats WSDL : ils sont parcourus par l'orchestrateur de services et indispensables pour leur assemblage, puis leur invocation.

La spécification offre plus largement les fonctionnalités suivantes pour gérer l'orchestration du processus :

- Type d'appels et d'attente de réponse de la part de services (commandes : *invoke*, *receive*, *reply*, *wait*).
- Invocation de services en série (commande : sequence).
- Invocation des services en parallèle (commande : flow).

- Manipulation[1] des données d'échange entre les services (commandes : *assign*, *copy*).
- Regroupement de séquence d'invocations de services (commande : *scope*).
- Gestion des erreurs (commandes : *throw*, *rethrow*).
- Gestion de procédures de compensation (commande : *compensate*).

BPEL ne fournit pas de représentation graphique normalisée des processus. BPMN (*Business Process Modeling Notation*) a été conçu dans cet esprit, mais la normalisation reste pauvre en regard des possibilités de la grammaire d'exécution. En attendant, certains environnements du marché proposent donc la leur. L'exemple de la figure 14.3 s'inspire de l'un d'entre eux.

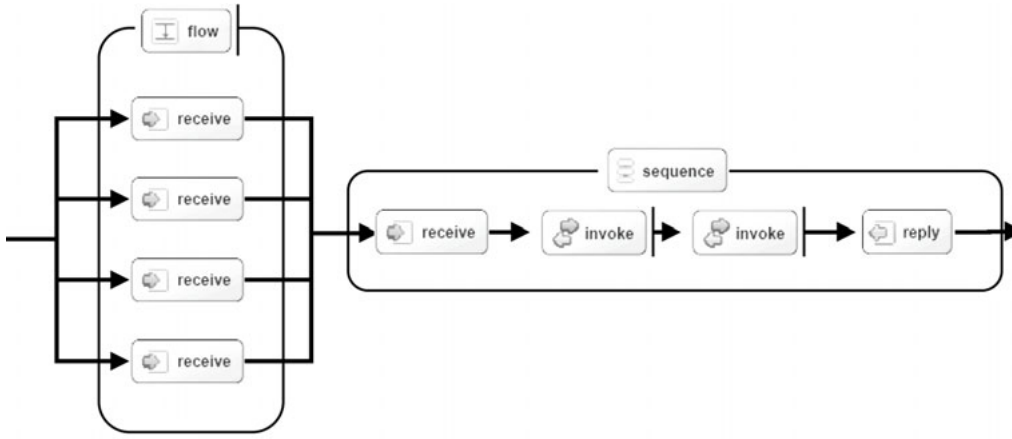



**Figure 14.3** — Exemple d'invocation de services en parallèle et en série

Enfin, BPEL fournit des mécanismes d'extension permettant de déléguer à des langages de « bas niveau » des parties de processus trop complexes pour être prise en charge directement par la grammaire XML elle-même.

### *Limites de BPEL 1.1*

#### *Gestion de sous-processus*

Si BPEL 1.1 propose de regrouper des séquences d'invocation de services (les scopes), cette fonctionnalité se limite à offrir une facilité d'écriture. La grammaire est en effet incapable de gérer des processus et sous-processus : BPEL travaille forcément à très faible granularité, ce qui n'est pas satisfaisant pour décrire des processus très complexes ou pour permettre de capitaliser sur des processus.

On peut gérer des sous-processus comme des processus parallèles, mais on perd dans ce cas le contexte du processus global.

1. Il s'agit de manipulation simple : BPEL 1.1 n'a pas vocation à faire des transformations de grammaire XML.

#### *Gestion de contexte*

La gestion de contexte proposée par BPEL se résume à du couper/coller de données entre résultat d'un service et paramètres d'invocation du service suivant : elle est donc très rudimentaire, et cela complique considérablement l'utilisation de BPEL.

De fait, il est nécessaire de prévoir une gestion de Contexte, via un service de Gestion de Contexte tel que décrit dans la partie 2, ou, au minimum, d'introduire dans BPEL des appels à des « morceaux de code » Java ou autre pour effectuer facilement ce couper/coller.

#### *Gestion des interventions humaines*

De plus, le reproche que l'on fait le plus souvent à BPEL 1.1 est son incapacité à gérer des tâches humaines (cf. partie 3).

### *Apports de BPEL 2.0*

BPEL 2.0 est actuellement à l'état de brouillon à l'OASIS : la spécification définitive devrait sortir courant 2006.

Parmi ses nouveautés, on trouve :

- La gestion de l'algorithmie *if-then-else*, *repeatUntil*, *forEach* (cf. paragraphe précédent).
- La gestion des activités de compensation à l'échelle d'un scope.
- La prise en charge de transformation XSLT dans le cadre de la gestion des variables.

Par ailleurs, un certain nombre d'extensions doivent sortir avec BPEL 2.0. Ces extensions seront gérées de manière indépendante car les différents acteurs de la normalisation ne sont pas d'accord sur leur pertinence.

L'extension la plus intéressante est BPEL4People : elle permettra de gérer les fameuses « *people activity* », c'est-à-dire de mélanger des Workflows humains aux activités d'orchestrations de processus. BPEL4People permettra en particulier de gérer des corbeilles de tâches et des chaînes de validations entre acteurs au sein d'une hiérarchie.

BPEL4People semble donc offrir la convergence tant attendue entre les processus automatiques (orchestration) et les processus humains (Workflow). Il semble que certains puristes aient considéré au sein de l'OASIS que cette extension sortait du scope de BPEL et aient donc refusé de l'inclure dans BPEL 2.0. De plus, BPEL4People conduit à des modifications assez profondes de l'existant BPEL, et il n'est pas évident que beaucoup d'éditeurs suivent cette extension. D'autant qu'il existe des solutions moins coûteuses et plus flexibles, comme celle proposée dans la partie 3.

Parmi les autres extensions à venir, on attend SPE (*Sub Processes Extension*) une norme qui permettra la gestion des sous processus avec imbrication et gestion partagée de contexte.

Enfin l'extension BPELJ permettra l'invocation directe de composants écrits java, l'objectif étant ici de sacrifier l'interopérabilité offerte par SOAP au profit de la performance d'exécution.

### 14.2.2 WS-CAF (OASIS)

WS-CAF signifie *Composite Application Framework*. Cette spécification issue de l'OASIS décrit comment construire une application composite sur la base de Web Services (cf. partie 2 pour la définition des applications composites).

De même que WS-Security, WS-CAF est un framework qui repose sur des sous-spécifications : WS-CTX, WS-CF et WS-TXM.

WS-CAF est à la frontière entre les spécifications de gestion de la composition de service et celles de gestion des transactions. Par conséquent, certaines sous-spécifications de WS-CAF peuvent entrer en concurrence avec celle décrites dans le paragraphe 13.3, tandis que d'autres entrent en concurrence avec BPEL décrit dans le paragraphe précédent.

WS-CAF est la somme des trois spécifications suivantes :

- **WS-CTX** (*Context Management*) permet d'échanger un contexte entre l'application composite créatrice du contexte et les services invoqués. WS-CTX est donc concurrent de WS-Coordination.
- **WS-CF** (*Coordination Framework*) permet de partager un contexte et de gérer un enchaînement d'activités transactionnelles ou non : synchroniser les participants, diffuser des informations, envoyer des alertes, etc. WS-CF a donc un recouvrement fonctionnel partiel avec BPEL.
- **WS-TXM** (*Transaction Management*) permet de gérer les aspects transactionnels si cela est nécessaire. La spécification traite des transactions 2 phases commit et des transactions longues. Elle entre donc en concurrence avec WS-AtomicTransaction et WS-BusinessActivity.


**Figure 14.4** — La composition selon l'OASIS

WS-CAF est un exemple très représentatif des problématiques de concurrence entre les spécifications Web Services. Il est d'autant plus complexe à appréhender que WS-CAF et BPEL sont deux normes gérées par l'OASIS. La vision de l'OASIS semble être la suivante : utiliser WS-CAF uniquement pour la gestion des transactions, utiliser BPEL pour la partie orchestration de haut niveau.

### 14.2.3 SCA

SCA signifie *Service Component architecture*. Cette spécification récemment émise dans sa version 1.0 par un groupement de plusieurs éditeurs vient d'être proposée à l'OASIS pour normalisation. La présence de SCA dans cette partie peut paraître discutable : en effet, il s'agit d'une spécification pour la composition de service indépendante de la technologie Web Services.

SCA peut être utilisée avec des Web Services, mais aussi avec des services écrits et Java, .NET, COBOL ou C++. On a choisi de la présenter dans ce chapitre car elle complète la vision sur les spécifications de composition de services.

Les problématiques adressées par SCA sont les suivantes :

- Redéploiement d'un service sans impacter les clients consommateurs de ce service.
- Packaging d'un service constitué de composants locaux ou distribués.
- Paramétrage technique du service au travers de fichiers XML.

SCA propose une grammaire XML pour expliciter comment un service se construit « statiquement » sur la base d'autres services. La spécification ne repose pas, contrairement à BPEL, sur un moteur d'orchestration. Il s'agit en revanche d'invocation point à point entre un service composite et le ou les services sous-jacents.

SCA spécifie un modèle d'assemblage des services, en précisant la dépendance du service appelant vis-à-vis des autres. La spécification décrit si les appels de service sont :

- Synchrones ou asynchrones.
- Sécurisés ou non.

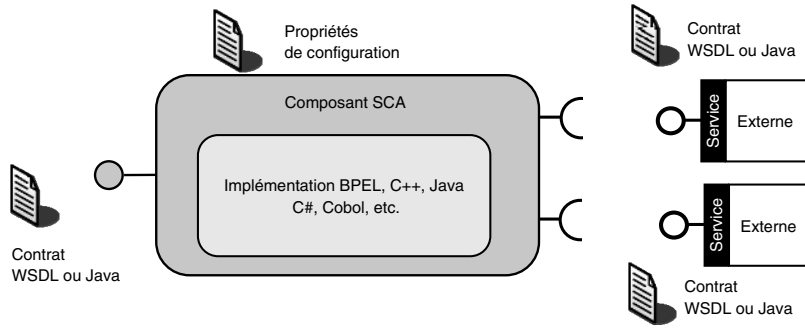



**Figure 14.5** — Un container SCA

- Transactionnels ou non.
- S'ils offrent une garantie d'acheminement ou non.

SCA définit donc la méthode d'invocation et d'encapsulation des services dans un langage XML, indépendant de la technologie utilisée pour appeler ces services. Cette technologie peut être SOAP, RMI, .Net Remoting, etc. La spécification est donc parfaitement adaptée à une architecture SOA hétérogène où l'on utiliserait de l'invocation Java au sein du réseau local, et de l'invocation SOAP lors d'échange avec des partenaires (ces aspects seront explicités dans la partie 5 de cet ouvrage).

SCA permet d'offrir des « containers de service SCA », qui se chargent d'automatiser l'assemblage et le déploiement des services en interprétant les fichiers XML.

C'est la généralisation au contexte SOA des containers EJB, et surtout du container de service SPRING. Beaucoup plus simple à mettre en œuvre que les containers EJB 2.x, cet outil popularise l'injection de dépendance, et de ce fait il facilite notablement l'assemblage de solutions. Mais SPRING est limité au monde Java : SCA est en quelque sorte la généralisation aux web service de SPRING.

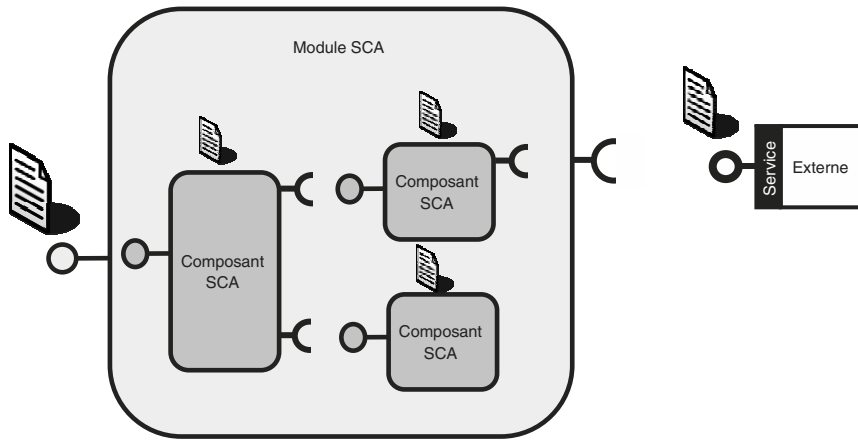



**Figure 14.6** — Encapsulation de containers SCA

## En résumé

Pour conclure ce chapitre, il faut retenir que les spécifications de composition de services sont plurielles, mais qu'elles n'adressent pas exactement les mêmes besoins. BPEL est cependant la grammaire la plus utilisée à ce jour.

Pour ce qui concerne l'assemblage de services au sein d'applications composites, SCA offre une norme très intéressante qui généralise la notion de container bien connue des architectes Java ou .NET.

# CINQUIÈME PARTIE

# SOA : Une mise en œuvre concrète

Le but de cette partie est de donner accès à des retours d'expériences concrets, sur la mise en œuvre des Services au sein de projets SOA, locaux au SI ou destinés à ouvrir le SI sur le monde extérieur (projet inter SI)…

Les chapitres précédents ont initié les réflexions avec la définition préalable du périmètre d'une solution métier et la modélisation des services et applications de cette solution. Il s'agit maintenant de confier aux équipes concernées la réalisation des services ou d'une infrastructure de services telle que définie précédemment. Des questions/réponses permettront d'élaguer l'univers des possibles afin de permettre un démarrage rapide de ce chantier de mise en œuvre de services au sein d'une architecture d'accueil.

Parmi les principales questions pour le maître d'œuvre des services, on retrouve :

- Tous les services ont-ils les mêmes contraintes ?
- Quelles sont les conventions et normes à utiliser ?

- Quels sont les choix techniques à faire au lancement de l'implémentation d'une solution métier ?
- Quels outils ou techniques vont faciliter les développements et l'exploitation ?

Cette partie veut donc démystifier le démarrage de la mise en place d'une solution qui respecte les principes SOA et souhaite en tirer des bénéfices. Il ne s'agit pas d'un référentiel systématique qui répond à toutes les questions mais d'un ensemble de points clefs pour le bon démarrage des projets.

Le chapitre 15 présente les différences techniques dans le contexte du SI étendu et du SI local.

Le chapitre 16 montre les atouts de WSDL comme un formalisme pivot capable de dépasser le cadre des WebServices. On y aborde aussi quelques aspects techniques clefs pour montrer quelques bonnes pratiques de conception.

Le chapitre 17 conclut sur les différents cas de figures techniques selon les typologies de service et présente une démarche pour s'adapter aux évolutions des contraintes techniques.

# 15

# SI étendu ou SI local ?

**Objectif**

Ce chapitre se donne pour objectif d'identifier les contraintes métiers et techniques pour préparer une solution conforme à SOA : savoir où utiliser ou non des Web Services. Comment faire ces choix sans être un devin ou avoir un recul technique de 10 ans ?

Cette problématique est notamment exprimée dans le cas de services à offrir à des partenaires, et dans le cas de services internes composant le SI local. Cette division SI étendu et SI Local est analysée et relativisée.

## 15.1 LA PORTÉE DES SERVICES

Après la reconnaissance des enjeux métiers et de la conservation de l'existant, les DSI doivent déployer des solutions au sein du SI. La problématique de services et d'architecture d'accueil des services est souvent séparée en deux espaces distincts : SI local et SI distant. Les caractéristiques de ces deux espaces de services sont différentes, ce qui peut faciliter certains choix.

En revanche, la frontière n'est pas toujours durable ou parfaitement identifiée. Tel service peut par exemple être à destination des partenaires et du SI local en même temps.

Le service Intervention illustré dans les parties précédentes reste un cas intéressant à décrire pour la mise en œuvre. La déréglementation demande d'offrir ce service à des partenaires externes mais il faut continuer à y accéder par le SI interne. Le même service aura, alors, des contraintes techniques d'exploitation différentes.

D'un côté il doit être accessible aux SI extérieurs sans présupposer de leur plate-forme technique et de l'autre permettre une intégration au SI interne dont la plate-forme est connue. Il se peut que les conditions d'accès au service diffèrent pour les partenaires et pour le SI interne. Ne serait-ce que pour éviter de sortir du SI pour y retourner, mais aussi pour des raisons de sécurité, par exemple.

Certaines caractéristiques de solutions métiers permettent d'identifier clairement les contraintes minimums à prendre en compte.

## 15.2 LES CONTRAINTES DE L'ENTREPRISE ÉTENDUE

### 15.2.1 Un terrain favorable aux Web Services

Dans le cas d'un service de haut niveau à partager avec des SI distant, on ne peut pas présupposer des technologies de plate-forme. De plus, les protocoles doivent rester les plus simples possible afin de maîtriser les outils de sécurité, monitoring, et fiabilité. Le WebService (au sens sérialisation XML) règne sur ce terrain et la stabilité du protocole porteur http est très adaptée.

Ces technologies sont encore jeunes et ne traitent pas aujourd'hui de tous les aspects techniques des échanges. Certains aspects restent fragiles voire peu adaptés, comme la gestion de transactions, l'utilisation de contextes distribués, la mesure de la qualité de service et la garantie de performance. Ces aspects sont tous en cours de stabilisation par les technologies WebServices comme le montre la partie 4.

L'approche SOA passe aussi par une typologie des services adaptée. Des Services Applicatifs à gros grain qui sont les seuls à être exposés aux SI distants sont les premiers à bénéficier de l'interopérabilité et l'indépendance technique des WebServices. Les services internes seront plus proches des contraintes techniques liées à la distribution synchrone, les transactions, etc.

### 15.2.2 Peut-on opposer l'EDI aux WebServices ?

Les technologies EDI dominent les échanges inter-entreprises existants. Les technologies XML et WebService en particulier sont souvent perçues comme un concurrent des solutions EDI. Le débat n'est pas terminé. Les sociétés et les éditeurs qui ont beaucoup investi dans des solutions EDI d'envergure sont sceptiques et affichent la suprématie de l'EDI en termes de performance, de sécurité et de qualité de service. Les pionniers XML et les éditeurs ayant opté pour des solutions XML/Web Services défendent pour leur part la rapidité de mise en place, la flexibilité et l'évolutivité offertes par ces solutions.

L'EDI est aujourd'hui mature et permet de maîtriser parfaitement les échanges massifs de données. Ces échanges représentent souvent des événements de gestion, ce qui les rend comparables à des demandes d'appels de services. En revanche, les technologies WebServices proposent un système d'invocation à l'unité.

Cette différence notoire ouvre la voie vertueuse : la complémentarité de l'association de l'EDI existante avec les technologies XML.

La mise en place de services d'échange via des WebServices est en effet très rapide et flexible. Elle permet d'intervenir vite là où une solution EDI demanderait un travail plus important. Des flux d'échanges XML sur ces technologies sont très adaptés à des échanges ponctuels de petite taille. Les solutions EDI offrent, quant à elles, plus de garanties sur des échanges massifs, mais aussi bien plus de contraintes.

## 15.3 L'ENJEU TECHNIQUE DU SI LOCAL

Au sein du SI local, selon les caractéristiques requises par les services (son mode de transaction, le support de la charge, la sécurité, etc.), il n'y a pas nécessairement de contrainte d'interopérabilité forte. La ou les plates-formes du SI local sont connues et globalement homogènes. Sur une telle plate-forme, les services distribués utiliseront les techniques de Remoting classiques, comme .NET TCP Remoting, ou CORBA, RMI/IIOP et l'ensemble des protocoles synchrones basés sur une distribution réseau des services. Les services de grain le plus fin (CRUD par exemple) seront implémentés sous la forme technique la plus simple. Dans le monde Java, il s'agit des POJO, « *PlainOld Java Object* », qui sont de simples classes Java, respectant la norme JavaBean. Leur équivalent .NET est le PONO, « *Plain Old.NET Object* », une simple classe indépendante des conteneurs offerts par .NET. On peut généraliser ces POJO/PONO à toutes les plates-formes avec cet acronyme : POxO qui désigne des composants simples.

> Le principe technique majeur est d'éviter une communication WebService entre deux composants de la même application, au sein de la même plate-forme technique, et situés sur un même serveur ou deux serveurs reliés sans intermédiaire par réseau très haut débit.

Pourquoi un tel principe ? D'abord pour des contraintes de performance. Cette contrainte de performance porte sur le temps d'exécution ET le temps d'invocation. Le temps d'invocation d'un Web Service inclut des traitements potentiellement coûteux (transformation XML, vérification du typage des données…). Si un service est critique en temps d'exécution ET qu'il participe à un enchaînement de services lui-même critique, il est donc opportun d'envisager des services les plus basiques possibles, donc ni WebServices, ni Remoting ou messaging, mais POxO.

Que faire si les services pré-existants ont des contraintes de robustesse (?) lors de leur réutilisation ? Les technologies Remoting apportent des réponses en terme de transaction et sécurité, les Web Services permettent la validation avancée des paramètres et la sécurité.

Tout dialogue à prévoir avec des plates-formes non définies implique en revanche de prévoir un accès WebService.

Les contraintes de déploiement non connues ou changeantes peuvent être résolues plus tard par la multiplicité des formes de déploiement d'un service. Le chapitre 16 présentera les choix à faire lors de la mise en œuvre des services au sein d'un SI.

## 15.4 UNE APPROCHE TECHNIQUE DIFFÉRENCIÉE

Lorsque l'on considère globalement les cas d'usage des services dans le SI étendu ou local, on peut proposer des techniques adaptées. En partant de l'idée maîtresse que SOA ne se limite pas aux Web Services, une première approche des cas d'usage est synthétisée dans le tableau suivant :

**Tableau 15.1** — Les cas d'usage des protocoles

| Cas d'usage | Contrat de service | encodage | Protocole de transport |
|---|---|---|---|
| Services fortement standardisés, ouvert sur Internet. | WSDL | SOAP | Web (HTTP) |
| Services fortement standardisés, ouvert sur Internet. | WSDL | SOAP | Autre protocole : FTP, SMTP, etc. |
| Services fortement standardisés, internes au SI. | WSDL | Encodage binaire | Protocole généraliste : JMS, … |
| Services faiblement standardisés, ouvert sur Internet (WEB 2.0). | XML « propriétaire » non WSDL | XML | REST |
| Services fortement standardisés, liés à un existant, et internes au SI. | Autre norme de type Remoting : IDL CORBA, EJB, .NET TCP… | Encodage binaire | Protocole spécialisé : IIOP, RMI |
| Services localisés au sein d'une application ou d'un groupe d'application. | Interface Java, C# | *Sans objet* | Sans objet (appel local à une JVM ou une CLR) |

L'emploi de l'infrastructure SOA se justifie à plusieurs niveaux dans l'urbanisation du SI, qu'ils s'agissent de simples applicatifs, de silos, ou de processus inter silos.

Au sein d'une application, SOA permet, via composition, la création de nouveaux services ou la réutilisation et l'orchestration de services existants, en général

non distribués (POxO). Au niveau d'un silo, SOA permet la communication entre applications, en général via un bus synchrone ou asynchrone. Au niveau inter silos, SOA s'envisage en général via un bus asynchrone et éventuellement via des Web Services lorsque les silos sont physiquement et géographiquement distribués. Enfin, au niveau du SI Étendu, SOA utilise des Web Services synchrones et sans doute asynchrones lorsque les normes le permettront.

## En résumé

Les services étendus à l'extérieur de l'entreprise sont la cible première des technologies WebService.

A contrario, les services de grain fin restent des composants non distribués, réutilisés par des services de plus gros grain.

Entre ces deux extrêmes, les services de grain plus ou moins moyen (Services Applicatifs, Services Fonctionnels) seront distribués ou non distribués, et en cas de distribution, seront accédés via Web Service, ou via une technologie classique.

La frontière est donc floue entre SI distant et SI local vu de la mise en œuvre SOA. Le tout WS est en tout cas une erreur.

# 16

# Les atouts de WSDL

## Objectif

Ce chapitre démontre dans un premier temps un intérêt réel à utiliser WSDL comme formalisme « universel » de définition des services. Il présente aussi des bonnes pratiques de la mise en place de ces définitions.

Dans un second temps, il aborde deux points pratiques de l'utilisation des spécifications des WebServices : les normes d'interopérabilité et l'utilisation des styles de messages SOAP.

Il s'agit de points techniques de bas niveau, mais leur utilisation a souvent tendance à être empirique ou historique parce que pas toujours maîtrisée. Il s'agit donc de donner les bases suffisantes à leur utilisation appropriée. Démystifier le WSDL, c'est faciliter son utilisation et sa généralisation.

## 16.1 UTILISER WSDL COMME FORMALISME PIVOT

### 16.1.1 Un socle formel qui dépasse les WebServices

Les WebServices ne sont pas la seule forme technique de réponse au sein d'une architecture SOA (cf. tableau 15.1, les cas d'usage des protocoles). Faut-il pour autant utiliser un formalisme de description des contrats de service différent pour chacun des cas ? On a vu dans la partie 4 que le langage de description des WebService définit l'interface d'utilisation d'un service dans sa partie abstraite. Cette définition peut aussi servir à la mise en œuvre de forme POxO ou Remoting des services par simple génération.

Cette vision permet de rentabiliser les compétences WSDL d'une équipe qui pourra les exercer avec ou sans la présence de WebServices proprement dit, en utilisant la partie abstraite du WSDL comme un formalise de définition de service.

WSDL est un socle intéressant pour bénéficier des apports du méta-langage XML : Ce langage est notamment auto-descriptif, il autorise l'extensibilité de syntaxe, il permet une validation automatique des types de données, etc. Bâtir sur ce socle permettra de bénéficier d'avancées technologiques, ne serait-ce que les améliorations des technologies de transport en terme de performance ou de gestion de nouveau protocoles ou middlewares.

### 16.1.2 Faut-il devenir un gourou du WSDL ?

La réponse est non. De nombreux outils dédiés au monde XML permettent déjà une représentation graphique d'un WSDL, comme, par exemple, XMLSPy d'Altova Software, ou d'autres éditeurs dédiées comme Stylus Studio, ou OxygenXML.

De plus, les approches MDA ont favorisé la projection de modèles UML vers une expression WSDL. Il est maintenant possible de modéliser un WSDL en UML. Le PIM[1] contiendra la partie abstraite du WSDL : Définitions, Types, PortTypes (interfaces), messages, et Parts. Le reste sera porté par le PSM : binding, services et ports.

Les outils WSAD/RAD, Sun One Studio, Visual Studio, etc., permettent déjà une modélisation UML des WebServices. Des modèles et profils UML existent et pourront être utilisés à terme dans tous les outils du marché, qu'ils soient Open Source ou propriétaires.

La connaissance de quelques notions importantes mais peu nombreuses sera utile afin de piloter ces outils en bénéficiant au mieux des technologies SOA et WebServices. Le chapitre 16.3 « Démystifier l'utilisation de WSDL : le Style à utiliser » présente quelques points sensibles pour guider les concepteurs et développeurs des services.

### 16.1.3 Quelle est la démarche de production des WSDL ?

#### *Approche par les modèles ou approche par le code ?*

Tous les outils permettent deux approches :

- Partir des modèles de services représentés en UML puis en dernière instance par le WSDL (approche par les modèles) et formalismes associés.
- A contrario, générer ces modèles WSDL à partir du code des services existant (approche par le code).

Microsoft publie beaucoup sur ses outils et leur capacité à gérer l'approche par le code. Il s'agit d'une stratégie orientée vers les communautés de développeurs. Ces

1. Cf. chapitre 11 pour un bref rappel sur MDA et les modèles PIM et PSM.

outils fonctionnent pour des projets de petite taille et indépendant d'un SI métier complet. Cette approche amène souvent **une utilisation technologique nivelée par le bas** du service Web. Elle ne permet pas vraiment l'intégration du point de vue métier dans le processus, alors que c'est un des bénéfices majeurs des SOA. En revanche, cette approche est un atout majeur s'il faut prototyper d'après un existant, ou adapter temporairement une solution ou si l'on a peu ou pas de problématique métier étendue qui répond à des enjeux sur la durée et la capitalisation.

L'approche par le code n'est donc pas foncièrement conforme à la démarche SOA. Il s'agit avant tout de raccourcis technologiques pour obtenir une base rapide à une description WSDL qu'il faudra adapter aux contraintes fonctionnelles dans une approche par les modèles si on veut bénéficier des apports de SOA.

Repartir des modèles permet de faire cohabiter l'expertise métier avec la maîtrise d'œuvre. Les modèles obtenus permettent de simuler au plus tôt les scénarios fonctionnels validant le découpage et le potentiel de réutilisation. Les outils sont aujourd'hui de plus en plus à même de supporter cette démarche.

### *WSDL et système existant*

Dans le cas de réutilisation complète de l'existant, par exemple des services Mainframe d'un SI, il est toujours payant de repartir d'une modélisation WSDL (ou UML) de ces services afin d'y intégrer toutes les contraintes métier d'utilisation qui seront à prendre en compte par les nouveaux composants applicatifs client.

Les équipes de développement utilisant les nouvelles technologies ont souvent été désorientées par les spécifications des services du Mainframe. Ce manque d'expérience et le fait que leur formation ne comprend pas toujours une présentation concrète des grands systèmes pèsent sur leur compréhension et leur efficacité à travailler avec cet existant. Une formalisation WSDL de ces services sera un pont immédiat entre ces technologies.

### *Séparation et capitalisation*

La facilité de maintenance est toujours au cœur des préoccupations de la maîtrise d'œuvre. Il se révèle de ce point de vue très bénéfique de distinguer dans un contrat WSDL l'aspect Abstrait (ce qui est indépendant de la technologie et des protocoles) et l'aspect Concret (ce qui est dépendant). Cette distinction classique permet de joindre ou réutiliser des sous-parties de modèles WSDL et de les redéployer à loisir : le WSDL est classiquement séparable en 3 parties :

- Les schémas d'une part, qui définissent les informations contenues dans les messages reçus/émis par le service : ces schémas peuvent préexister avant la notion de service car ils représentent les entités métier.
- Une définition abstraite du service, contenant sa description sans les modalités d'implémentation (protocole et déploiement).
- Une partie concrète qui propose les liaisons d'une interface selon un certain format avec un protocole donné à une adresse donnée.

WSDL permet d'importer des WSDL et des schémas. La partie concrète d'un contrat WSDL peut donc importer les définitions abstraites des services qui eux même peuvent importer des schémas. Cette démarche apporte une plus grande facilité de maintenance et de lisibilité.

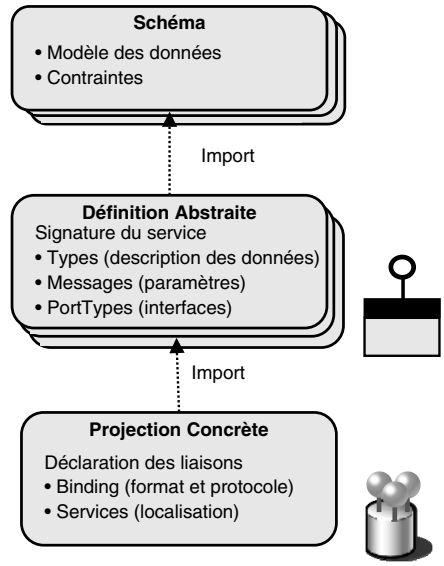



**Figure 16.1** — Trois aspects distincts d'un WSDL

Lors de la modification d'un service, on peut encore composer des WSDL en procédant par extension. L'importation offre un mécanisme proche de l'héritage qui permet de ne pas modifier des WSDL déjà en production.

**Principe :** utiliser les mécanismes de schéma et d'importation pour construire de façon modulaire les contrats WSDL.

### 16.1.4 WSDL résout-il tous les problèmes ?

La réponse est évidemment non. WSDL se cantonne à la description d'un service en terme statique. Dans ce rôle, il est nécessaire et suffisant. La norme WSDL 2 étendra son périmètre en intégrant des éléments de qualité de service.

En revanche, rien ne permet actuellement d'étendre le contrat aux aspects qualité de service, la composition, le monitoring, etc., en utilisant WSDL 1.1. Des extensions spécifiques non normées sont disponibles en dehors de tout standard. Ces aspects sont abordés par d'autres normes et ont vocation à être portées par les infrastructures accueillant les services : serveurs d'application et ESB. La partie 7 aborde les Bus de services ESB. On cantonnera donc le WSDL à ce qu'il fait bien.

# 16.2 WSDL ET INTEROPÉRABILITÉ

L'avènement des services Web a donné lieu à une promesse d'interopérabilité facilitée. Hélas, ce vœu pieux a été rapidement contrarié. Entre divers mondes technologiques, force était de constater que sans garde fou, les WebServices s'avéraient une technologie de plus avec son lot de nouveaux problèmes.

## 16.1.1 Une réaction rapide au démarrage raté des WebServices

Les organismes de standardisation et les grands éditeurs ont collaboré à mettre en place ces gardes fous au travers de l'organisation WS-I pour « WebServices-Interoperabilité ». Il s'agit de restreindre une partie des libertés offertes par les spécifications à la base des WebServices, soit pour simplifier les choix des implémentations, soit pour masquer des portions des spécifications qui étaient trop floues pour une implémentation canonique et stable. Chaque sous-ensemble traité par WS-I est appelé un Profile. La première de ces normes est appelée le Basic-Profile.

Les Profiles suivants portent sur la sécurité des WebServices. WS-I fournit aussi des applications exemples et des outils de tests afin d'intégrer au mieux les recommandations.

| On trouvera les travaux WS-I à cette adresse : http://www.ws-i.org/

## 16.1.2 Une norme d'interopérabilité de premier niveau : WS-I Basic Profile

Basic-Profile 1.0 est le premier résultat des travaux sur le renforcement de l'interopérabilité des WebServices. Ce Profile porte sur les aspects statiques des WebServices.

Basic-Profile contraint principalement la définition des messages SOAP et la définition des services. Il s'agit en soit d'une centaine de recommandations/restrictions élémentaires sur les messages SOAP (uniquement à l'attention des moteurs) et sur les descriptions WSDL. Ici, il s'agit de simplifier le WSDL pour simplifier les choix des développeurs et des outils d'édition et de modélisation des services. Les outils de génération intègrent la vérification et la conformité à WS-I Basic Profile. On peut par exemple citer AXIS du groupe Apache, dont les générateurs sont conformes Basic Profile 1.0.

WS-I Basic Profile 1.0 porte sur la spécification WSDL 1.1, ce qui explique le choix de cet ouvrage de montrer des exemples de WSDL correspondant à cette version. Cette norme continuera d'évoluer pour prendre en compte tous les éléments qui apportent une garantie d'interopérabilité.

Enfin, WS-I Basic Profile 1.0 a pour objectif de faciliter la génération de code à partir de spécifications WSDL. Initialement, ce travail vise à simplifier le traitement du WSDL par les moteurs SOAP. Mais cet avantage est disponible pour tous les autres moteurs ou générateurs exploitant WSDL. WS-I renforce donc la stabilité des

formalisations WSDL et des outils WSDL-Centriques. Cela donne encore du poids à l'utilisation du WSDL comme un formalisme pivot quelle que soit la technologie d'implémentation des services.

## 16.3 DÉMYSTIFIER L'UTILISATION DE WSDL : LE STYLE À UTILISER

Le syndrome de la page blanche revient lorsqu'il faut décider du style d'échange à utiliser pour tel WebService. Il existe 4 styles : RPC/encoded, RPC/Literal, Document/encoded et Document/Literal. Souvent le choix entre ces styles a été guidé par le premier exemple trouvé sur un site Internet. La majeure partie des développeurs ne connaît pas les différences profondes induites par les styles.

Or le choix du style peut influer notablement sur la pérennité et la performance d'une solution WebServices. La production de WSDL étant souvent confiée à des outils, paramétrer efficacement ces outils en terme de style SOAP requiert une connaissance minimum pour éviter de propager un mauvais choix sur tous les WSDL générés.

### 16.3.1 RPC ou Document

Les styles principaux de communication d'un WebService doivent être Document ou RPC[1]. Les noms choisis par la spécification WSDL sont trompeurs car ces deux styles n'ont rien à voir avec un modèle de programmation par message ou par appel de fonction.

> Le style s'applique uniquement au mécanisme de traduction d'un binding WSDL lors de la construction d'un message SOAP.

Les services suivent leur modèle de programmation indépendamment de ces choix. Ainsi, on peut utiliser le style RPC dans un WebService pour envoyer un message à un consommateur asynchrone de type JMS/Websphere MQ/MS MQ, etc.

### 16.3.2 Litteral ou Encoded

On retrouve la même précision pour Litteral ou Encoded qui sont des notions utilisées lors du mapping du WSDL vers SOAP à l'envoi ou à la réception des messages.

Litteral est explicite : le corps du message SOAP contient des fragments littéraux XML simples. Le seul contenu du WSDL permet leur identification et leur utilisation.

Encoded ajoute des informations de sérialisation/dé-sérialisation entre XML et « une plate-forme » donnée. Ce qui rend cette technique aléatoire en terme d'inter-

1. RPC : *Remote Procedure Call*, appel d'une procédure à distance.

opérabilité. Un exemple de message SOAP simple pour montrer la différence entre Literal et Encoded est disponible plus loin.

Pourquoi cette norme ? Encoded ajoute aussi la normalisation d'un attribut Href qui permet de référencer dans ce dialecte un autre nœud d'un arbre XML, offrant ainsi la capacité innée de traiter des graphes d'éléments, graphes qui ne sont pas exprimables en XML Schéma.

### 16.3.3 Conformité avec WS-I

Les styles XXX-Encoded ne sont pas conformes à WS-I Basic-Profile comme on peut s'y attendre car ils imposent des méta-informations d'encodage/décodage qui peuvent être dépendantes d'une plate-forme. Ces pratiques sont un risque en terme d'interopérabilité. On trouvera énormément de services RPC-Encoded dans les existants avec lesquels il faut compter. Si l'interopérabilité doit coexister avec l'utilisation de cet existant, on peut devoir encapsuler ces services dans un WSDL conforme WS-I.

### 16.3.4 Comparaison des styles

La partie concrète du WSDL comprend des informations sur le protocole de communication avec le service. Le binding est l'occasion de choisir les facettes de SOAP utilisées. Souvent, les choix de binding sont peu compris, ou historiques.

Il est facile de montrer comment une connaissance superficielle peut aiguiller ces choix. Considérons le service ServiceIntervention et l'opération récupérer état demande. La signature en Java de ce service serait exprimée comme suit :

```
public Interface ServiceIntervention {…
     public String récupérerEtatDemande(     int numeroDemande,
                                             int numeroDistributeur);
…}
```

L'utilisation de Java est fortuite. De même que l'utilisation d'éléments de code a un simple caractère indicatif qui ne remet pas en cause une approche par les modèles. Ce service peut être vu de différente manière par le couple WSDL/SOAP.

#### *Le style RPC*

Commençons par les modes RPC/Encoded et Literal. Si RPC/Encoded n'est par conforme à WS-I Basic Profile 1.0, il sert d'illustration au propos pour montrer que l'impact RPC/Document Literal/encoded affecte uniquement la communication SOAP.

Le changement de encoded vers Literal impacte seulement la partie WSDL concrète. Le message SOAP est impacté.

Dans la figure 16.2, le contenu du message SOAP contient des éléments XML qui ne sont pas exprimés dans un schéma. On ne peut pas valider systématiquement ce contenu avec les techniques classiques offertes par la plate-forme XML. Seul le

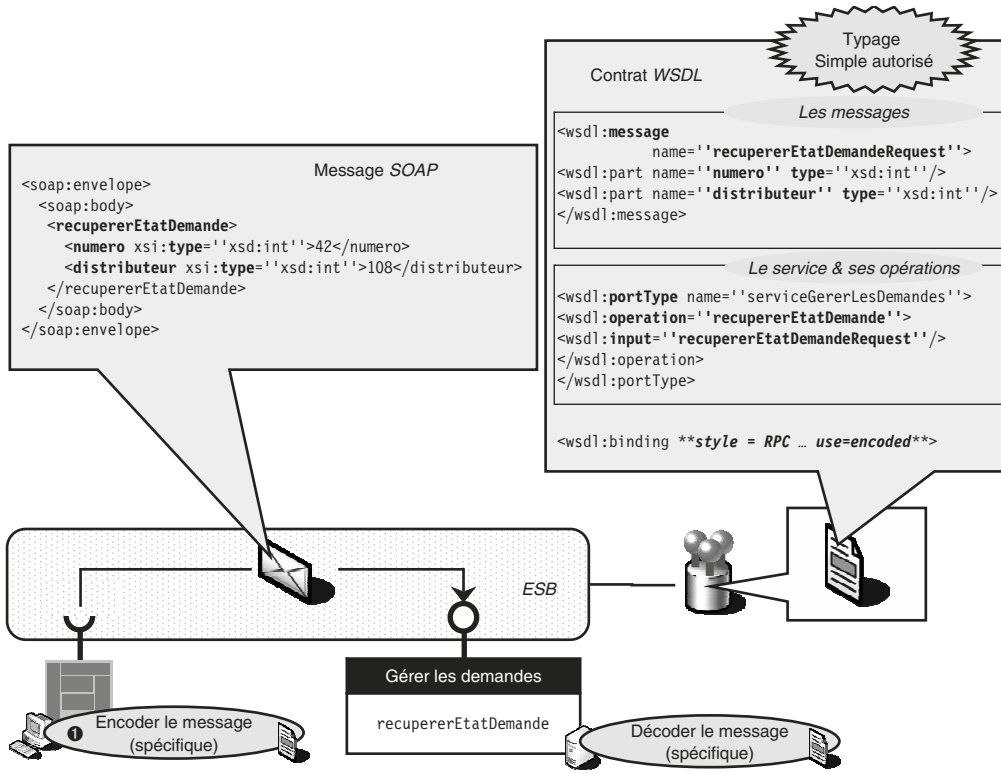



**Figure 16.2** — Style RPC/Encoded

type des « parts » des messages SOAP est accessible. Si une contrainte du WSDL est plus complexe qu'une validation superficielle de formulaire, il faudra traiter cette validation avec du développement et éventuellement l'utilisation de composants annexes.

Lors de l'utilisation du style RPC/Literal, seul le message SOAP sera simplifié, car il ne porte plus les instructions d'encodage décodage.

```
<soap :envelope>
     <soap :body>
          <recupererEtatDemande>
               <numero>42</numero>
               <distributeur>108</distributeur>
          </recupererEtatDemande>
     </soap :body>
</soap :envelope>
```

### *Le style Document/Literal*

Le mode Document/Encoded n'est pas du tout reconnu par la norme WS-I et reste une possibilité inusitée. Il est bon pour tous de l'ignorer et un exemple ne servirait pas le propos.

Le mode Document/Literal est en revanche important : dans ce mode, le WSDL doit maintenant obligatoirement contenir le schéma des messages.

Le message SOAP de la figure 16.3 contient les éléments NumeroDemande et NumeroDistributeur, éléments qui sont définis dans un schéma. On peut donc valider l'ensemble du message SOAP en se basant sur le seul schéma. Le moteur SOAP ou l'ESB pourra le faire. C'est la plus-value de ce mode. Il permet la validation des données entrantes d'un service avec l'ensemble de la technologie des schémas. C'est une garantie importante pour celui qui passe par exemple un contexte intégralement en paramètre.

En revanche, le WSDL est plus complexe à écrire. Cette complexité est en général prise en charge par la sophistication des outils de création/modélisation.

Un autre avantage du mode document réside dans le traitement du message SOAP par le récepteur : dans le mode Document/Literal, le récepteur n'est pas obligé d'attendre d'avoir reçu la totalité du message pour le traiter, il peut le traiter au fil de l'eau dès la réception des premiers octets. Exprimé plus techniquement, ce mode n'impose pas la lecture via DOM du message SOAP, il permet de donner accès à ce message via des API plus performantes comme SAX ou StAX. Ce point est crucial pour la mise à l'échelle et la performance d'une infrastructure de WebService. Cette spécificité technique permet également de maîtriser la consommation de mémoire des appels de services.

Lors du passage au style Document/Literal, le nom de l'opération a été perdu. Le code du service doit pouvoir décider de l'opération à effectuer selon les données

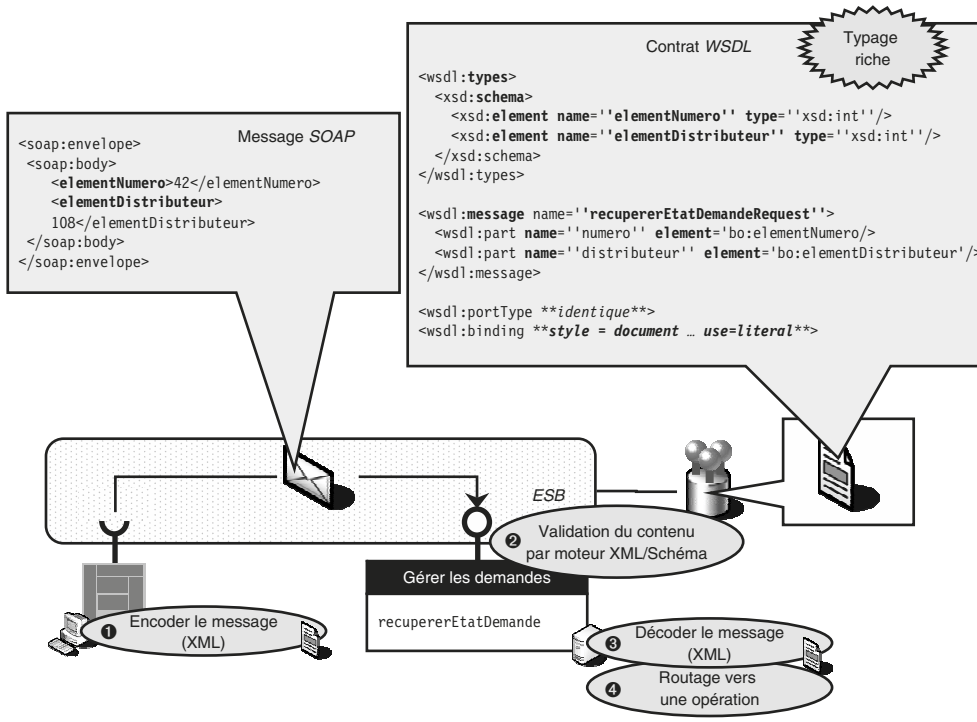



**Figure 16.3** — Style Document/Literal

du document. La programmation sera donc potentiellement plus complexe que lors du style RPC. De plus, le message SOAP n'est pas conforme à WS-I Profile qui interdit plusieurs enfants dans la balise <soap:body>. Cette restriction rend Document/Literal pas toujours conforme.

En résumé, ce mode offre deux avantages incontestables, au prix de trois inconvénients :

+ L'infrastructure SOA est capable de valider les messages reçus (vis-à-vis des schémas XML).
+ Les performances sont au rendez-vous.
– Le contrat WSDL du service est potentiellement plus complexe.
– L'implémentation du service est également plus complexe (elle doit découvrir elle-même le nom de l'opération appelée).
– Ce mode est conforme WS-I Basic Profile sous certaines restrictions.

### *Le style Document/Literal wrapped*

Enfin, il reste le mode Document/Literal wrapped, qui n'est pas dans la norme WSDL mais qui est massivement utilisé et défendu par Microsoft. Il s'agit d'ajouter le nom

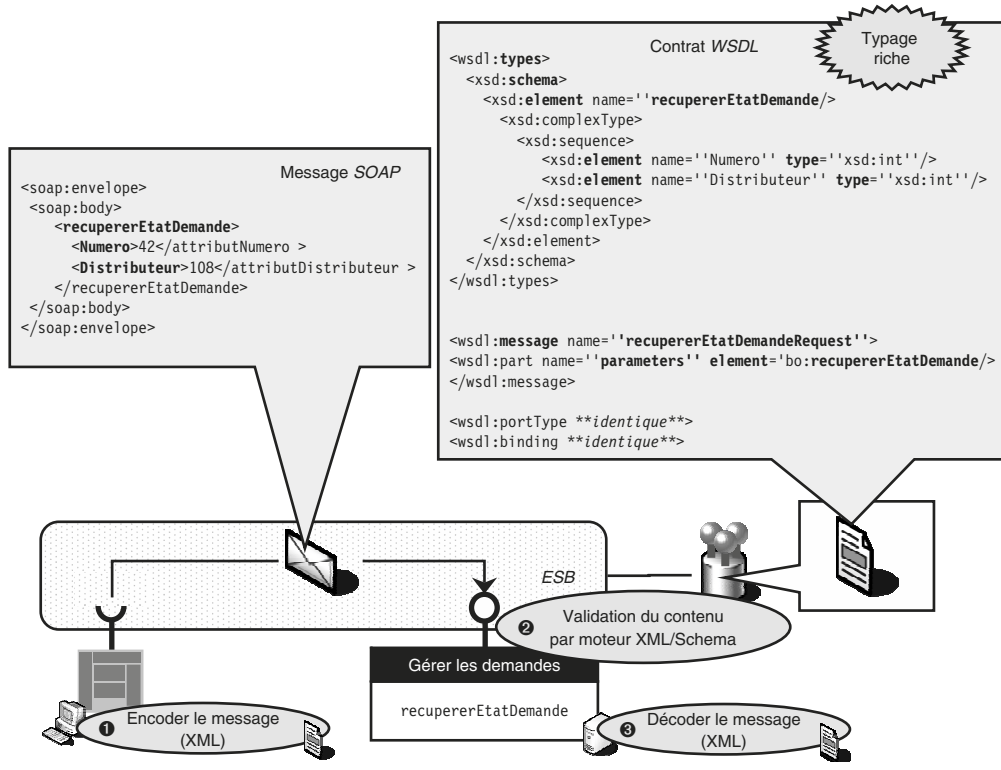



**Figure 16.4** — Style Document/Literal wrapped

de l'opération en élément parent des éléments du corps du message SOAP. Cette astuce permet de se conformer au WS-I Basic Profile car cet élément sera la balise <soap:body> contrairement au message précédent. Un premier intérêt, par rapport au mode Document/Literal est de supprimer la complexité de l'implémentation puisque le nom de l'opération est désormais présent dans le message.

Dans ce mode, le message en entrée a une **unique** <part> nommée « parameters » qui doit être un élément sans attribut et non un type complexe. Le nom de cet élément est le nom de l'opération. La conformité WS-I Basic Profile est assurée.

Le message en sortie aura des contraintes analogues et son nom sera le nom de l'opération suffixée par « Response », ce qui est une des contraintes de WS-I BP 1.0.

### 16.3.5 Bilan de l'utilisation des Styles WSDL

Formaliser les informations transportées dans les messages échangés par les services dans un schéma XML est payant, cela permet d'utiliser des outils de binding lorsque l'on traite ces informations sur une plate-forme donnée et de faire valider les appels au service. Les développeurs pourront traiter rapidement ces schémas pour faire apparaître les objets de transfert ou directement les objets métiers utilisés par les implémentations des services.

**Principe :** il est nécessaire d'utiliser systématiquement des schémas pour décrire ses données, à la place de simples contraintes de type associées à un élément.

Ces avantages légitiment une montée en compétences des équipes de maîtrise d'œuvre.

L'utilisation de style RPC est cantonnée à l'exploitation d'existant RPC encoded utilisé pour traiter des graphes d'éléments. Ce style non conforme WS-I BP 1.1 sera à migrer ou à traiter comme une contrainte historique.

L'usage du style Document/Literal Wrapped est aujourd'hui un standard. Bien qu'un peu plus complexe, il satisfait les critères techniques de performance, cohérence et interopérabilité.

**Principe :** favoriser l'usage du mode Document/Literal Wrapped autant que possible.

RPC encoded a été largement utilisé, il est donc utile de le connaître pour gérer l'existant, quitte à encapsuler ou faire migrer celui-ci., On peut aussi noter que le style RPC autorise le polymorphisme qui disparaîtra avec WSDL 2.0. Il s'agit donc d'une contrainte de compatibilité ascendante. Mais WSDL a le bon goût d'être en XML, donc facilement manipulable. Si un référentiel de service a été généré avec des outils, il y a de fortes chances de pouvoir migrer automatiquement l'ensemble des déclarations.

Le W3C offre un outil de conversion de WSDL 1.1 en WSDL 2.0. Cet outil est fourni par le W3C. Il fonctionne d'autant mieux que le WSDL est conforme à WS-I Basic Profile 1.0. L'intérêt en terme de pérennité de ces normes est démontré.

http://www.w3.org/2006/02/WSDLConvert.html

Voici un tableau récapitulatif des différents styles :

| Style | Avantages | Inconvénients |
|---|---|---|
| RPC/Encoded | Nom de l'opération accessible au distpatcheur recevant l'appel.<br>Messages et WSDL très simple.<br>Traite des graphes d'éléments. | **Non conforme WS-I Basic Profile.**<br>Impossible de valider les appels.<br>Verbeux par les informations de codage/décodage. |
| RPC/Literal | Conforme WS-I Basic Profile.<br>Nom de l'opération accessible.<br>Messages et WSDL très simple. | Impose une lecture DOM des messages, donc des contraintes mémoire. |
| Document/Encoded | N/A | Non conforme WS-I Basic Profile. |
| Document/Literal | conforme WS-I Basic Profile, mais avec des restrictions.<br>Permet une lecture via SAX ou StAX pour ne pas être contraint par la taille des messages.<br>Validation du message possible. | Messages et WSDL complexe.<br>Le nom de l'opération a disparu : le dispatching peut devenir complexe. |
| Document/Literal Wrapped | Conforme WS-I Basic Profile.<br>Validation du message possible.<br>Le nom de l'opération accessible. | Wrapped n'est pas une spécification WSDL.<br>Mais cela devient un standard de facto.<br>Messages et WSDL encore un peu plus complexe. |

## En résumé

**WSDL est un formalisme de description de service ouvert à de multiples technologies.** C'est un élément de pérennité et de convergence. Il est un moyen d'Assurer les enjeux fonctionnels en respectant une approche par les modèles.

La formalisation WSDL des services Legacy est un plus non négligeable afin de permettre aux équipes nouvelles technologies de composer avec ces services.

**La conformité avec WS-I est payante :** l'interopérabilité est au rendez-vous, ainsi que la performance et la pérennité.

**L'usage du style Document/Literal Wrapped est aujourd'hui un standard.** Il rassemble les critères techniques de performance, cohérence et interopérabilité.

Les outils gèrent tous les styles ainsi que la conformité WS-I pour la plupart : grâce à ces outils, utiliser ces normes et les spécificités de WSDL reste simple.

L'approche par les modèles qui permet un plus grand contrôle des WSDL utilisés semble encore une fois plus adaptée.

# 17

# Choisir la technologie d'implémentation

**Objectif**

Ce chapitre aborde les choix de déploiement des services en fonction des contraintes techniques. Il s'agit de ne pas hypothéquer la future cartographie technique du SI par des choix trop restrictifs. La maîtrise d'œuvre doit systématiser sa réflexion pour avancer le plus tôt possible sans avoir à connaître toutes les contraintes dès le début.

## 17.1 RESPECTER LA GRANULARITÉ DES SERVICES

La typologie des services évoquée dans la partie 3 fait ressortir plusieurs niveaux. La granularité sera un facteur clef de sélection des technologies pour les services de plus haut et de plus bas niveau.

On peut déployer le même service de deux manières différentes. Il s'agit de deux instances du même service avec différents protocoles, voire même différentes implémentations.

Le Service Applicatif, de plus haut niveau sera le type même de service gros grain à exposer dans le SI étendu. Sa granularité en fait un service à forte valeur dont l'exécution peut justifier une performance moyenne.

Les services fonctionnels peuvent offrir une grande variété de granularité et d'utilisation. Ces services sont faits pour être agrégés en un service de plus haut niveau. On trouve tous les cas d'implémentation à ce niveau, selon le besoin de performance et de contrôle de la qualité de service.

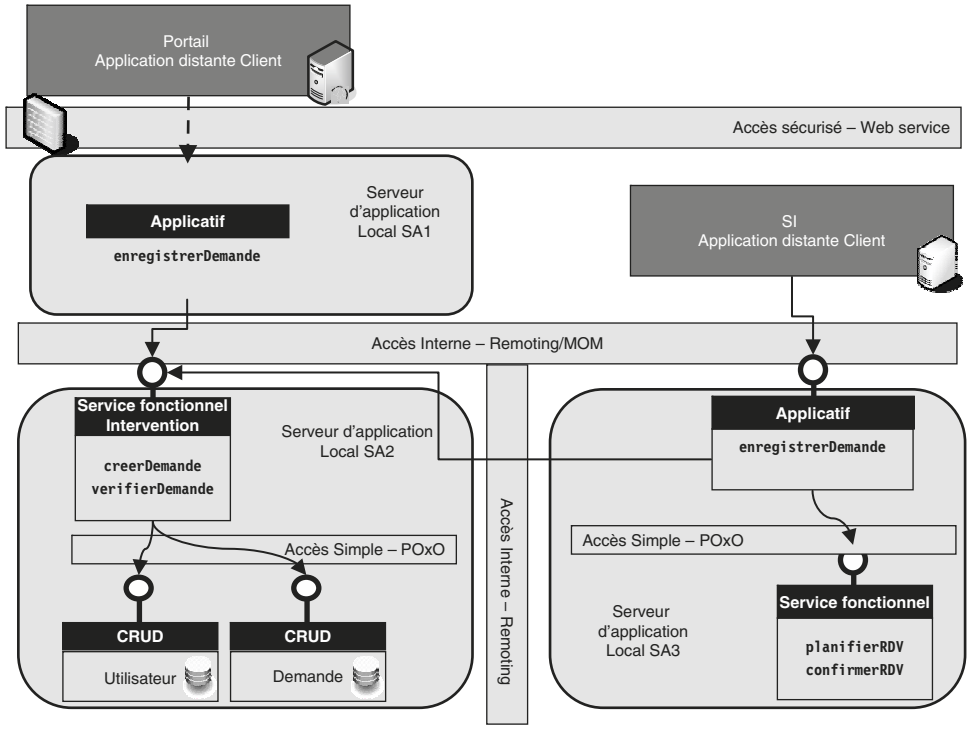



**Figure 17.1** — Typologie et déploiement de services

**Tableau 17.1** — Granularité et typologie

| Type de service | Performance | Interopérabilité avec l'extérieur du SI | Contraintes techniques |
|---|---|---|---|
| Service métier applicatif (SA) | Ces services représentent souvent des opérations complexes pour lesquelles un client peut accepter un temps de traitement lourd. On peut aussi les utiliser dans des contextes avec un besoin de SLA. | On ne peut pas imposer la plate-forme technique au client du SI Étendu. | Sans contraintes de performance majeure, les solutions WebServices sont les plus indiquées. On peut aussi redéployer des versions SLA du service, soit en Remoting soit avec des ESB assurant le SLA. |
| Service métier fonctionnel (SF) | Les temps de traitements sont variables selon les fonctionnalités. | Dépend du contexte. On doit se prémunir de choix qui hypothèquent le futur et offrir toutes les possibilités. | Souvent des besoins de transactions et de distribution qui imposent une façade Remoting. Certainement le plus ouvert à tous les cas techniques. |
| Service métier CRUD | Les temps d'appel et d'exécution doivent être négligeables vis-à-vis des services appelant. | Sans objet : un service CRUD ne doit pas être directement accessible. | Faible. De simples POxO doivent pouvoir réaliser ces services. |

Les services CRUD doivent rester au sein du SI Local et ne gagnent rien à être distribués. Ils seront les représentants les plus simples de cette typologie.

## 17.2 TRIPTYQUE SALUTAIRE

Un service sera soit un WebService, soit un simple objet, soit un service distribué de type Remoting. Cette approche peut se systématiser en utilisant le WSDL de description d'un service pour produire systématiquement cet ensemble :

- une interface et une implémentation du service sous sa forme la plus simple (PoxO) ;
- accompagné d'une façade technologique Web Services ;
- d'une ou plusieurs façades Remoting (synchrone et/ou asynchrone).

Les outils de génération de code produisent couramment au minimum une façade WebService et une implémentation Remoting à partir du WSDL. On peut aussi générer ses propres implémentations (squelette de code) s'il est utile de conserver une certaine indépendance vis-à-vis des outils et des serveurs d'applications utilisés.

Pourquoi systématiser ce jeu de façades ? Afin de créer un contexte d'utilisation maîtrisable. L'implémentation systématique sous forme de POxO permet de constituer des équipes de développement de services constituées de développeurs peu spécialisés, sans devoir reposer sur des experts de toutes les technologies de Remoting et des WebServices.

Cette démarche permet de ne pas avoir à passer trop de temps sur le mode d'utilisation du service avant sa production, qu'il soit synchrone ou asynchrone, distribué ou non, déployé sur plusieurs sites ou dans un unique environnement, etc. Au besoin, selon les contraintes et les possibilités techniques de déploiement, on pourra utiliser telle façade ou bien le service POxO directement.

Les gains sont multiples :

- Uniformiser l'expression des contrats de services, en conservant dès leur identification la sélection des technologies présentées dans les tableaux précédents.
- Déduire l'implémentation POxO (squelette de code) du contrat WSDL.
- Déduire l'interface Remoting du contrat WSDL.
- L'implémentation du traitement sera indépendante de la visibilité technique de l'infrastructure.
- Une grande partie des choix sera reportée à la phase d'intégration où l'on est mieux informé de l'ensemble des contraintes à prendre en compte.

Les coûts sont connus. En systématisant cette approche, on connaît la surcharge de production d'un service selon ses équipes et ses contraintes de développement. Il sera alors possible de décider si le coût constaté est compensé par la flexibilité apportée. Ce bilan est spécifique aux enjeux d'une entreprise qui lui permettra une évaluation précise de ses priorités.

## 17.3 STANDARDISER L'APPEL AU SERVICE

Voir les services au travers d'intermédiaires permet de stabiliser le code de l'appelant, ce qui est fondamental dans une démarche d'application composite appelant divers services susceptibles d'évoluer.

La solution intégrée est l'ESB (bientôt associé à un container SCA), qui permet de paramétrer quel service répond à un appel et sous quelles conditions (noms, contrat, etc.). Cette solution décharge les développeurs des complexités et contraintes d'accès à un service, pris en charge par un module d'infrastructure de services. On retrouve le mode de fonctionnement des serveurs d'application qui prennent en charge la complexité de déploiement, sécurité, activation, etc., des applications.

Les ESB ne sont pas indispensables dans tous les contextes. En attendant les conteneurs SCA, il reste des solutions programmatiques utilisant des Proxy et des conteneurs légers. Le framework Spring propose une gestion déclarative des services basés sur l'injection de dépendances. Ce type de produit permet notamment l'utili-

**Spring : un conteneur de services léger**

Spring est une des solutions légères d'assemblage de service les plus en vogue. Utilisable en environnement J2EE et .NET, ce framework technique est doté d'un ensemble de composants. Sa plus value majeure au sens des services est le conteneur léger de Spring, appelé aussi conteneur d'injection de dépendance.
L'injection de dépendance est une technique qui permet d'établir dynamiquement ou statiquement l'association (ou dépendance) entre le consommateur d'un service et une implémentation spécifique, au travers d'une interface. Le consommateur n'a pas de connaissance de l'implémentation particulière, il ne connaît qu'une interface (Java ou C#). C'est l'intégrateur qui choisit l'implémentation ciblée au moment du déploiement. Ce mécanisme est particulièrement adapté aux changements d'implémentation, pour des besoins de tests ou de bouchon, ou d'adaptation à des contraintes techniques différentes selon les cas d'utilisation.
Ce composant, qui connaît un grand succès, s'oppose aux conteneurs J2EE ou .NET qui offrent une surenchère de fonctionnalités. Il s'agit ici de fournir une gestion automatisée et déclarative de la configuration et de l'association des consommateurs et des implémentations de services au sein d'une application. La configuration permet de dire quelle classe d'implémentation fournira quel service logique et de lier au démarrage tous les services réels ainsi définis. C'est la base d'un couplage lâche entre services. On obtient du coup une grande flexibilité dans le déploiement des services : par exemple, la mise en place de services « bouchons » sur des plates-formes de développement ou de tests est immédiate. L'association d'un ensemble d'implémentations est déclarative et le développeur n'a pas besoin de connaître plus que l'interface d'utilisation du service.
Spring offre donc une base technique et une approche par les interfaces conforme à la philosophie SOA. C'est la raison pour laquelle il a fortement inspiré la norme SCA (cf. partie 4, chapitre 14).

sation des principes SOA sur des infrastructures de petite et de grande taille sans déployer de solution surdimensionnée. D'autre part, on trouve aussi des solutions comme WSIF[1] qui propose à l'instar de SCA d'atteindre un service défini en WSDL dont l'implémentation n'est pas limitée à la technologie Web Service.

Ces concepts de solutions SOA légères procèdent toutes des principes OCP (*Open Close Principle*) du développement Objet : le code doit être fermé aux modifications et ouvert aux extensions. On veut qu'une évolution métier ne change pas du code déjà testé. Seul l'ajout de code est toléré.

**WSIF : un adaptateur de service**

WSIF, ou *Web Service Invocation Framework*, est une initiative IBM qui se pose en intermédiaire de l'appel au service afin de stabiliser le code appelant et de pouvoir opter pour la performance maximale sans impact sur le code de l'application métier. Ce projet a été donné en 2002 à la fondation Apache par IBM. On trouve une version encore plus avancée dans le serveur d'application Websphere. Ce composant Java d'invocation de service veut stabiliser le code de l'appelant. L'appel du service est uniquement basé sur la partie abstraite du WSDL. L'objectif de fournir un appel de service standardisé est conforme au design Orienté Objet et aux problématiques SOA. Le développeur utilisant WSIF n'interagit pas avec le protocole choisi, SOAP ou autre. Seule l'interface définie par le WSDL est exposée.
À l'appel, WSIF permet par extension du WSDL de définir des binding classiques SOAP comme des binding vers JAVA (accès POJO) ou EJB ou JMS.
WSIF permet une invocation via un proxy généré ou dynamiquement découvert. Il permet d'appeler tout service défini en WSDL au travers de son API disponible aujourd'hui en Java.

Au sein du WSDL, on peut déclarer plusieurs binding. Le choix du binding à utiliser peut aussi être déclaré à l'utilisation. Cela permet de configurer les appels de services sans intervenir sur le code applicatif.

Les différents binding correspondant au SOAP traditionnel ou à une extension WSDL sont gérés par un composant nommé Provider. Les Providers disponibles gèrent aujourd'hui Java, SOAP, EJB et JMS. C'est le Provider qui gère toute la spécificité technique de ces protocoles et extensions. Grâce à divers Providers, le changement de moteur SOAP utilisé devient transparent au sens du service. Il est possible de créer ses propres providers.

WSIF est une solution élégante pour les moteurs BPEL qui doivent appeler des services sans payer systématiquement le prix d'appel WebService au sein de son SI. http://ws.apache.org/wsif.

On cherche la stabilité de l'appelant, une continuité annoncée du principe CAF qui avance par composition de services (voir le chapitre 5 de la partie 2 sur les variantes de services). Les réponses à base de Spring ou WSIF proposent de déployer

1. WSIF : *Web Services Invocation Framework*. Projet du groupe Apache.

ces techniques sans solliciter des infrastructures très complexes inutilement. Le succès de ces concepts vient de cette simplicité : tout le monde n'a pas besoin d'assurer de la haute disponibilité et des services transactionnels synchrones distribués. Du moins, tous les services du SI n'ont pas ces contraintes.

Cette spécification annonce des produits WSDL-centriques dont la continuité sera SCA, Il s'agit d'outils et d'infrastructures de composants qui misent sur l'avenir du WSDL utilisé comme un formalise pivot. Ces outils se conforment aux normes et s'ouvrent aux extensions WSDL pour prendre en compte le plus de diversité technique possible. Les extensions WSDL sont les socles pour offrir de nombreux protocoles et modes de localisation. Ils impactent plus particulièrement les parties concrètes des descriptions de services. C'est le cas de WSE[1] de Microsoft, qui apporte la compatibilité avec Windows Communication Framework, ou des extensions communément offertes par les ESB : protocole de transfert FTP, MOM, sécurité et signature électronique, etc.

## En résumé

La Capitalisation WSDL permet de facilement accéder à un service quel que soit le mode de déploiement choisi parmi le triptyque WS/Remoting/Simple Composant.

Disposer de toutes ces facettes techniques d'un service permet ainsi d'être flexible.

Les choix de déploiement dépendent des typologies de services et des contraintes techniques des plates-formes de production.

Les solutions ESB compatibles SCA permettront de mieux en mieux un assemblage et un appel des services standardisés, conformément à la philosophie de Spring. Cette évolution déplace la complexité vers l'administration de la plate-forme, en suivant l'exemple des serveurs d'application.

Enfin, du côté des consommateurs de service, une technologie telle que WSIF, qui préfigure en partie SCA, permet de masquer complètement le triptyque à un consommateur : celui-ci invoque le service de façon totalement indépendante du mode de déploiement du service, ce qui représente le stade ultime de l'indépendance de SOA vis-à-vis des technologies et des middlewares.

1. *Web Services Enhancement* : amélioration et extensions proposées par Microsoft.

# SIXIÈME PARTIE

# SOA et Web 2.0

Les parties précédentes du livre, en particulier les parties 3 et 5, portent essentiellement sur l'utilisation de SOA pour rénover le Système d'Information (SI), en conservant tout ou partie des applications existantes. Cette rénovation s'accompagne d'un effort de modularité afin de rendre ainsi le SI plus flexible. La mise en place de nouveaux processus métier ainsi que l'ouverture du SI à d'autres SI sont les principales justifications (métier) de l'adoption d'une telle approche SOA.

Or, le thème de l'ouverture du SI aux clients de l'entreprise prend de plus en plus d'importance avec l'apparition de la vague « Web 2.0 ». Il s'agit de permettre à ces clients (ainsi qu'aux prospects, partenaires, etc.) d'interagir directement avec l'entreprise, sans intermédiaire.

Le Web 1.0 faisait de l'internaute un lecteur d'information, un acheteur, et un navigateur dans le réseau.

Le Web 2.0 prétend désormais faire de l'internaute un acteur beaucoup plus dynamique, avec les avantages et les inconvénients que cela suppose pour l'entreprise. L'internaute crée désormais de l'information : il donne son avis sur le site de l'entreprise, anime un blog, discute avec d'autres clients sur un wiki, crée de la rumeur ou contribue à évaluer la qualité des services et des produits, explique comment il utilise ou il a abandonné tel ou tel produit… Certains pensent qu'il faudrait ouvrir également ce type de fonctions aux salariés de l'entreprise.

L'objectif de la présente partie est de proposer une réflexion sur ce type d'ouverture du SI, et ainsi de positionner SOA vis-à-vis du Web 2.0.

On notera d'emblée que le présupposé n'est pas d'opposer ici SOA et Web 2.0, comme tentent de le faire quelques évangélistes zélés ayant découvert une nouvelle terre promise. SOA est avant tout une approche d'urbanisation, de modélisation et d'architecture : elle n'est pas liée à une technologie particulière – même si les Web Services y jouent un rôle important. Le Web 2.0 offre de nouvelles perspectives pour construire une SOA et ce sont ces perspectives qui sont au cœur de la présente réflexion.

Pour préciser le périmètre de cette réflexion, il faut bien voir que l'ouverture du système d'information via le Web 2.0 recouvre deux grandes facettes :

- une facette fonctionnelle, qui voit la mise en place de nouveaux outils & usages :
  - outils collaboratifs de création et de partage d'info : blog, wiki, espace « lecteurs » des journaux électroniques;
  - outils de classification collective des infos;
  - réseaux sociaux, etc.
- une facette architecturale et technique, qui voit l'émergence de nouveaux outils et techniques : éditeurs de Mashups, composants AJAX, services REST.

La réflexion se concentre sur la facette technique, la question centrale étant : peut-on faire du SOA avec ces technologies Web 2.0 ? Si oui, comment ? (de la même façon qu'il y a 10 ans, la question était : peut-on faire des applications informatiques client/serveur avec les techniques Web 1.0 ?). La simplicité est un facteur souvent mis en avant par ces évangélistes Web 2.0 : est-ce une promesse tenable ?

L'objectif de cette partie est donc de présenter ces différents concepts et d'en proposer une analyse aussi objective que possible, en ayant toujours à l'esprit qu'un Système d'Information est un peu plus complexe qu'un « simple » site Web. En bref, peut-on commencer à parler de SOA 2.0 ?

Cette partie comprend deux chapitres. Le chapitre 18 propose une analyse des concepts architecturaux fondateurs des applications interactives du Web 2.0. Le chapitre 19 présente le style d'architecture REST, et le positionne par rapport au style d'architecture proposé par les Web Services.

# 18

# Architectures Web 2.0

## Objectif

L'objectif de ce chapitre est de présenter l'approche Web 2.0 sur le plan architectural.

Il présente d'abord la philosophie SOA du Web 2.0, en pointant les différences d'approche avec une SOA « classique » (c'est-à-dire telle que présentée dans les chapitres précédents).

Puis il propose un zoom sur les outils et technologies permettant de construire un Mashup, c'est-à-dire une application composite Web 2.0.

## 18.1 SOA ET WEB 2.0 : DEUX APPROCHES DIFFÉRENTES ?

Comme on l'a vu (chapitres 4 et 8 notamment), l'ouverture du SI avec SOA a débuté par la réalisation et la publication de Web Service. Ces Web Services sont essentiellement des façades : elles recueillent les requêtes issues de l'extérieur du SI, effectuent les contrôles de sécurité nécessaires, puis transfèrent la requête vers le SI « cœur de métier ». Une telle requête sera traitée (en général) par un processus métier.

Un Système d'information est classiquement structuré (cf. chapitre 4, figure 4.5) selon une architecture à trois étages : étage « applications interactives », étage « logique métier », étage « informations & référentiels ».

L'approche proposée jusqu'ici peut être résumée en disant que SOA structure puis donne accès à l'étage « logique métier » de l'entreprise. L'objectif d'une SOA est de construire au sein de cet étage une hiérarchie de services, en jouant sur la granularité et la composition de services. Ces services sont utilisés par des applications composites : processus ou applications interactives. Pour construire ces services, on mixe plusieurs approches :

- approche top down : identifier les services applicatifs nécessaires pour les applications, ce qui implique de modéliser ces applications clientes (cf. chapitres 9 et 10) ;
- approche bottom up : identifier les services CRUD permettant d'accéder aux informations métier, en particulier les informations structurées (stockées dans les bases de données relationnelles) ;
- approche « meet in the middle » : identifier les services fonctionnels, construire les services applicatifs par composition de services (CRUD, fonctionnels, techniques).

Le Web 2.0 propose une vision quelque peu différente. Tout part en fait des données. Ces données ont en général deux caractéristiques : d'une part elles ne sont pas ou peu structurées (il peut s'agir de documents, d'images, de vidéos, etc.), et d'autre part elles sont distribuées dans différents sites Web (comme ceux de Google, Amazon, Wikipédia, eBay, etc.).

Le premier objectif du Web 2.0 est d'abord de fournir à tout un chacun ces données, via l'approche architecturale **REST**. Il s'agira, en bref, de mettre en place des services d'accès CRUD aussi simples que possible.

Le second objectif est de permettre de construire, aussi rapidement que possible, des applications exploitant ces services d'accès : c'est le concept de **Mashups**. Un Mashup est une application composite au sens strict du terme, puisqu'il s'agit de combiner plusieurs services (REST principalement, mais pas uniquement) pour construire une application finale. Cette application est exécutée dans un contexte « navigateur Web » et s'appuie fortement sur JavaScript.

Si l'on revient à la structuration en étage du SI, évoquée en début de section, alors on peut dire que, via le concept de Mashups, et la technologie **AJAX**, le Web 2.0 ouvre l'étage « applications interactives » aux internautes. Via le concept **REST**, le Web 2.0 propose une nouvelle façon d'accéder aux informations de l'étage « bases d'informations ».

La figure 18.1 présente cette approche en regard de l'approche SOA « classique ». Elle suppose qu'un Mashup peut aussi bien être utilisé par un internaute pour interagir avec l'entreprise, que par un salarié au sein même de l'entreprise.

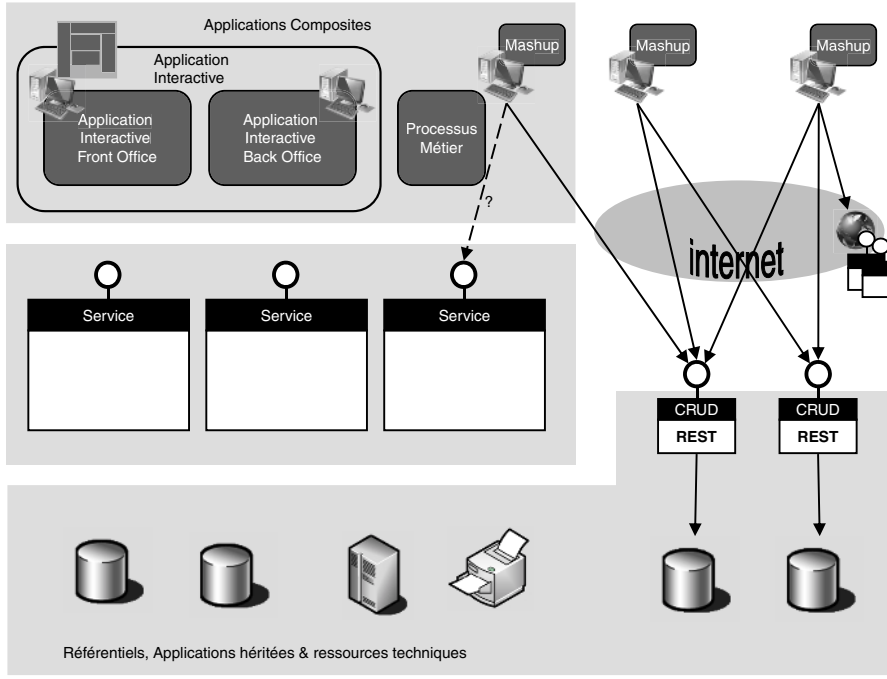



**Figure 18.1** – Web 2.0 versus SOA ?

Les sections suivantes présentent une analyse architecturale de l'évolution des applications interactives via le Web 2.0.

## 18.2 LES MASHUPS : PRINCIPES, AVANTAGES, PROBLÈMES

Les Mashups sont des applications composites publiées et accessibles sur le Web. Les services appelés sont des services fournis par différents sites Web. L'objectif premier d'un Mashup est d'agréger (intégrer, synthétiser) des informations provenant de différentes sources.

### 18.2.1 Un exemple représentatif

Dans le cadre de l'exemple général Fil Rouge, présenté au chapitre 7 (section 7.2), un exemple classique d'un Mashup consistera à visualiser sur une carte (géographique ou satellite) les Points De Distribution, ou PDD, (c'est-à-dire les compteurs à gaz et/ou les compteurs électriques), approvisionnés par un producteur X et situés sur une commune dont l'utilisateur du Mashup aura saisi au préalable le nom. C'est un Mashup, orienté SI, puisqu'il s'agit d'agréger des informations géographiques peu structurées (une carte) et des descriptions structurées (la description de chaque PDD).

La logique de cette application est décrite dans le tableau 18.1.

**Tableau 18.1** – Cas d'utilisation du Mashup « Fil Rouge »

| Étape | Description de l'étape | Service appelé | Fournisseur de l'information |
|---|---|---|---|
| Étape 1 | L'utilisateur fournit à l'application la zone géographique concernée, par exemple le département. | Service de recherche des départements français. | Base de données géo-postales, fournie par ou achetée à un Service Postal. |
| Étape 2 | L'utilisateur précise, s'il le souhaite, la zone concernée, par exemple en choisissant la commune concernée. | Service de recherche des communes du département choisi à l'étape 1. | Base de données géo-postales. |
| Étape 3 | L'application récupère la liste des PDD présents dans la commune désignée et approvisionnés par X. | Service de recherche des PDD. | Base de données industrielles, propre à l'entreprise. |
| Étape 4 | L'application récupère une carte correspondant à la commune désignée. | Service de lecture d'une carte. | Il existe de nombreux fournisseurs de carte, dont le premier a été Google Maps, mais aussi Yahoo Maps, via Michelin, etc. |
| Étape 5 | L'application affiche la carte et les PDDs. | | |

Ce type d'exemple, bien que simple, est désormais très courant, avec l'explosion de l'usage des informations géo référencées, non seulement sur le Web, mais aussi désormais sur les téléphones mobiles accédant au Web et au GPS.

La figure 18.2 donne un exemple de rendu, construit avec Google Maps ©.


**Figure 18.2** – Un Mashup orienté données géo référencées

### 18.2.2 Mashup et SOA

On constate l'importance de l'orientation Service pour construire un Mashup. Mais en quoi cet exemple se distingue-t-il des applications composites classiques, décrites au chapitre 10 ?

#### *Interopérabilité*

Une application faisant appel à des services est dépendante de ces services sous trois aspects :

- elle dépend du nom de l'opération invoquée : getMap(), rechercherDépartementsDeFrance(), etc.;
- elle dépend de la localisation du service (son adresse), mais sur ce point il n'y a pas de différence fondamentale entre les approches;
- elle dépend de la définition et du format des informations échangées via cette opération.

Un Mashup invoque un service en respectant la philosophie REST (présenté au chapitre 19) : on notera que cette philosophie implique que le nom des opérations soit standardisé, contrairement à une approche SOA classique. Ces noms standardisés sont en fait ceux des commandes HTTP (GET, POST, etc.). Un Mashup ne dépend donc pas du nom des opérations offertes par un service REST.

En revanche, elle dépend fortement du format des informations, c'est-à-dire de la façon dont ces informations sont encodées pour voyager sur la toile. A priori, cet encodage devrait être basé sur XML. Cependant, XML étant considéré comme « complexe » par certaines communautés Web 2.0, les services REST renvoient souvent leurs informations encodées selon des formats supposés plus simples. Un format populaire est le format JSON.

JSON veut dire *JavaScript Object Notation*. JSON est un format textuel reprenant le formalisme des objets JavaScript. Autrement dit, JSON est un moyen de transporter des objets JavaScript en les sérialisant/dé-sérialisant. Voici un exemple d'encodage JSON :

```
{"localisation" :
          {"id" :  "36290",
           "commune" : "MeziereEnBrenne",
           "destination" : "maison de la pisciculture",
           "adresse" : "3 place de la mairie"
          }
}
```

Dans la perspective adoptée dans ce chapitre, l'intérêt est qu'un document encodé en JSON est immédiatement transformable en un objet JavaScript, donc manipulable facilement par tout script JavaScript, c'est-à-dire par tout Mashup. La méthode la plus immédiate (et la plus rapide) pour ce faire est :

```
Var monObjet = eval(document_json);
```

Mais si le document contient un code malicieux, eval conduit à une catastrophe. La méthode préférable est d'utiliser un « parseur » disponible sous JavaScript, qui n'exécute pas mais décode et transforme. On fera cependant attention aux performances dudit « parseur ».

### *Ergonomie*

L'objectif des créateurs des premiers Mashups était non seulement fonctionnel : l'agrégation d'informations, mais également technique : une réalisation aussi simple et aussi proche du Web que possible, et complètement indépendante des querelles de chapelle linguistiques (Java versus C# versus PHP versus Ruby, etc.).

Pour satisfaire cet objectif, la construction des Mashups s'appuie fortement sur les standards du Web : HTML et JavaScript. Construire l'application décrite dans le tableau 18.1, c'est, dans la logique Web, avant tout écrire des pages HTML (pour l'affichage des informations et le dialogue homme-machine).

Cependant, ce retour aux sources du Web 1.0 est apparu insatisfaisant : le dialogue Web classique (c'est-à-dire via le protocole HTTP) oblige chaque interaction utilisateur à recharger une page HTML complète, et de plus offre des composants graphiques limités et plus que rustiques. C'est la raison pour laquelle est apparu AJAX, « cœur » architectural du Web 2.0 et des Mashups, et objet de la prochaine section.

## 18.3 AJAX : PRINCIPES, AVANTAGES & INCONVÉNIENTS

La création d'AJAX a pour objectif d'offrir une ergonomie décente aux utilisateurs du Web 2.0. Il s'agit donc :

- d'offrir des composants graphiques (nettement) plus évolués que ceux offerts par HTML ;
- d'offrir un confort d'utilisation accru : il s'agit d'éviter le rechargement systématique d'une page HTML à chaque action de l'utilisateur, qui nuit à la réactivité des interfaces homme-machine ;
- de s'appuyer néanmoins sur le Web et ses ingrédients de base : le navigateur, HTML, JavaScript, HTTP.

Dans le Mashup décrit par le tableau 18.1, les composants graphiques évolués seront par exemple un champ de saisie « auto complétant » pour la saisie du département, ou un composant « afficheur de carte », etc.

La figure 18.3 illustre[1] le fonctionnement d'un composant de type champ de saisie « auto complétant » : dès que l'utilisateur a saisi la première lettre du département

1. Cette illustration a été obtenue avec la bibliothèque AJAX EXT JS : http://extjs.com/. Il est important de noter que la recherche est lancée à partir de la saisie du x-ième caractère, x étant paramétrable.

recherché, le composant lance une requête de recherche (sans rechargement de la page accueillant ce composant) et affiche le résultat dans une liste (liste paginée de surcroît, puisque la liste obtenue peut être longue).


**Figure 18.3** – Un exemple de composant AJAX

## 18.3.1 Le principe de fonctionnement AJAX

### *AJAX s'appuie sur trois fonctions clefs du Web*

Pour satisfaire ces objectifs, AJAX s'appuie sur trois fonctionnalités clefs du navigateur et de son acolyte JavaScript.

D'abord, la capacité du navigateur à détecter des événements utilisateurs (cliquer avec la souris, positionner le curseur dans un champ de saisie…) et à invoquer une fonction JavaScript lorsqu'il détecte un tel événement. Ce sont les fameux `onClick=fonction1()`, `onFocus=fonction2()`, etc.

Ensuite, AJAX exploite les propriétés d'un objet central : `XMLHttpRequest`. Cet objet encapsule l'appel à un service via HTTP, plus exactement l'appel à une URL Internet, puis transmet le résultat de cet appel à une fonction appelée CallBack. Cette fonction callback est écrite par le développeur AJAX.

Il est important de noter que cet aller et retour n'implique pas le rechargement de la page HTML qui contient le code d'utilisation de `XMLHttpRequest`. Le résultat de l'invocation de l'URL est a priori encodé en XML, d'où la signification de AJAX : *Asynchronous* (parce qu'on ne recharge pas la page) *JavaScript* (parce que le code est écrit en JavaScript et inclut dans une page HTML) *And XML*. Sauf que dans certains cas, on n'utilise pas XML mais JSON.

La troisième fonctionnalité clef est le fait qu'une fonction JavaScript peut manipuler la page HTML qui l'invoque, pour en modifier le contenu. C'est justement ce qui est souhaité : la fonction callback évoquée plus haut aura précisément pour objectif de modifier la page HTML pour afficher le résultat de l'invocation de `XMLHttpRequest`.

Cette manipulation s'effectue via DOM, *Document Object Model*. DOM spécifie comment les éléments d'une page HTML sont manipulables directement dans une fonction JavaScript.

### *Fonctionnement d'une application AJAX*

Cette section décrit le principe de construction d'une application AJAX telle que celle présentée dans le tableau 18.1. Le concepteur commencera par inclure dans le corps de la page (statique) HTML un bouton qui, une fois le département puis la commune saisis par l'utilisateur, permettra de lancer (via la gestion d'événement) l'affichage de la carte (cf. figure 18.1) :

```
<p><input type="button" value="afficher la carte"
onClick="recupererPDDEtAfficherUneCarteAvecAjax();"></p>
```

On définit ensuite, toujours dans la page HTML, l'endroit où sera affichée la carte :

```
<div id="google_map_div"
     style="width : 500px; height : 300px">
</div>
```

Puis on définira (dans la page HTML), les fonctions JavaScript nécessaires.

La première fonction est la fonction appelée lors du clic sur le bouton : `recupererPDDEtAfficherUneCarteAvecAjax()`. Code exemple :

```
<script language="javascript" type="text/javascript">
var requete = new XMLHttpRequest();
function recupererPDDEtAfficherUneCarteAvecAjax() {
//étape 1
requete.open("GET", URL du service de recherche PDD, true);
//étape 2
requete.onreadystatechange = ajaxCallback_AfficherCarte;
//étape 3
requete.send(null);
}
```

Première étape de cette fonction : elle précise à l'objet `XMLHttpRequest` le service à invoquer (il s'agit du service de recherche des PDD).

Deuxième étape : elle précise à cet objet `XMLHttpRequest` la fonction callback que l'objet devra appeler une fois que le service invoqué aura renvoyé son résultat. La fonction callback affichera le résultat obtenu.

Troisième étape : la fonction demande à l'objet `XMLHttpRequest` de s'exécuter, c'est-à-dire d'envoyer la requête au service.

La fonction callback, `ajaxCallback_AfficherCarte`, affichera une carte puis les différentes adresses. Elle utilise pour cela des objets et fonctions JavaScript, notamment l'objet représentant une carte, GMap2, qui sont fournis par Google (ou le fournisseur choisi). Voici ce que serait cette fonction (dans une version simplifiée pour le propos) :

```
function ajaxCallback_afficherCarte() {
//la fonction crée une nouvelle carte via un objet javascript GMap2
//(défini par Google), en précisant à cette carte où elle doit
//s'afficher dans la page HTML
var carte = new GMap2(
           document.getElementById("google_map_div"));
//le résultat de la requête est disponible dans l'objet XMLHttpRequest :
//var resultat = requete.responseText
//pour chaque adresse de PDD dans le résultat, la fonction crée
//et affiche ce que Google appelle un marqueur, cf les objets
//de l'illustration 18.2. Les fonctions de création de marqueur :
//Gmarker(), et d'affichage d'un marqueur sur une carte : addOverlay(),
//sont fournies par Google.
For(i=0; i< resultat.length; i++){
             var adresse =
resultat.localisations.localisation[i].adresse
             var marqueur = new GMarker(adresse);
             carte.addOverlay(marqueur);}
}
</script>
```

La figure 18.4 synthétise ce principe de fonctionnement.

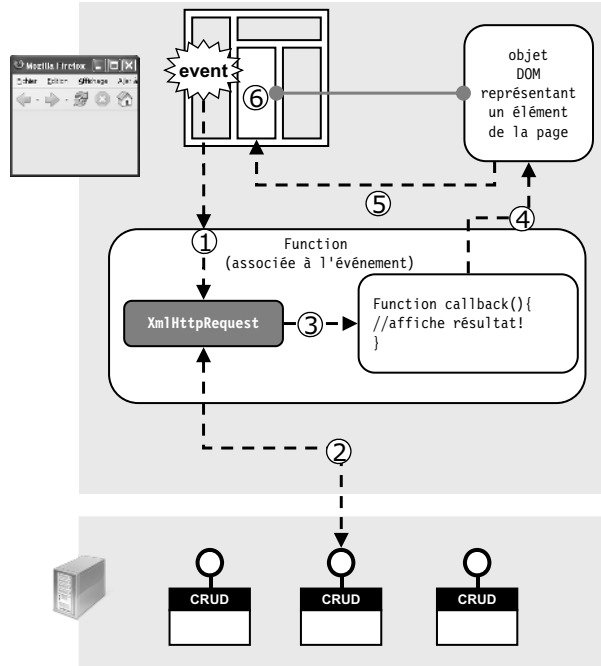



**Figure 18.4** – Principe de fonctionnement AJAX

Les principes présentés ici paraissent simples – et ils le sont. Cependant, lorsqu'il s'agit de développer des comportements plus sophistiqués (champ auto complétant…), les choses se compliquent nettement. D'où l'apparition de bibliothèques de composants AJAX, prêtes à l'emploi (ou presque…), sous forme de tag HTML, telles que DOJO, PROTOTYPE, SCRIPTACULOUS, DWR, et plus récemment JQUERY, EXT…

Chaque composant d'une telle bibliothèque encapsule le principe décrit ci-dessus : il contient les fonctions JavaScript nécessaires (dont la fameuse fonction callback), et il est (souvent) associé à un tag permettant une insertion aisée dans une page HTML.

### 18.3.2 AJAX & SOA : les limites

Une application AJAX conduit donc naturellement à envisager de connecter directement le navigateur (plus exactement un composant AJAX) à un service CRUD distant. On en revient donc à une architecture qui ressemble furieusement à une architecture de type client/serveur, avec un client plus ou moins lourd.

Sous couvert de revenir techniquement aux fondamentaux du Web via HTML et JavaScript, on peut donc légitimement se demander si, au contraire, AJAX ne s'en éloigne pas sur le plan architectural.

Le Web 1.0 a en effet permis de (re)centraliser sur des serveurs les applications interactives, via les architectures JEE par exemple, et les frameworks associés. Ceci a permis de résoudre un certain nombre de soucis endémiques du client/serveur, que ce soit en termes de performance des échanges client/serveur, de montée en charge, ou de déploiement d'application. Or, le Web 2.0 disloque ce type d'architecture, en localisant une partie des traitements applicatifs côté navigateur Web, c'est-à-dire côté client.

Par ailleurs, les services CRUD invoqués via AJAX seront d'une granularité plus ou moins faible. De plus, l'accès à ce type de service sera fréquent, puisque le nombre de clients sera élevé. En bref, c'est à peu près tout ce qui a été dénoncé comme des pratiques à risque dans les chapitres précédents, notamment sur le plan des performances.

On remarquera enfin que JavaScript a une réputation sulfureuse de langage « non industriel », contrairement à Java, C#, ou même PHP. Il est vrai que pendant longtemps JavaScript a souffert d'un manque d'outils (débogueur, outils de tests, éditeur de code évolué…). La situation a évolué, mais reste moins satisfaisante que pour d'autres outils. De plus, obliger le développeur à maîtriser JavaScript, c'est l'obliger à être bilingue, puisqu'il devra également connaître Java ou C# ou… (voire trilingue si on y ajoute XML ou une variante).

On voit donc qu'AJAX pose des problèmes architecturaux indéniables dans le cadre d'une SOA. Mais les qualités ergonomiques des applications AJAX sont à elles seules un argument puissant en faveur de cette approche, AJAX devrait s'inscrire durablement dans le paysage des applications y compris de gestion. Il est donc nécessaire

de gommer les inconvénients indéniables que l'on vient de décrire. Le paragraphe qui suit liste les réponses apportées par la communauté du Web 2.0.

### 18.3.3 AJAX & SOA : éléments de solution

#### *Performance des services CRUD*

Pour améliorer l'accès à ces services, il est possible de bénéficier de la mise en place d'un cache d'information.

Dans l'exemple du Mashup « fil rouge », la mise en place d'un cache pour les étapes 1 et 2 peut s'avérer très intéressante : dans la mesure où les informations sont stables (on n'ajoute pas fréquemment un nouveau département français ou une nouvelle commune), l'efficacité du cache sera maximale.

#### *Architecture des Mashups*

Le Web 2.0, on l'a vu, conduit subrepticement à répartir l'application composite entre le client (avec les composants AJAX) et le serveur, qui doit au minimum gérer l'affichage et la navigation entre les pages HTML (devenues moins nombreuses, puisque plus riches).

Ceci a conduit à « mélanger » les frameworks classiques, tels que STRUTS ou les implémentations JSF, avec des bibliothèques de composants AJAX. L'objectif est de permettre au développeur de continuer à utiliser ses outils préférés, associés à un langage « industriel ». Il s'agit également de lui éviter au maximum de toucher à JavaScript.

Mais ceci conduit à une architecture complexe à construire, à comprendre (pour les développeurs) et à maintenir. La complexité provient d'abord du fait que l'application MVC est séparée en deux : la Vue est en grande partie gérée côté client, les Contrôleurs sont écartelés entre le client et le serveur, le Modèle reste côté serveur, mais est partiellement répliqué sur le client (via des objets JavaScript).

On remarquera que la programmation cliente devient événementielle, alors que la programmation serveur reste une gestion de navigation entre pages (en ce qui concerne STRUTS et équivalents). En ce qui concerne les JSF, la coexistence de deux architectures événementielles, sur le client et sur le serveur, rend l'intégration AJAX ↔ JSF complexe.

En particulier, du fait de cette séparation client/serveur et de la différence d'approche événementielle/navigationnel, quelques questions délicates se posent. Tout d'abord, que devient la gestion de l'état de l'application ? Or, bien gérer l'état de l'application est important puisque cela conditionne son comportement. Doit-on distinguer entre un « macro état » (où en est-on dans la navigation entre pages ?) et un « micro état » (dans quel état se trouve la page, composée de composants AJAX ?) ?

Ensuite, que devient la gestion de contexte, dans le cas d'applications composites associées à des sessions de travail « longues » (cf. § 10.2) ? Où positionner le contexte, qui agit comme un container d'objets métier : sur le client ? Sur le serveur ?

Face à ce désordre, deux approches s'opposent : la première s'appelle « tout sur le client » et la seconde « tout sur le serveur ».

### *Approche « tout sur le client »*

La première approche consiste à considérer que tout se joue côté client, et qu'il est inutile de s'encombrer d'un framework MVC côté serveur. L'application cliente dialogue directement avec des services, via XML ou JSON.

Deux avantages peuvent être accordés à cette approche. D'une part, elle simplifie l'architecture et elle rend effectivement le client indépendant du ou des langages d'implémentation des services; d'autre part, le prototypage d'application est sans doute plus simple.

Elle a trois inconvénients majeurs.

Tout d'abord, cette approche suppose que les services sauront retourner les informations nécessaires dans le bon formalisme. Or, cette hypothèse n'est pas toujours vérifiée au sein du SI : les services qui existent déjà sous forme de WS-* ou d'EJB, ou les services reposant sur des données mainframes, devront être modifiés pour retourner du JSON, par exemple. À moins de développer une hiérarchie de services, avec des services CRUD « encapsulés » par des services applicatifs chargés du travail de traduction de format : cette approche réaliste n'est pourtant pas celle actuellement proposée par la majorité des tenants du Web 2.0.

Le deuxième inconvénient de cette approche est qu'elle suppose que les services accédés par l'application cliente sont tous hébergés par le serveur d'où provient la page HTML hébergeant cette application cliente. Cette contrainte basique du Web peut être contournée, mais au prix d'une complexification du client.

Le troisième inconvénient de cette approche « tout sur le client AJAX » est qu'elle semble considérer que le modèle MVC est mort, et que l'on peut donc s'en passer.

Les critiques de MVC oublient que cette approche a trouvé son origine dans la programmation événementielle de clients très lourds (Windows, Smalltalk…) puis s'est adaptée à la programmation navigationnelle orientée clients très légers. C'est donc qu'elle dépasse ces clivages, parce qu'elle apporte une garantie de réaliser des applications modulaires, maintenables et adaptées aux besoins.

La conclusion sera donc simple : tant qu'il n'existera pas de véritables « framework JavaScript MVC », il sera déconseillé voire improductif d'utiliser cette approche pour développer des applications de gestion stratégiques. Et même pour développer des sites Web, on se trouvera bien de prendre les précautions nécessaires pour éviter le bricolage.

### *RIA ou RDA ?*

Cette approche « tout sur le client » est clairement un retour au client/serveur à base de client lourd, même si la présence du navigateur fait croire à une approche client « léger ».

On parle d'ailleurs d'application riche (RIA pour *Rich Internet Application*), et non plus légère. Si on s'intéresse à cette approche, autant alors s'intéresser à des technologies matures telles que FLEX d'Adobe, qui de plus bénéficie de la présence de bons frameworks MVC tels que PureMVC. Et pourquoi alors s'embarrasser du navigateur, qui impose quelques contraintes fortes ? C'est précisément la démarche d'ADOBE avec son offre AIR[1], qui peut s'interpréter comme « AIR = FLEX sans le navigateur Web ». Dans ce cadre, la distinction RIA versus RDA (*Rich Desktop Application*) n'existe plus réellement.

### *Approche « tout sur le serveur »*

La seconde approche, « tout sur le serveur », consiste à proposer au développeur un framework lui permettant de développer une application événementielle en Java, côté serveur. Mais au lieu de s'exécuter effectivement du côté serveur, l'environnement associé au framework contient un compilateur qui va générer à partir du code Java, le code JavaScript embarqué dans les pages HTML envoyées au navigateur. L'approche « tout sur le serveur » est donc une appellation un peu trompeuse, puisque l'application s'exécute en partie côté client. Cette approche est représentée par le framework GWT (Google Web Toolkit).

L'objectif est donc de concilier les avantages d'AJAX (bonne ergonomie de l'application, programmation événementielle) tout en bénéficiant d'une approche plus industrielle (la programmation s'effectuant en Java, GWT masque complètement au développeur le JavaScript et la manipulation de composants AJAX).

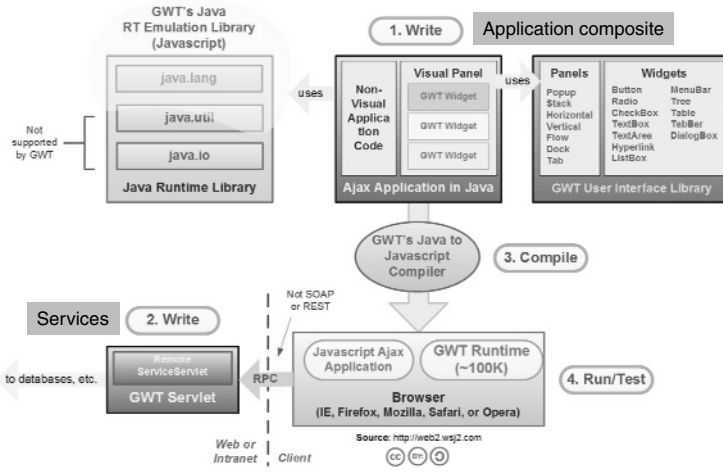



**Figure 18.5** – La mise en œuvre de GWT (source : http://web2.wsj.com)

L'efficacité de cette approche en termes de performance dépend de la qualité du compilateur et de l'architecture sous-jacente. L'intérêt indéniable de cette approche

1. *Adobe Integrated Runtime*.

doit donc être tempéré par la nécessité de contrôler les performances de l'application produite. GWT suscite beaucoup d'intérêt, car il pourrait réconcilier les tenants du Web 2.0 d'une part, et les nostalgiques de la programmation événementielle des clients lourds et de sa puissance expressive. L'essor du poids lourd Google laisse espérer un avenir intéressant à ce framework.

# 18.4 CONSTRUIRE ET INTÉGRER LES MASHUPS

## 18.4.1 Construire les Mashups

Développer un Mashup reste encore une affaire de techniciens. Cependant, certains acteurs importants du Web, et non des moindres, essaient de promouvoir des éditeurs de modélisation graphique et de génération de Mashups. On citera Yahoo avec Yahoo Pipes, IBM avec QEDWiki, Microsoft avec Popfly…

L'idée de départ est la suivante : on remarquera qu'un Mashup, tel que celui décrit dans l'exemple du tableau 18.1, est en fait une sorte de pipe (à la Unix) : le Mashup reçoit un premier flux d'information (la commune), utilise ce flux pour récupérer un deuxième flux (les adresses) et enfin injecte ce deuxième flux dans une carte.

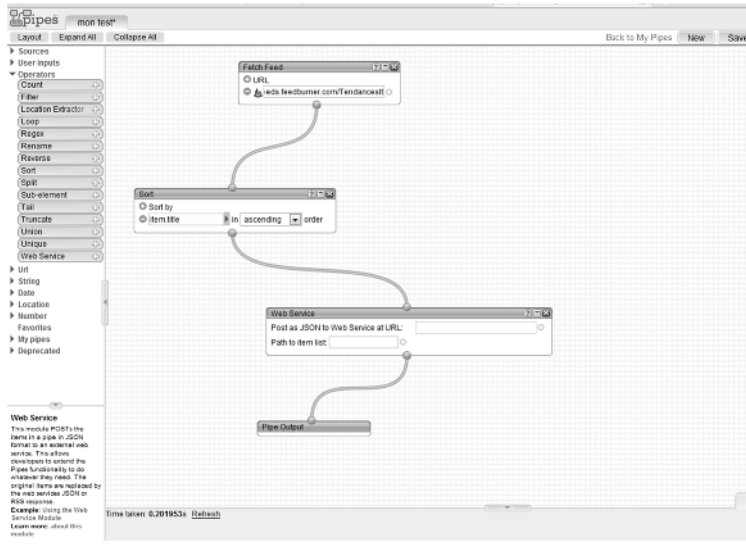



**Figure 18.6** – La création de Mashup avec Yahoo ! Pipes®

Un éditeur de Mashups tel que Yahoo ! Pipes (cf. figure 18.6) fonctionnera donc – idéalement – de la façon suivante :

- le développeur de Mashups sélectionne un ou plusieurs flux d'information « d'entrée », en précisant le type de flux (flux RSS, flux WS-*, flux JSON, etc.);
- le développeur applique à ce ou ces flux des opérations logiques : sélection d'une information particulière dans le flux (le code postal par exemple), tri ou filtre des informations, sélection ou au contraire agrégation d'information, etc. : l'objectif étant d'obtenir un flux d'information « résultat »;
- le développeur sélectionne un objet graphique pour visualiser ce flux « résultat »;
- enfin, le développeur via l'outil donne un nom au Mashup et le transforme en un widget que l'on peut afficher dans un agrégateur de flux (cf. le paragraphe suivant).

On remarquera au passage qu'un éditeur de Mashups digne de ce nom est accessible **via le web** : ce n'est pas un client lourd à la ECLIPSE ou Visual Studio.

Pour que cette approche ait un quelconque intérêt dans le cadre d'une SOA orientée SI, l'éditeur de Mashups doit satisfaire quelques critères fondamentaux.

L'outil doit en priorité permettre de travailler sur des flux d'informations structurées, et pas uniquement des flux d'information de type « news » issus de blog, de wiki, etc. Ces flux structurés proviendront des bases de données SQL, ou de Web Services décrits en WSDL.

Les objets graphiques ne peuvent pas tous être des cartes à la Google Maps : l'éditeur de Mashups doit donc permettre de construire des objets graphiques d'affichage de résultat, tels que des formulaires (affichage de données issues des bases de données) ou des graphiques (affichage de chiffres, statistiques, etc.).

Enfin, il doit être simple d'utilisation. Le nœud du problème est là : il faudra (pour l'instant ?) choisir entre simplicité et richesse des informations traitées.

Du plus simple (Yahoo Pipes !, Open Kapow, Dapper) aux plus sophistiqués (Jackbe[1], Datamashups[2], BEA Aqualogic Pages[3]), les outils ne manquent pas : il conviendra donc d'ajouter aux critères énumérés ci-dessus, le critère industriel de pérennité, bien que ce critère comporte toujours une part de subjectivité.

Certains critiquent cette approche « Mashups » en trouvant ce type d'outil trop simple pour les développeurs d'entreprise, et trop complexe pour les utilisateurs « finaux ». Cependant, la prolifération des macros Excel ou VB dans les entreprises devrait convaincre que beaucoup d'utilisateurs finaux sont aptes à utiliser un outil de développement pourvu que celui-ci soit adapté à leur besoin. Et la pression qui pèse sur les épaules des développeurs les rend ouverts aux outils de productivité même s'ils ne sont pas parfaits.

1. http://jackbe.com/
2. http://datamashups.com/
3. http://www.bea.com/enterprise/pages

En conclusion, on conseillera donc d'évaluer cette approche sur des besoins SI certes simples, mais réels : on pense par exemple à l'encapsulation d'un moteur de recherche, à l'utilisation de cartes géographiques, ou à l'affichage de données de type « tableau de bord décisionnel synthétique ».

### 18.4.2 Intégrer les Mashups

Certains sites d'agrégation destinés au grand public permettent d'intégrer très facilement des Mashups sous la forme de pages composites. Les plus connus sont Netvibes ou iGoogle. Ces sites offrent une bibliothèque de Mashups sur étagères, généralement intitulées « Widgets », que l'on positionne dans la page par simple drag and drop.

Cette approche Web 2.0 pour la construction de portails personnalisés connaît un tel engouement qu'elle influence les offres de portails des grands éditeurs (Microsoft, BEA, IBM, etc.).

De plus, les acteurs du Web 2.0 ont normalisé le format des Widgets sous la norme UWA (*Universal Widget API*). Cette norme connaît, elle aussi, un fort engouement et est en cours d'implémentation par les grands éditeurs.


**Figure 18.7** – La composition de Widgets avec Netvibes®

# 19

# REST et Web Services

## Objectif

Ce chapitre se propose de présenter REST et d'analyser cette solution en comparaison avec une approche Web Service.

La complexité est un reproche souvent fait aux technologies associées aux solutions SOA. Ce n'est pas complètement injustifié, mais le présent ouvrage prévient le lecteur : si le métier n'a pas d'enjeux d'ouverture d'entreprise, de besoin de structuration du SI, ni de pilotage par les processus, alors cette complexité ne sert personne. En revanche, s'il faut orchestrer le SI, le structurer et le préparer à une ouverture maîtrisée, alors il faut s'attendre à payer l'entrée dans une démarche SOA.

La vraie complexité est dans l'impact sur l'organisation, mais certaines communautés incriminent la galaxie des normes WS-* et les fameux Web Services, derniers représentants du vice de complexité en date. Cette perception est largement due à une utilisation naïve de ces technologies : sans typologie de service, la granularité fait faire systématiquement les pires choix de performances, surtout si on considère l'anti-pattern SOA=Web Service, dénoncé plusieurs fois dans cet ouvrage.

Reste que la mise en œuvre des technologies au service d'une SOA comprend des complexités à la hauteur des enjeux métiers associés. En revanche, il existe une riche gamme de réponses technologiques, plus ou moins complexes, qui traitent telle ou telle partie du cahier des charges techniques SOA. Parmi ces technologies, on trouve deux grandes familles a priori opposées. D'un côté les Web Services et les normes WS-*, parangon de la complexité et de la richesse, et de l'autre côté, l'architecture REST et sa philosophie de la simplicité.

Le débat REST contre les solutions WS-* est-il un simple débat technique ? A priori non. Ce chapitre décrypte les deux aspects fondateurs des divergences de ces approches. Le premier aspect est effectivement technique : REST choisit délibérément des directions opposées à des principes de SOAP, avec des succès et des limites qu'on découvrira ci-après. L'autre aspect concerne les modèles architecturaux représentant les interactions entre systèmes : les solutions WS-* sont guidées par « les verbes » (les opérations offertes par les services) alors que le monde REST voit les systèmes d'information comme un ensemble de « noms », ou de choses (c'est-à-dire les ressources informationnelles de l'entreprise, vues comme des entités navigables). Avant de présenter ces deux aspects, le chapitre propose une rapide introduction aux principes de REST.

## 19.1 REST : LES PRINCIPES

Qu'est-ce que REST ? Il s'agit d'un modèle d'architecture proche des origines du Web.

Le débat REST versus Web Service a rempli blogs, forums, magasines en ligne, ou livres blancs ces deux dernières années. REST est le tenant de la simplicité utile face à l'usine tentaculaire des Web Service et des normes WS-* associées. Au premier abord, il est vrai que REST est un exemple rutilant de simplicité, en expliquant que l'infrastructure Web existante est un socle parfaitement suffisant pour faire (presque) tout ce qui est nécessaire à une SOA.

REST signifie *Representationnal State Transfert* : c'est un concept formalisé en 2000 par Roy Fielding. Ce style d'architecture est censé représenter l'essence d'une architecture Web bien fondée et efficace : un réseau de ressources (vue comme une machine à états virtuelle) au travers desquelles l'utilisateur avance dans son application en utilisant des liens (transition d'état) qui lui donnent accès à d'autres ressources (état de l'application) qui lui sont transmises et « rendues » pour son utilisation. Il s'agit du style d'architecture aujourd'hui communément utilisé dans une infrastructure Web.

Ce style prône la standardisation des services pour garantir l'interopérabilité, une approche assez sensée en soi. Pour REST, le monde est un ensemble de ressources, accessibles par quatre services uniquement, représentés par les méthodes HTTP: GET, PUT, POST, DELETE. Tout est ensuite affaire d'identification de ces ressources, via le concept d'URI. Depuis le milieu des années 1990, nous avons vu éclore des solutions d'envergure basées sur cette architecture, et en premier lieu le Web lui-même (au sens de réseau de sites). L'idée est de reconnaître dans cette technologie un support pour implémenter une SOA au sens de cet ouvrage.

Supposons que l'application Fil Rouge gère des contrats d'abonnement dans un contexte Web 2.0. L'accès à un contrat, par exemple le contrat de Michel Chombier dans la figure 19.1, place l'application dans un état donné. Cet état est associé à la visualisation d'une page HTML (page affichant les informations du contrat) : on dit

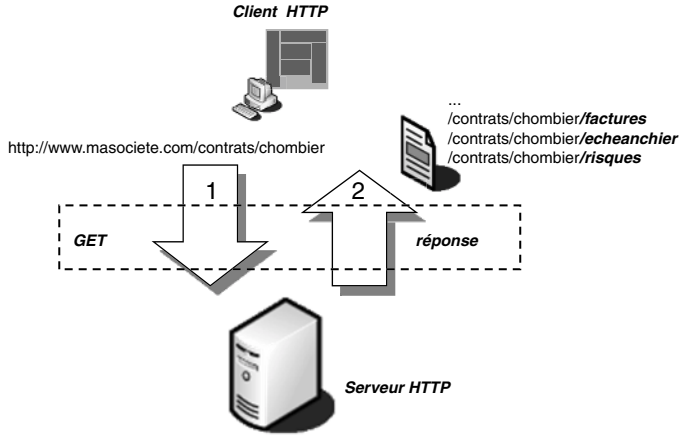



**Figure 19.1** – L'invocation de services avec REST

que la page représente la ressource (au sens ensemble d'informations élémentaires) associée à l'état.

Si l'utilisateur utilise un hyperlien affiché dans la représentation du contrat, c'est-à-dire dans la page HTML associée au contrat, il quitte l'état associé à la représentation de la ressource « contrat » pour accéder à un autre état, associé à une autre ressource, par exemple la « liste des dernières factures ».

Autrement dit, la navigation typique du Web est assimilée à des transferts d'un état à un autre état, chaque état étant lié à la représentation (HTML, XHTML, etc.) d'une ressource informationnelle. D'où le nom : REST = *REpresentational State Transfer*.

REST n'est pas un standard, mais un style d'architecture.

Le Web est un bel exemple de système à base de REST – c'est même l'archétype du style REST ! La plupart des services existants, dont certains de longue date (depuis 1995 environ), commande de livre, dictionnaires en ligne, services de traduction, etc. sont des Services REST. Nous sommes tous les monsieur Jourdain du Web ayant de grandes chances d'avoir déjà utilisé REST sans le savoir.

Ceci explique que toute implémentation informatique se voulant conforme à REST repose sur les standards du Web et certains standards documentaires classiques : HTTP (pour le protocole et les services élémentaires d'accès aux ressources), URL (pour la localisation des ressources), XML/HTML/XHTML/GIF/JPEG/, etc. (pour la Représentations de Resource), text/XML, text/html, image/gif, image/jpeg, etc. (pour les Types de Ressource, Types MIME).

### *Illustration avec un système de gestion de commande*

Si on prend l'exemple d'un fabriquant de produit (par exemple un fournisseur de compteurs) voulant ouvrir son SI en s'appuyant sur REST, il déploiera un ensemble de services ouverts sur le Web pour ses nombreux clients :

- la liste de ses produits : *get a list of parts*;

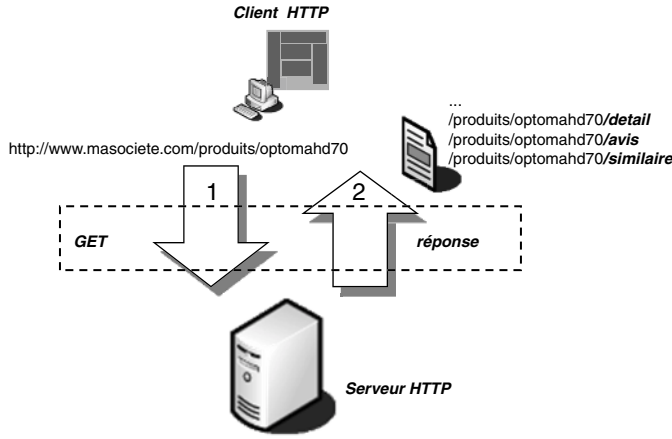



**Figure 19.2** – Système de gestion de commande avec REST

- le détail d'un produit : *get detailed information about a particular part*;
- la mise à jour d'un produit;
- la commande d'un produit : *submit a Purchase Order* – Cette requête s'appuiera sur un POST;
- la suppression d'une commande.

La figure 19.3 montre (de façon simplifiée) comment ces services seront implémentés avec le style REST, et leur équivalent en Web Services.

| Style REST | Style WS-* |
|---|---|
| `GET(URI)` | `lireCatalogueProduits()` |
| `PUT(URI, resource)` | `mettreAJourProduit(produit)` |
| `POST(URI, resource)` | `creerCommande(commande)` |
| `DELETE(URI)` | `supprimerCommande(IdCommande)` |

**Figure 19.3** – Style REST versus style WS-*

### *Accès en lecture*

Dans une implémentation REST, il n'y a pas vraiment de services d'accès à l'information : c'est la ressource elle-même qui est accessible par un simple GET. Si on

prend l'exemple de la lecture de la liste des produits, cette lecture s'effectuera en invoquant l'URL suivante :

```
http://www.masociete.com/produits
```

Cette invocation se traduit classiquement dans le monde WEB par une requête HTTP GET.

La réponse à cette invocation est une ressource dont le type repose sur un standard du Web, par exemple, une page HTML ou un document XML. Si besoin, il est possible de paramétrer les services de « lecture » pour avoir tel ou tel type de sortie, par exemple avec un paramètre de requête HTTP :

```
http://www.masociete.com/produits?format=xml
```

Si on veut utiliser REST en lieu et place des WS-* pour mettre en place des services de lecture d'information, la réponse sera encodée en XML. L'exemple ci-dessous est le document XML « catalogue des produits ».

```
//liste des produits, en XML, avec des liens xlinks (norme) en plus des ids
< ?xml version="1.0" ?>
<cat :catalogue xmlns :cat=http://www.mondepot.com

     xmlns :xlink=http://www.w3.org/1999/xlink

     xmlns :xsi=http://www.w3.org/2001/XMLSchema-instance

     xsi :schemaLocation="http://www.mondepot.com http://www.mondepot.com/
catalogue.xsd">

     <cat :produit id="00042" xlink :href="http://www.mondepot.com/produits/
00042"/>

     <cat :produit id="00108" xlink :href="http://www.mondepot.com/produits/
00108"/>

     <cat :produit id="00666" xlink :href="http://www.mondepot.com/produits/
00666"/>

     <cat :produit id="00747" xlink :href="http://www.mondepot.com/produits/
00747"/>

</cat :catalogue>
```

On remarquera que le document XML « catalogue » contient, pour chaque produit, d'une part l'identifiant de ce produit, mais également un hyperlien vers la description (la « représentation » si on veut être puriste) dudit produit. En XML, cet hyperlien est signalé par l'attribut `xlink`.

L'application « catalogue Web » effectuera la lecture d'un produit en invoquant cet hyperlien, c'est-à-dire l'URL présente dans le document précédent :

```
http://www.masociete.com/produits/00042?format=xml
//détail en GET ! ! !
```

La réponse sera un document XML comme suit :

```
< ?xml version="1.0" ?>
<pro:produit xmlns :cat=http://www.mondepot.com
     xmlns :xlink=http://www.w3.org/1999/xlink
     xmlns :xsi=http://www.w3.org/2001/XMLSchema-instance
     xsi :schemaLocation="http://www.mondepot.com http://www.mondepot.com/
produit.xsd">

     <produit-id>00042</produit-id>
     <nom>produit-X</nom>
     <description>Le produit qu'il vous faut</description>
     <tarif>le tarif de base</tarif>
     <documentation  xlink :href="http://www.mondepot.com/produits/00747/
documentation"/>
     <categorie> une catégorie </categorie>
     <stock> 12 </stock>
</pro :produit>
```

On note la cohérence dans les ressources appelées et leur représentation : des liens successifs, exprimés via `xlink`, sont utilisés soit pour donner accès à un ensemble de résultats, soit pour donner accès au détail des représentations offertes à l'utilisateur. Cette démarche de découverte « dynamique » est particulièrement adaptée à des échanges internaute ↔ informations Web.

On notera aussi qu'on n'expose pas des ressources physiques, mais toujours des ressources logiques, via le mécanisme d'URL. L'implémentation de l'accès à ces ressources offre donc un niveau de découplage suffisant pour permettre de tout changer côté serveur. Et ce, même si l'url change, car on peut utiliser par exemple la réponse HTTP `302 moved` pour signaler à l'application cliente ce changement d'URL.

Ce dernier exemple montre l'intérêt de l'approche REST : en s'appuyant pleinement sur HTTP (non seulement sur les `request` mais aussi sur les `response`), il est possible de faire face à des situations d'évolution du SI pas forcément triviales.

### *Recherches complexes*

Reste maintenant à aller au-delà de ces premiers exemples relativement simples : quid d'une requête du monde réel, avec des vrais filtres liés à un acte métier ?

Comment par exemple représenter la requête « donnez-moi les franchises supérieures à 200 euros associées aux objets assurés de type "véranda" dans le contrat "habitation" de Mr Chombier ». Est-ce représentable avec la fameuse technique de l'URL simple et sympathique ? La réponse est oui… jusqu'à un certain niveau.

```
http://www.masociete.com/contrat/chombier/habitation/risque/veranda/franchises
?montantmin=200
```

ou bien :

```
http://www.masociete.com/contrat/chombier/habitation/franchises
?risque=veranda&montantmin=200
```

Les exemples précédents sont des implémentations envisageables.

Mais la philosophie REST nous demande de nous préparer aux requêtes d'une certaine complexité en proposant un formulaire d'échange, à remplir par l'appelant. Les réponses seront encore une fois une représentation contenant des « liens » vers les détails à utiliser. De cette manière, les URL compliquées qui demandent une connaissance de la manière dont nous nous organisons en offrant nos services ne sont pas générées par les clients, mais par le fournisseur…

***Création d'un objet métier***

Comment créer une nouvelle ressource, par exemple un nouvel objet métier « commande », puisque par définition, cet objet, avant sa création, n'existe pas et n'a donc pas d'URL ?

La solution REST proposée est en général de mettre à disposition une URL spécifique de la création d'un objet métier. Cette URL est invoquée via une requête POST, qui transportera la description de l'objet à créer. Cette description sera un document XML, produit par un internaute via un formulaire HTML (on revient à la soumission classique d'un formulaire Web). Ce document peut aussi être produit par un SI externe.

En reprenant l'exemple du fabriquant de produit, le service REST de commande d'un produit se traduira par la publication d'une URL de soumission d'une commande. L'application cliente doit créer (via un formulaire ou un programme) une instance XML conforme à la description d'une commande.

Le service de commande devra répondre, après création de la commande, par l'exposition d'une URL d'accès à la commande créée. Ceci permettra ensuite au client de retrouver sa commande plus tard. La commande est alors devenue une ressource partagée identifiée par URI.

« Conforme à la description d'une commande », a-t-on dit : mais comment garantir que l'application cliente va émettre une commande « conforme » ? Il faut pour cela que la description XML de la commande soit conforme à un format décidé à l'avance, on parle de « format pivot » : dans ce contexte, ce sera un schéma XSD décrivant une commande type. C'est la contrainte d'interopérabilité classique permettant à un client et à un service de se comprendre mutuellement.

Dans le monde des WS-*, la norme WSDL est la réponse à cette contrainte : elle permet de décrire et de publier ces schémas de conformité.

Mais dans le monde REST ? Ce problème, après avoir été nié, a fini par être reconnu comme un vrai problème. Un débat s'est même engagé sur l'opportunité d'utiliser WSDL pour cela. On parle aussi de WRDL (*Web Ressource Description Language*), car des absolutistes de REST trouvent WSDL trop WS-* centric, et souhaitent une approche plus simple et (surtout ?) dédiée.

Le schéma décrivant une commande serait alors rendu public dans le contrat de service via WSDL, ou WRDL, ou via un simple document schéma XSD hébergé par le site web. L'intérêt d'un langage comme WSDL ou WRDL est de proposer ensuite des serveurs prêts à l'emploi (SOAP/ WSDL ou REST/WRDL) capables de vérifier eux-mêmes la conformité d'une requête au schéma de description. Si on utilise un

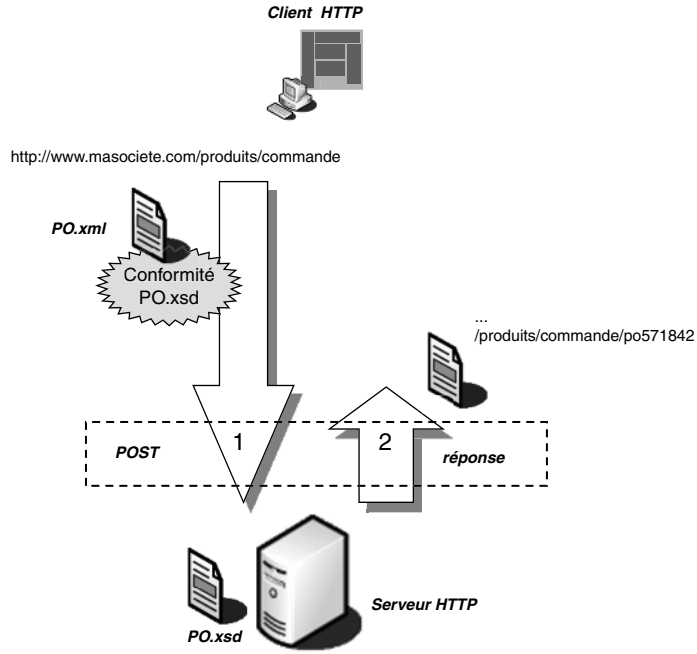



**Figure 19.4** – Utilisation de schéma XML

« simple » schéma WSDL, ce sera au réalisateur du service de création de commande de vérifier en premier lieu que la commande est bien conforme.

L'instance XML de commande est transmise, comme on l'a vu, comme une donnée transportée par une simple requête HTTP POST.

On retrouve ici des caractéristiques bien connues de l'univers WS-* et des messages basés sur SOAP. Les précautions à prendre sont donc les mêmes que dans les appels distants de type SOAP : trouver le moyen de garantir les paramètres d'appel et un nommage ou une reconnaissance de l'opération appelée (à comparer avec le Document/Wrapped mentionné lors du chapitre 5).

La différence entre REST et WS-* est donc beaucoup plus ténue que les évangélistes du WEB 2.0 veulent bien le reconnaître, en tout cas sur cette gamme de services « réalistes » que sont l'appel d'une opération de recherche « complexe » ou de création/affectation, c'est-à-dire les services dont la sémantique dépasse le simple nom de la ressource.

### *Caractéristiques*

En conclusion, les caractéristiques d'un réseau basé sur REST sont les suivantes :

- Tout le système est composé uniquement de ressources, et chaque ressource est nommée par une URI.
- L'interaction avec le système est de type client-serveur, synchrone, par système de « pull ».

- L'interface d'invocation est standardisée et uniforme pour toutes les ressources (i.e., HTTP GET, POST, PUT, DELETE).
- L'invocation d'une ressource est sans état : chaque requête cliente a toute l'information nécessaire pour la comprendre et ne repose pas sur un contexte stocké sur le serveur.
- Toute réponse à une invocation doit être capable d'être identifiée comme « cachable » ou « non-cachable » afin de permettre de maîtriser la performance.
- Les représentations (clientes) des ressources sont liées par des URL pour permettre aux clients de progresser d'un état à un autre.

# 19.2 COMPARAISON AVEC L'APPROCHE SOAP

Comment traiter ces mêmes besoins avec SOAP ? Les chapitres précédents ont montré comment utiliser des Web Services, des schémas et des messages SOAP pour exposer tel ou tel service. Comme on a commencé à le voir, il y a quelques différences avec l'approche REST.

Les envois de message SOAP sont, **en mode Web/HTTP uniquement,** portés par **des requêtes de type POST** : ils n'utilisent pas d'URL spécifique autre que celle du bus de service ou du gestionnaire de message SOAP offert par le serveur. L'information de routage est masquée dans l'enveloppe SOAP. Il y a autant de sémantiques telles que `lireCatalogueProduits()`, `afficherDetailProduit(IdProduit)`, ou bien encore `commanderProduit(commande)`, que de services exposés. La volonté est d'exposer des contrats dirigés par le **verbe**.

Le standard SOAP offre un système de contrôleur unifié. Cette politique de masquage des URL classiques nous fait perdre par avance la simplicité du GET : **les serveurs de cache et de proxy ne pourront pas nous aider ici**.

L'abstraction qui veut servir des standards d'interopérabilité se fait donc au détriment des services d'une infrastructure Web existante en termes de cache, sécurité par filtrage d'url, etc. **Elle demande l'installation et la médiation d'une nouvelle gamme de logiciel** : **les bus de service/ESB/Listener, etc.** nécessaires pour permettre l'ouverture de l'enveloppe SOAP et accéder aux différentes informations de routage, sécurité, etc. De plus, à chaque réponse, il n'est pas nécessaire au sens de SOAP et des Web Services de proposer des accès de type url pour avancer dans la consommation des services. Les Web Services partent du principe que **le client se renseigne sur le contrat du service**, ses schémas et la sémantique de ses interfaces. Le client doit « savoir » comment avancer dans le détail.

## 19.2.1 Constat technique : REST remplit son contrat

Du point de vue du défenseur de REST, la sentence est sans appel. REST résout tout et les Web Services nous font perdre du temps et de l'argent.

Les intermédiaires et médiateurs du Web ont besoin de requêtes claires : proxy, filtrages d'url, cache, etc. Les changements d'états et les options de continuités sont clairs en REST et l'utilisation de `xlink` permet même leur accès à des acteurs automatiques.

L'imposition du POST par SOAP déroute tous les médiateurs classiques du Web. Dans le monde WS/SOAP, cette connaissance doit être appréhendée par le client.

L'interface générique à la REST a tout pour séduire, notamment les applications à développement rapide de style Mashups, qui demandent un accès normalisé au SI. GET/POST/PUT/DELETE sont encore un avocat de la simplicité de REST dans le domaine Web.

WSDL et SOAP offrent potentiellement une sémantique spécifique à chaque interface de service déployée. Certains y voient le critère d'interopérabilité de REST… sur le Web. Par analogie, SOAP fait ici preuve d'ingérence technique.

Prenons le cas d'un service courrier : toutes les décisions de routage des courriers se font sur la lecture des informations extérieures à l'enveloppe, REST serait alors analogue à la poste, alors que WS/SOAP nous proposerait de fonctionner en ouvrant tous les courriers.

Une première solution pour le monde des WS-* concerne la mise en place de la norme WS-Transfer. Cette norme veut en effet permettre d'enrichir les échanges SOAP avec des URL et des méthodes standardisées basées sur les méthodes HTTP. La simplicité REST dans le monde WS-* ? Une veille active sur l'avenir de cette norme s'impose.

Une solution plus générale pour SOAP pourrait être une évolution vers le fameux Web sémantique, qui permettrait à des acteurs automatiques de « découvrir » le fonctionnement des messages SOAP, leur sens. Ce n'est pas à l'entreprise aujourd'hui de faire de la veille sur le Web sémantique, et on laissera cette approche technologique faire ses preuves, probablement à moyen/long terme.

### 19.2.2 REST est une solution de bon goût « pour le Web »

Attention : interopérabilité, transitions claires, utilisation maximale des intermédiaires, les caractéristiques alléchantes de REST sont basées sur le Web uniquement ! C'est-à-dire un monde client / serveur strictement synchrone.

Or, les accès asynchrones de MOM, JMS, SMTP et autres offrent des propriétés supplémentaires : asynchronisme, multicast, robustesse, fiabilité, etc. L'approche « message » a sa propre légitimité, que l'approche Web ne remet pas en cause. Le monde WS-* a reconnu cette légitimité, pas le monde REST.

REST versus WS-* est donc dans les faits plutôt une opposition REST vs SOAP. Nous tombons dans un problème technique, principalement sur le fait que les ténors du Web veulent construire sur le Web par essence synchrone et rien d'autre.

Un autre point important : malgré les débats et initiatives autour de WRDL, le monde REST continue de voir les annuaires de services comme des chimères inutiles : UDDI est bien trop compliqué, son protocole d'accès est largement suppléé par un bon fichier WRDL/WSDL et autres schémas derrière un honnête serveur HTTP.

On reconnaîtra que ce constat n'est pas faux : autant WSDL & SOAP (contrairement à ce qui est écrit ici ou là) sont largement utilisés dans le monde du SOA d'entreprise, autant UDDI n'a pas eu le même succès. Cependant, WS-* ou REST, se pose de la même façon le problème du versioning, de la découverte dynamique d'un service, etc.

La seule réponse REST est que les ressources se découvrent par requêtes successives via des contenus RDF/RSS qui guident le client vers la bonne ressource de manière explicite, ce qui veut dire en fournissant des URL au sens REST. La robustesse de la simplicité peut se payer par un nombre croissant d'allers retours. Et RDF n'est pas encore d'un emploi très répandu…

## 19.3 RECOMMANDATIONS

REST reprend l'infrastructure en place en termes de cache/sécurité (premier niveau), accès HTTP et connectivité dynamique sans grande planification. Les problèmes sont l'orthogonalité avec l'existant Web et la philosophie Web uniquement (proxy, cache, etc.).

Dans une architecture strictement Internet (mise en place de site Web, etc.), SOAP pose plusieurs problèmes :

- URI cachée dans l'enveloppe (géré par WS-transfer/adressing);
- objectif/méthode cachée dans l'enveloppe;
- opérations spécifiques (opposé à GET/PUT/DELETE/POST).

REST permet de respecter les fondamentaux techniques de l'architecture Web et d'en bénéficier.

Si l'on souhaite utiliser REST, quelques recommandations s'imposent :

**Bonnes pratiques REST**

Pour les usages Web centric, voici le résumé des bonnes pratiques REST :
1. Toute ressource exposée doit être identifiée par une URI.
2. Préférer les URI logiques aux URI physiques :
• `http://www.masociete.com/gizmos/747` (souhaitable)
• `http://www.masociete.com/gizmos/747.html` (moins souhaitable)
Les URI logiques sont des interfaces de façades pratiques pour modifier l'implémentation sans prévenir le client.
3. En corollaire, utiliser des noms dans les URI, à la place de verbes. Les ressources sont des choses et non des actions. ➢

➢
4. Toute requête GET doit n'avoir aucun effet de bord. Elle doit rester « safe ».
5. Utiliser des liens dans vos réponses aux requêtes ! En connectant votre réponse à d'autres ressources, vous permettez à vos clients d'être autoguidés sur « la suite » des opérations. (En opposition avec « tout est à décider avec les données fournies »).
6. Minimiser l'utilisation de Query String :
• `http://www.masociete.com/gizmo/747` (souhaitable)
• `http://www.masociete.com/gizmo?gizmo-id=747` (moins souhaitable)
L'accès direct au détail de la demande de ressource est plus efficace que d'aller chercher ce qui est demandé dans la query String.
7. Utiliser le « / » dans une URI pour représenter une relation parent/enfant, maître/détail, composite/composant.
8. Fournir les accès graduels aux données aux clients : une représentation devrait fournir des liens pour obtenir « ses détails ».
9. Toujours utiliser GET lorsqu'il s'agit de fournir une représentation. Ne pas utiliser POST.

Avant de généraliser REST dans une SOA d'entreprise, on se trouvera bien, cependant, de réfléchir auparavant aux questions qui sont listées ci-après.

### 19.3.1 Hors du « tout Web », REST perd de sa pertinence

SOAP reprend sa légitimité dans un système de participation « clos », c'est-à-dire où les participants sont prêts à se mettre d'accord en amont, en termes d'optimisation et d'interface métier. Or, selon les chapitres précédents, une SOA riche concerne précisément ces acteurs.

#### *Appel d'opérations*

Les opérations de recherche complexe et de soumission de nouveaux éléments métier sont naturellement traitées dans un POST dans un cadre REST.

Or si l'on observe, comme on l'a fait précédemment, les contraintes d'utilisation des méthodes POST dans cette architecture, on y retrouve tous les principes à respecter pour obtenir un échange stable tel que prôné par les Web Services au sens SOAP !

Les données, qui voyagent dans la partie data de la requête HTTP, sont souvent encapsulée par un format XML. De plus, dans un SI, ces données seront structurées, et liées entre elles via un graphe d'objet. Qu'il y ait une enveloppe SOAP ou non ne change donc pas grand-chose à l'implémentation côté serveur. Ce serveur devra inclure obligatoirement les composants de traitement de ce type de requête : parsers, validateurs, routage par rapport au contenu d'un document, sécurité – tout est retrouvé ici dans la soumission de données, il n'y a pas de différence entre REST et SOAP.

Certains aficionados de REST répondent en prônant ATOM comme format d'échange, voire JSON ou des formats d'échanges adaptés au Web : ces formats sont supposés moins contraignant/consommateurs que du SOAP + schémas classiques.

Mais les normes ATOM sont des normes XML quoi qu'il en soit. De plus, elles ne sont pas encore complètement stabilisées. Enfin, elles sont en but à la concurrence de RSS au sein du monde WEB 2.0, ce qui fait que leur adoption généralisée (condition de leur succès en dehors de ce monde) n'est pas acquise.

Quant à JSON et consort, les formats natifs accessibles en JavaScript, ils sont évidemment très intéressants comme on l'a vu au chapitre précédent. Cependant on élude souvent la difficulté à les produire, et leur utilisation simple se cantonne une fois de plus au Web et à la partie cliente.

En conclusion, REST, comme SOAP, implique la nécessité d'encadrer la description des échanges de données de façon formelle et validée. Les solutions REST existent certes, et sont relativement faciles à utiliser sur les clients qui utilisent nativement ces représentations (JSON); de plus les flux sont plus clairs que les schémas privés car leur description est standardisée (ATOM).

Mais leur pleine utilisation repose sur un fort pouvoir auto descriptif (découverte par l'application du sens des données au fil de l'eau), pouvoir peu utilisé par les applications clientes actuelles. De plus, les problématiques clefs de grappe d'informations et de scenarii de chargement (exposées au chapitre 8, section 8.3.2) ne sont pas prises en compte par ces formats d'échanges : en particulier, il n'y a pas de réponse pour les requêtes qui rapportent des grappes d'objets complexes dont certaines ne sont pas entièrement chargées (chargement par nécessitée). Il n'y a pas de réponse non plus pour les accès paginés (sur une collection de résultats trop importante) – dans un SI d'entreprise, il n'y a pas que des accès séquentiels simples !

ATOM annonce qu'il va apporter des solutions, mais nous sommes au même niveau de visibilité que pour les publicités autour de WS-Transfer ou WS-Enumeration… La gestion des POST demandera côté serveur des infrastructures de gestion de contexte adaptées, correspondant aux structures déjà abordées dans le cas des solutions WS-*.

#### *Contexte asynchrone*

Les technologies WS-* sont d'ores et déjà armées pour les besoins de couplage asynchrone. REST ne propose rien ici, à part un travail à la charge du client. La sentence est sans appel si une interface asynchrone doit être nécessairement exposée aux clients extérieurs. Ici, les médiateurs nouveaux de type ESB ont toute leur place. Les WS-* sont en terrain conquis, ou presque.

### 19.3.2 Le verbe SOA en opposition aux ressources REST

Nous avons vu que les différences technologiques montrent des gammes d'utilisations optimales différentes pour une technologie REST et une technologie de Web Services basée sur SOAP. Mais il y a une approche plus « fondamentale » encore que REST vs SOAP.

En prenant du recul, REST vs WS-* reste d'actualité, ou plutôt REST (accès des informations : orientation « nom de choses ») vs RPC (accès à des traitements :

orientation verbe). On est ici dans la représentation du métier soit par des noms soit par des verbes.

L'ensemble des méthodologies qui permettent de faire émerger des services conformes aux enjeux métiers utilise des représentations à base d'actions, d'opérations au service des processus de l'entreprise. Cet ouvrage a déjà montré qu'il est difficile et nécessaire de faire apparaître une typologie de service pertinente pour bénéficier des apports d'une SOA. L'approche REST semble ici ajouter une complexité entre les besoins métiers et une solution technique : tout service (par exemple les services fonctionnels, cf. chapitre 8) sont-ils modélisables sous forme de ressources REST ? Il faudra donc de toute façon adapter les méthodes d'urbanisation du SI aux approches REST.

D'un côté, REST propose donc une universalité technologique, HTTP et ses verbes choisis associés à une représentation CRUD & représentation plus ou moins standard (ATOM…) pour les ressources de l'entreprise vues comme des données. De l'autre côté, WS-* prône la représentation explicite des données ET des verbes spécifiques aux services exposés. SOAP est alors le premier représentant du middleware d'échange, ignorant même le protocole de transport.

Dans ce contexte technique, les deux approches ont lieu d'exister, peuvent coexister et s'enrichir l'une et l'autre.

La communauté REST réfléchit sur l'utilité d'un WSDL like pour décrire les data échangées et souhaite élaborer WRDL (Ressource), c'est-à-dire la représentation d'un contrat au sens REST. Dès lors que le client doit parcourir une grappe d'objets complexe, les « verbes » vont reprendre le dessus, même si une approche REST se propose de les générer au besoin. Les URL montrées aux paragraphes précédents pour illustrer des requêtes plus complexes que « donne-moi le contrat n° 1080042 » illustrent ce point.

L'approche REST implique de découvrir ces verbes et ressources nommées lors de l'utilisation. Tant que ces utilisations successives d'appels/liens transmis sont adaptées aux besoins tant en termes de contraintes fonctionnelles que de contraintes de performances, REST est effectivement une solution d'implémentation fiable. En revanche, si performance ou richesse d'échange imposent plus de connaissance réciproque, alors les solutions WS-* sont une solution qui dépasse le problème de la technologie pour apporter des solutions adaptées à l'expression même des enjeux métiers.

REST est donc efficace et éligible si :

- les opérations de lecture GET sont sans état et répétables n fois : ce qui implique par exemple qu'il n'y a pas de problématique de verrou en lecture ;
- les opérations de modification et de suppression/archivage sont idempotentes, c'est-à-dire que l'on peut les demander n fois sans introduire d'incohérence : relance suite à une perte de connexion, pour vérification, etc.

Effectivement, au sens HTTP/Web du terme, pour des ressources classiques sur Internet, ces propriétés sont vraies. Si les services ainsi exposés conservent ces pro-

priétés, alors les solutions apportées par REST sont très adaptées. On peut demander n fois un GET sur une ressource et le résultat sera le même. On peut demander n fois l'altération d'une ressource, par PUT ou DELETE, et le serveur Web saura faire le traitement une seule fois et continuer sans effet pour le client (la ressource est modifiée, la ressource n'existe pas…).

Il ne faut pas se voiler la face : quels que soient les problèmes sous-jacents à REST, la simplification apparente par rapport aux WS-* et à WSDL reste attractive. L'approche « code2wsdl » (par rapport à l'approche « wsdl2code », théoriquement mieux adaptée, mais moins simple) permet de générer plus facilement les web services, au risque de générer des services de granularité peu ou pas adaptés. Mais elle ne masque qu'en partie la complexité WS-*.

REST a donc sa place dans les solutions d'implémentations des SOA.

Cependant, REST va devoir se pencher sérieusement sur la représentation des informations manipulées pour bien se positionner dans l'approche Mashups : cette approche repose en effet sur une description formelle des informations. Standardiser les verbes ne suffit pas pour bâtir rapidement des applications simples. ATOM est sans doute un premier élément de réponse, mais le passage à WRDL ou équivalent risque de se révéler nécessaire – dans ce cas, la différence entre REST et WS-* s'estompera nécessairement un peu plus.

## 19.4 COMMENT CHOISIR ENTRE REST ET SOAP ?

En conclusion, WS-*/SOAP et REST sont deux solutions d'implémentations légitimes.

REST se cantonne aux seuls principes d'architecture de la solution informatique alors que WS-* accompagne les MOA et MOE dans des scenarii plus profonds au sens de l'expression d'un SI, notamment en accompagnant la modélisation et la typologie des services et informations métier, et en incitant à réfléchir sur les politiques d'organisation.

La principale différence est le choix de ce qui est offert en standard : REST standardise les verbes et les architectures techniques, alors que WS-* standardise l'expression des échanges et les moyens d'exposer des services spécifiques. Dans de nombreux cas, le premier niveau de standard REST pourra être suffisant. C'est le lieu de prédilection des applications rapides de style MASHUPS, à condition de se cantonner (pour l'instant du moins) aux informations non structurées.

Dans les cas où les interactions avec les SI sont plus complexes (asynchronisme, sécurité, verrouillage des informations…) et que les contrats sont fortement contraints, SOAP trouve toute sa légitimité. C'est-à-dire que la complexité sous-jacente de ces solutions est une réponse à une réelle complexité des enjeux métiers, pas un gadget technologique.

Pour identifier son besoin, il est donc nécessaire de savoir ce que l'on souhaite standardiser : une démarche SOA commence bien avant de se demander si REST ou WS-* sera la colonne vertébrale de l'interopérabilité.

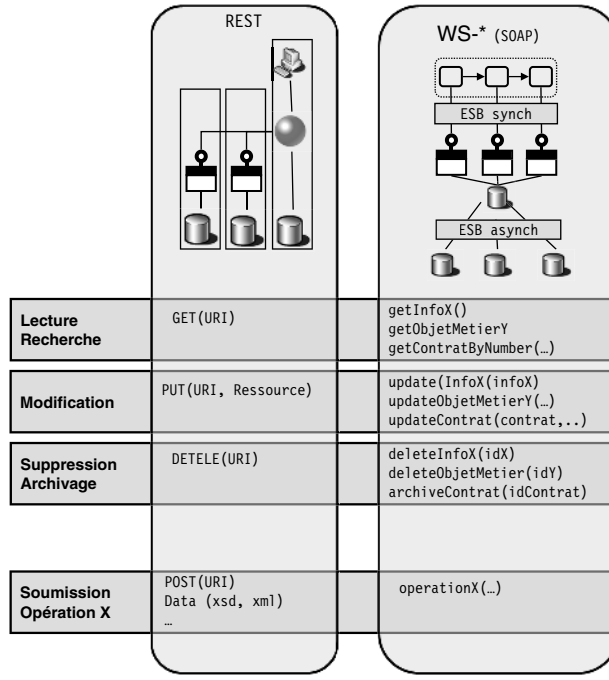



**Figure 19.5** – Synthèse des différences entre REST et SOAP

## En résumé

Le Web 2.0 tend à remplacer SOA comme terme à la mode, dans les gros titres de la blogosphère. Certains tentent d'opposer les deux approches, alors que le Web 2.0 représente dans les faits une nouvelle approche pour construire une SOA. Après en avoir rappelé les principes, la présentation des aspects « applications composites » du Web 2.0 a mis en évidence les avantages et les inconvénients de cette approche pour bâtir une SOA. Avantage en termes de simplicité et d'ergonomie pour l'utilisateur final, inconvénient en termes d'immaturité et de difficulté de maintenance dès lors que l'application dépasse le cadre de quelques pages Web. Il n'est donc pas certain qu'il faille déjà parler de SOA 2.0.

# SEPTIÈME PARTIE

# Décrypter l'offre du marché

Le but de cette septième partie est en premier lieu de caractériser les outils d'infrastructure nécessaires au déploiement des solutions SOA, comme le bus d'entreprise, le registre de services, ou l'orchestrateur, mais aussi les nouveaux outils disponibles pour la direction informatique, principalement la chaîne de fabrication logicielle pour SOA, et la console de suivi unifiée des services et processus.

L'engouement que suscite l'approche SOA conduit l'offre d'outils à devenir pléthorique : une grille de critères de comparaison est donc proposée pour servir d'aide au choix.

Enfin, les acteurs majeurs sont introduits de façon synthétique : l'objectif est de brosser un rapide panorama de l'offre, sans détails inutiles du fait de l'évolution rapide du marché.[1]

1. Les outils ne sont pas évalués dans la grille de critères pour éviter l'obsolescence trop rapide du contenu de cette partie.

# 20

# Caractéristiques de la plate-forme SOA

## Objectif

L'objectif est ici de dresser un panorama des modules potentiellement utiles pour un environnement permettant de construire, d'héberger, d'exécuter et d'administrer les solutions métiers SOA.

De l'ensemble des offres en matière d'infrastructure SOA émerge le concept d'ESB (*Entreprise Service Bus*), mais force est de constater que c'est un concept « attrape-tout » aux contours flous.

Ce chapitre souhaite donc préciser le périmètre de ce qu'est un ESB, en lister les principales fonctions techniques, et présenter les outils complémentaires de l'ESB, tels que le registre des services, l'orchestrateur, le distributeur de requêtes CRUD ou le cache de données. Il introduit de ce fait le concept de plate-forme SOA, encore appelée APS, *Application Platform Suite*.

L'exposé se veut une présentation architecturale, indépendante de tout vendeur particulier. Associé au chapitre suivant, qui récapitule les critères discriminants par module, il constitue ainsi un guide préliminaire permettant à l'équipe chargée de l'infrastructure SOA d'effectuer un choix éclairé d'outils.

## 20.1 L'INFRASTRUCTURE NÉCESSAIRE POUR SOA

L'infrastructure SOA est présentée par la figure 20.1.

Cette infrastructure comprend les composants suivants (les sous-chapitres suivants détaillent leurs fonctions respectives) :

- Le bus d'acheminement des messages d'appels de services, associé à un annuaire de service, souvent baptisé ESB – *Enterprise Software Bus*.
- Le moteur d'orchestration des processus, destiné au BPM (*Business Process Management*).
- Le container de services supportant la norme SCA.
- Les outils permettant d'accéder aux référentiels existants (EII/MDM).
- Les outils de suivi (BAM, SAM).
- Les outils d'administration.
- Le portail composite, regroupant les applications composites et offrant ainsi un point d'accès unique aux utilisateurs des différentes applications interactives et processus métier disponibles.
- La chaîne de fabrication des composants SOA (services et Applications Composites).

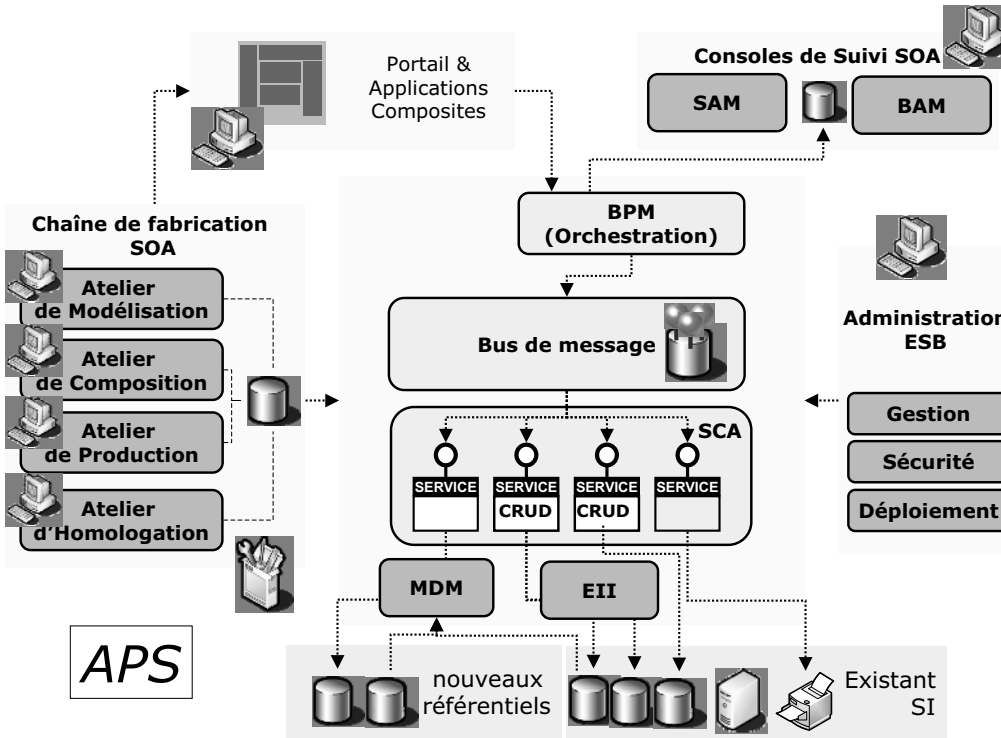



**Figure 20.1** — Les modules d'infrastructure SOA

Que recouvre exactement l'appellation ESB ? Au sens strict du terme, il s'agit d'abord du bus de communication de messages et d'événements métiers : ce composant permet à un consommateur (une application interactive, un service, un moteur de BPM externe) d'accéder à un service. Ce bus est associé aux outils de suivi technique SAM.

Sous la pression marketing des éditeurs, le périmètre de l'ESB s'accompage souvent d'un conteneur de service SCA, d'un moteur de routage et embrasse parfois un moteur de règles métiers, un moteur d'orchestration BPEL accompagnés de consoles d'administration.

Malheureusement, il n'est pas rare de voir certains éditeurs englober sous la dénomination ESB la totalité des composants présentés sur la figure 18.1, y compris le portail composite, l'accès aux référentiels avec des outils MDM/EII, voire certains ateliers de la chaîne de fabrication ! Les comparaisons entre outils deviennent alors difficiles, tant les fonctionnalités finissent par être nombreuses. Il conviendra donc de prendre garde à la modularité et l'ouverture des constituants regroupés.

Dans la suite du chapitre, on prend résolument le parti de restreindre l'appellation ESB au seul composant responsable du transport et de l'acheminement des messages entre client et services. L'appellation APS (*Application Platform Suite*) désigne quand à elle l'ensemble des composants d'infrastructure présentée par la figure 20.1, y compris la chaîne de fabrication.

### 20.1.1 Le Bus d'Entreprise, ou ESB

#### *Les besoins*

Un ESB doit permettre à un développeur d'applications ou de services composites d'accéder aux services disponibles de façon standardisée, flexible et transparente.

Cela implique concrètement de rendre les consommateurs des services aussi indépendants que possible :

- Du protocole de communication utilisé par le consommateur et/ou par le service : le consommateur doit pouvoir accéder au service via HTTP, mais aussi via FTP, JMS, JCA, SMTP, RMI, .NET Remoting etc.
- De la technologie de déploiement des services : que ce service soit déployé comme Web Service, composant .NET, comme EJB ou comme un simple composant COM, Java ou PHP, le consommateur doit – dans l'idéal – y accéder de la même façon.
- De la localisation des services.

Symétriquement, l'ESB doit permettre à un développeur de services de déployer ses services quels que soient la technologie et le protocole de communication choisis. De plus, afin de faciliter l'adoption de l'approche SOA, on impose de plus en plus à un ESB d'offrir des outils d'accès à des composants existants, notamment aux référentiels légataires (EII/MDM).

### *Les caractéristiques technologiques*

De ce fait, le bus doit dans l'idéal :

- Supporter WSDL comme langage « universel » de description des services (même si le service est déployé en tant qu'EJB par exemple).
- Dialoguer avec les clients et les services via des protocoles hétérogènes :
  – Non WS-* (e.g. Natif HTTP, MQ, JMS, .NET, BAPI, CICS, etc.).
  – WS-* (e.g. SOAP, WS-RM, WS-Adressing, etc.).
- Proposer plusieurs modes d'appel de service (synchrone, asynchrone…).
- Tracer les appels de services :
  – Stockage de compteurs techniques, avertissements, erreurs.
  – Accès indirect (fichier, base de données) ou direct (HTTP, JMS, WMI, JMX, SNMP…) de ces traces vers une console de suivi technique SAM.
- Assurer la sécurisation des messages.
- Proposer l'hébergement de services composites locaux (support de la norme SCA).
- Proposer une console d'administration pour son paramétrage et le déploiement des services.

Pour satisfaire ces objectifs, la plupart des ESB s'appuient sur une approche « messagerie asynchrone ». Cela veut dire que, quels que soient les protocoles d'appel consommateur ↔ ESB, et ESB ↔ service, l'ESB transforme en interne ces appels en messages. Il assure ensuite le transfert et le routage en toute transparence vis à vis des consommateurs et pourvoyeurs du service. Le routage est basé sur l'en-tête d'un message, qui précisera destinataire et protocole, mais un routage évolué pourra prendre en compte le contenu du message. De plus, le routage peut se faire avec ou sans accusé de réception, avec ou sans garantie d'acheminement, etc.

Ces caractéristiques, font de l'ESB une sorte de passerelle « universelle », l'aboutissement, diront certains, de la longue marche débutée avec RPC et poursuivie avec CORBA, DCOM, RMI, EJB, les middleware orientés messages, JMS, les moniteurs transactionnels (TUXEDO), les brokers d'intégration asynchrones (EAI), etc.

### *L'architecture d'un ESB*

La figure 20.2 présente les différentes briques fonctionnelles nécessaires pour que l'ESB satisfasse ses objectifs.

Les composants « registre de services », « SAM » et « EII/MDM » sont décrits plus loin. On s'en tient, dans ce sous-chapitre, à ce que l'on peut appeler le cœur de l'ESB, c'est-à-dire :

- Le bus de message.
- Les adaptateurs de protocoles et connecteurs directs.
- Le moteur de transformation des messages.
- Le moteur de routage des messages.
- Le container SCA de services locaux.

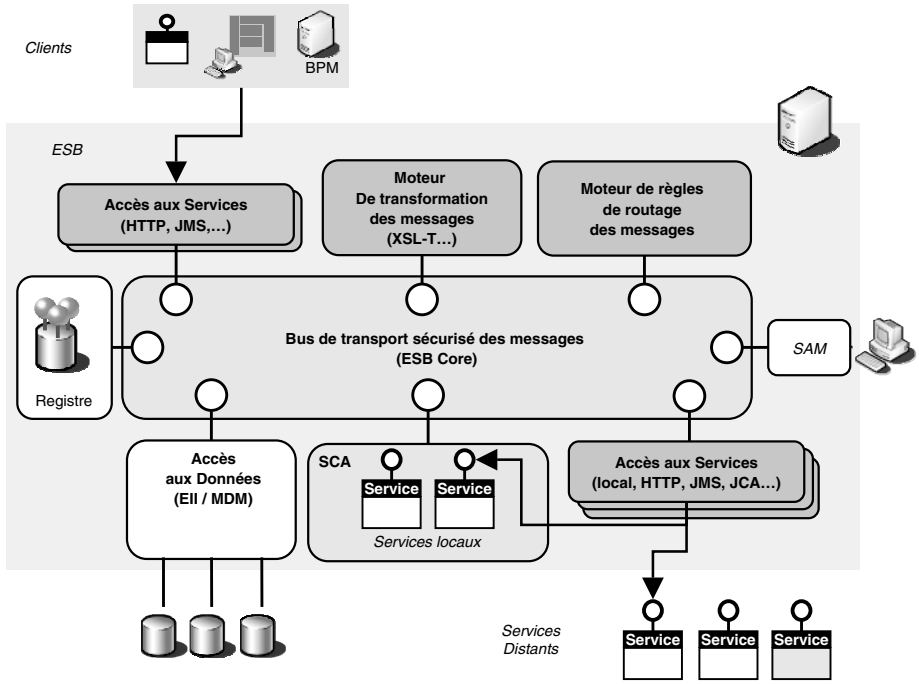



**Figure 20.2** — Zoom sur l'ESB

Le bus de message assure le transport et la sécurisation des messages entre les différents serveurs impliqués dans le déploiement des clients et des services.

Les adaptateurs de protocole permettent à l'ESB de supporter divers protocoles de communication dans son dialogue avec les consommateurs et les pourvoyeurs de services. L'ESB pourra aussi se voir muni de connecteurs directs afin de faciliter l'accès à l'existant sans couche service.

Le moteur de routage assure l'aiguillage des messages vers leurs destinataires selon des règles.

Le moteur de transformation assure la conversion syntaxique de format de messages entre deux protocoles (si le protocole d'invocation du service par le consommateur est différent du protocole d'activation du service par l'ESB). Il peut aller plus loin, en assurant également une conversion sémantique des informations XML échangées entre consommateurs et services (cf. les remarques du paragraphe « Transformer les messages »).

Les paragraphes ci-dessous reviennent sur quelques points clefs de l'ESB.

### *Appeler un service*

Le bus met à disposition des consommateurs de services plusieurs modes d'appel (ou mode d'interaction) de ces services :

- Mode synchrone.
- Mode asynchrone « *one way* ».

- Mode asynchrone avec « *call back* ».
- Mode asynchrone « publication/abonnement ».
- Mode conversationnel.

Le mode d'accès synchrone bloque le consommateur en attendant la réponse du service. C'est le mode classique d'accès, similaire d'emploi à un appel procédural dans un langage de programmation.

Le mode asynchrone « *one way* » libère le consommateur dès que celui-ci a appelé le service. Autrement dit, le consommateur n'attend pas de réponse du service. Ce mode a été popularisé par les MOM (*Message Oriented Middleware*) puis par les EAI. Il permet notamment de propager des événements entre applications. Le consommateur peut demander une garantie d'acheminement du message ou pas.

Le mode asynchrone avec « *call back* » libère le consommateur dès que celui-ci a appelé le service, mais le consommateur précise lors de son appel qu'il souhaite obtenir une réponse ultérieurement. Pour cela, le consommateur spécifie au service appelé quelle opération celui-ci doit rappeler pour envoyer sa réponse : cette opération est appelée *call back*.

Le mode asynchrone « publication/abonnement » permet à un consommateur de transférer de l'information simultanément vers plusieurs services, sans se soucier du nombre et de l'identité de ces services. L'idée de base est de permettre à un service de s'abonner (*suscribe*) à une publication de message. Lorsqu'un consommateur publie (*publish*) un tel message, c'est à l'ESB de prendre en charge la diffusion de ce message à tous les abonnés intéressés. Ce mode de communication permet notamment la diffusion d'événements ou d'informations (propagation des taux de changes ou de la cotation d'une action vers le front office et le back office d'une salle de marché, par exemple).

Enfin, le mode conversationnel permet au consommateur d'engager une conversation avec le service appelé, une conversation étant une suite d'appels et de réponses. L'intérêt de ce mode est d'alléger le travail de l'ESB, qui n'a plus à créer à chaque appel de service un nouveau contexte de travail : ce contexte technique, géré et stocké par l'ESB, est partagé entre chaque appel/réponse.

### *Transformer les messages*

La question se pose de savoir s'il faut aussi ajouter, au sein du bus, un moteur de traduction, à l'instar d'un interprète servant d'intermédiaire dans la vie réelle.

Cette inclusion peut constituer un piège. En effet, l'objectif premier du traducteur est d'effectuer des transformations syntaxiques entre protocoles de communication.

Mais il peut être tentant d'utiliser ce traducteur pour lui faire jouer un rôle d'EAI/ETL, c'est-à-dire un rôle de transformateur sémantique de données métier, en référence à un langage pivot définissant la sémantique de ces données. Or dans une approche SOA, ce sont les services métier qui doivent se charger d'une telle transformation ou adaptation, ce qui est nettement plus flexible.

### *Router les messages*

Quelles sont les méta-données permettant de router un message vers un service ? En général, le bus prendra en compte les paramètres suivants :

- Les informations contenues dans le contrat de service : adresse, protocole de transport (informations fournies par le contrat WSDL).
- La qualité de service demandé (par exemple : garantie d'acheminement et tolérance aux pannes, délai maximum de réponse, etc.) (informations fournies dans l'idéal par un fichier WS-Policy).
- L'entête et/ou le corps du message.
- Les règles de transformations (XSLT, XQuery, langage natif, etc.) à appliquer lors de la conversion vers le protocole interne de l'ESB.

## *Le fonctionnement dynamique*

Sans rentrer dans des détails techniques inutiles, il est intéressant de comprendre dans les grandes lignes le fonctionnement d'un ESB pour bien marquer la différence avec un simple appel de Web Service sur HTTP. La figure 20.3 propose un diagramme de séquence illustrant ce fonctionnement sur l'application « fil rouge ».

❶ Une demande de prestation saisie sur le portail par un fournisseur est envoyée via un connecteur JMS (protocole asynchrone) à l'ESB.

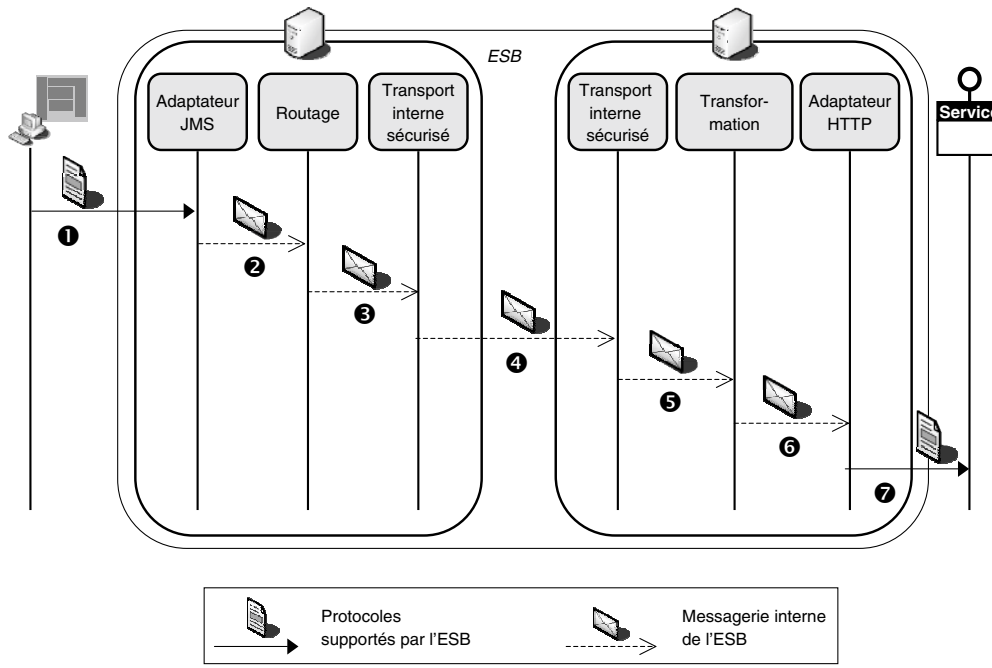



**Figure 20.3** — Fonctionnement de l'ESB

❷ Le connecteur transforme la demande en un message interne ESB et envoie ce message au moteur de routage.

❸ Celui-ci détermine la localisation du service métier à appeler (c'est un Service Applicatif, cf. chapitre 8), en fonction de la demande déposée.

❹ Le mécanisme de transport interne à l'ESB achemine le message de façon sécurisée vers le nœud de rattachement du Service Applicatif.

❺ Là, le moteur de transformation de l'ESB transforme le message reçu dans un message au format attendu par le service. Si on suppose que le service est déployé comme Web Service, le message va être traduit en format SOAP.

❻ Le message est transmis à l'adaptateur approprié – ici, l'adaptateur http.

❼ L'adaptateur invoque le service.

Le lecteur peut légitimement se demander si l'ESB ne devient pas une usine à gaz : pourquoi ne pas se contenter de déployer un « simple » frontal Web Service (tel que l'outil Open Source Axis 1.x) ? De nombreuses raisons militent en faveur d'une solution ESB pour un déploiement industriel de SOA au niveau de l'entreprise[1] :

- L'application utilisant le frontal JMS n'est pas bloquée par l'appel du service, ce qui peut être un élément important de la performance globale de la solution.
- L'acheminement de l'appel peut être sécurisé par l'ESB sans intervention des équipes en charge des solutions ou des services.
- Le consommateur peut déléguer à l'ESB la détermination de l'implémentation du service à appeler (via SCA) : cela peut notablement faciliter le découplage de l'évolution des clients d'un côté et des services de l'autre.
- Ce découplage peut également permettre de masquer la véritable adresse d'un service web à un consommateur externe au système d'information, ce qui contribue à la sécurisation des accès.
- L'ESB offre un niveau de supervision (console SAM) que n'offre pas un simple frontal Web Service.

En fait, l'offre ESB va probablement se structurer d'une façon comparable à la situation actuelle des serveurs d'application dans le monde Java : l'apparition d'offres très complètes tels que les serveurs IBM WEBSPHERE ou BEA WEBLOGIC n'a pas entraîné la disparition d'un outil tel que TOMCAT. Le niveau de maturité de l'appropriation de la démarche SOA par une direction informatique la fera probablement passer progressivement d'une architecture ESB simple et localisée, à une architecture plus complète, déployée au niveau de l'entreprise.

### *Le déploiement*

On se bornera à rappeler qu'il en va de l'ESB, comme des autres outils de messagerie : il y a deux topologies de déploiement d'un ESB, la topologie « décentralisée » (chaque pourvoyeur de service est un nœud de communication supportant une réplique du

1. Axis dans sa version 2.x propose d'ailleurs un certain nombre de fonctionnalités qui le rapproche d'un ESB.

noyau ESB) et la topologie « centralisée » (l'ESB est un hub de communication centralisé sur une machine).

La topologie décentralisée est la plus robuste (la perte d'un nœud de communication n'a pas d'impact sur les autres nœuds), mais aussi la plus complexe à installer et configurer. La figure 20.3 adopte cette topologie.

### *ESB versus EAI*

Force est de constater que certains EAI se sont retrouvés du jour au lendemain, baptisés du terme à la mode d'ESB. Quelques caractéristiques marquent cependant des différences notables.

**En quoi l'ESB diffère-t-il du broker EAI ?**

Le support des Web Services et plus généralement des normes WS*- (WSDL et BPEL par exemple), JBI et SCA est un premier point important. Le monde de l'EAI est en effet un monde propriétaire et monolithique, alors que le monde ESB se doit (par définition) de favoriser l'ouverture, la modularité tout en supportant une topologie décentralisée. Il peut dans ce cadre intéresser plus largement les communautés Open Source, même si à ce jour les initiatives restent, à l'instar des brokers EAI, en retard des produits commercialisés.
Le second point concerne l'utilisation de XML comme langage universel permettant de définir un modèle « pivot » des contrats (messages) métiers manipulables aussi bien dans l'ESB (transport des documents XML dans les messages ESB) que dans les clients et les services eux-mêmes (transformation document XML ↔ objets métier).
Le troisième point concerne la possibilité de publier des services via le Web.
Le quatrième point concerne le support d'un registre de services, permettant un déploiement « dynamique » de l'implémentation du service invoqué ainsi qu'un contrôle des contrats.
Le cinquième point concerne le routage de messages, plus sophistiqué avec SOAP et ses extensions, notamment sécuritaires.

Le transport des messages et événements est-il pour autant unifié (et standardisé) ? Sur le papier, on l'a vu, le bus ESB est indépendant par définition des technologies utilisées dans le SI.

Dans la pratique[1], l'ESB n'est pas encore aussi universel. Il est vrai que cette universalité est rendu complexe par la très grande diversité des technologies « classiques », telles que les moniteurs transactionnels (CICS, Tuxedo) et les EAI (WebMethods) dans la banque et l'assurance, les MOM (MQ series d'IBM ou RendezVous de Tibco) dans les salles de marché, Corba dans les télécoms, les ERP dans l'industrie, et plus généralement les middlewares du monde Java (RMI/JMS/EJB), ceux de Microsoft (.NETRemoting/DCOM), etc. Il convient alors de rechercher un « degré d'unification » via la réalisation de « bridges » entre l'ESB et la technologie légataire visée.

1. Ceci restant vrai actuellement pour la majorité des éditeurs.

Par ailleurs, il est possible d'envisager une période de transition entre EAI et ESB. Par exemple, les architectes du SI pourront envisager l'EAI pour établir les grandes départementales du SI, en même temps que l'ESB pour les boulevards périphériques du SI et les autoroutes inter-SI.

### 20.1.2 Le registre des services

#### *Les besoins*

Les développeurs d'applications consommatrices ont besoin d'accéder aux contrats décrivant les services pour en prendre connaissance.

L'infrastructure (ESB) a également besoin d'accéder non seulement aux contrats, mais également à la description des dépendances entre services et éventuellement aux implémentations de ces services.

Le concept de Registre de Service recouvre la réponse aux deux besoins :

- Le besoin d'un annuaire de services, contenant uniquement les contrats de service, et permettant de rendre public ces contrats et de les consulter.
- Le besoin d'un référentiel de services, contenant non seulement les contrats, mais également la description des dépendances entre services et éventuellement les implémentations de ces services.

> En fonction du périmètre de l'ESB associé et de l'intégration du registre à cet ESB, le registre peut également gérer les modèles de processus, les grammaires métiers, les scripts de transformation de messages, les règles de routage, les portlets, etc.

Le terme neutre de registre de services regroupe ces fonctions d'annuaire et de référentiel. L'annuaire peut supporter ou non la norme UDDI (cf. partie 4, chapitre 12). Le référentiel de service est lié au support de la norme SCA (cf. partie 4, chapitre 14).

L'introduction du concept de registre de services est donc clairement structurante pour SOA.

#### *L'architecture*

La figure 20.4 présente les fonctions d'un tel registre.

Les principales fonctions à attendre d'un tel registre sont donc :

- La publication des contrats et des implémentations, via une intégration plus ou moins poussée avec les ateliers de la chaîne de fabrication des services.
- La classification des services :
  - Documentation pour les consommateurs.
  - Indexation sur le plan technique (messages, interfaces, protocoles acceptables, SLA etc.).
  - Indexation sur le plan métier (domaine sectoriel, couverture géographique, tarification, réglementation, etc.).

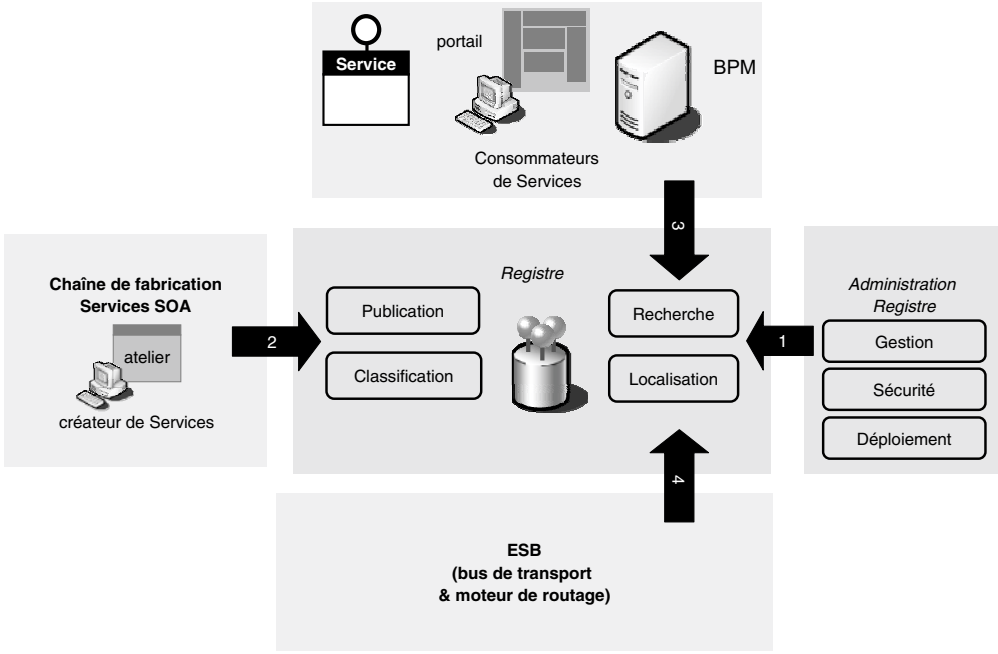



**Figure 20.4** — Zoom sur le registre de services

- La gestion des services :
  - Stockage des implémentations.
  - Gestion des modifications, des versions, des variantes, de plusieurs environnements (e.g. : étude, test, pré-production, production, etc.).
  - Gestion des dépendances entre services.
- La recherche (sécurisée) d'un service :
  - Recherche interactive (pages de documentation générées – souvent Web).
  - Recherche programmatique (requête de sélection).

### *Le déploiement*

Il existe potentiellement trois topologies principales de registres, comme l'illustre la figure 20.5 :

- **Référentiel centralisé :** Il apparaît suffisant dans un contexte d'utilisation interne à l'entreprise pourvu que les consommateurs ne soient pas eux-mêmes trop répartis; en revanche il ne peut convenir à un contexte inter-entreprise.
- **Multiples Référentiels de services :** Cette topologie est préférable si la contribution des pourvoyeurs de services est très répartie sans qu'un besoin d'unification pour les consommateurs soit nécessaire (par exemple, registres verticaux par domaines sectoriels).

- **Référentiel de services fédéré :** Il s'agit d'une combinaison des deux cas précédents, cette topologie donne l'illusion d'un unique référentiel pour le consommateur de service, tandis que des référentiels locaux à chacun des pourvoyeurs sont physiquement distribués et fédérés.

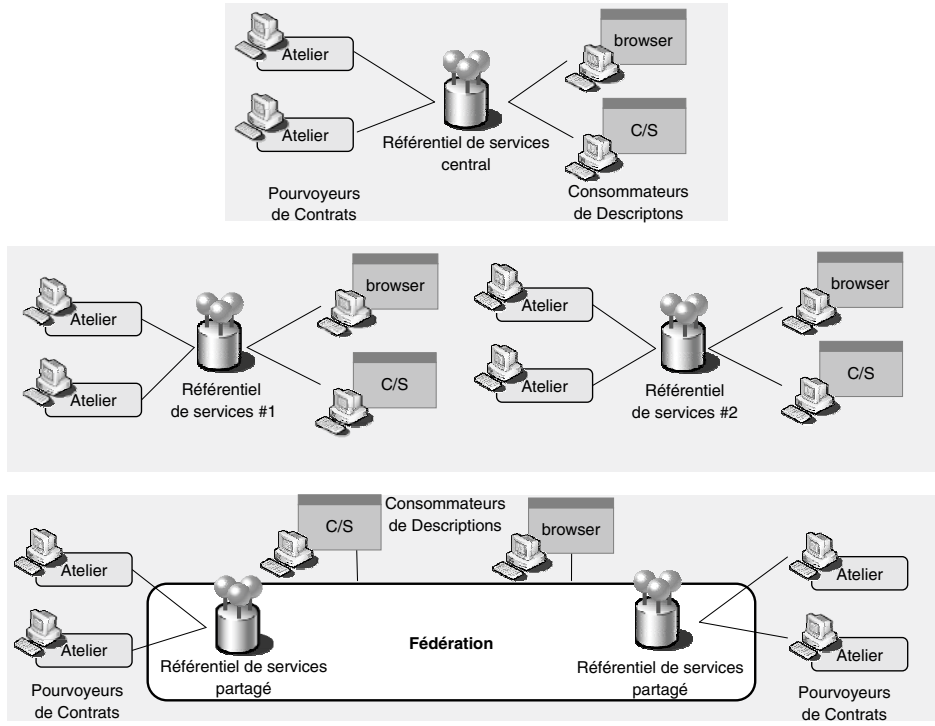



**Figure 20.5** — Les topologies d'un registre de service

## 20.1.3 L'accès aux référentiels de données (EII/MDM)

### *Les besoins*

L'accès CRUD aux informations métiers est un enjeu important de la création d'une bibliothèque de services réutilisables, comme on l'a vu partie 3 – chapitre 8. La création de services CRUD n'est pas aussi triviale qu'il peut paraître, il existe donc un besoin fort en matière d'outillage à la fabrication et l'exécution de tels services. Ce besoin peut être gradué du plus simple au plus complexe, selon le niveau de sophistication et de performance requis.

Le premier besoin, prioritaire, est la génération de services CRUD à partir d'une description des Informations Métier à manipuler. Ce besoin est couvert par les outils de type EII[1] ou directement par un module de la chaîne de fabrication.

1. La terminologie anglo-saxonne retenue est celle d'*Enterprise Information Integration* (EII).

Le deuxième besoin porte sur la nécessité d'agréger des données de base pour reconstituer ces informations métiers. Ce besoin se retrouve lorsque les référentiels concernés existent déjà, et qu'il n'est pas possible de les unifier sans les modifier trop profondément. On a déjà cité l'exemple classique de l'information métier « client », qui agrège des informations « personne morale » (= l'entreprise cliente), « personne physique » (= les correspondants dans cette entreprise cliente), « adresses géographiques » (adresse de livraison, adresse de facturation…), « adresses bancaires » « historique des commandes », « historique des paiements », « contacts marketing », etc. On peut également citer une autre utilité : celui d'harmoniser l'accès aux bases clientes des différentes filiales d'une même société : le besoin est alors d'effectuer des correspondances sémantiques entre les concepts, en utilisant des méta-informations (par exemple, la correspondance « nom du client = nom-client = last name = customerName », etc.). Ces besoins sont couverts par les EII les plus avancés.

**Les principes du MDM**

Le *Master Data Management* gère des tables métiers de références mises à jour en fonction d'une politique de réplication paramétrable, et visibles au travers de services. Contrairement à l'EII qui nécessite la disponibilité temps réel des sources, le MDM stocke les données.

Il s'agit donc de collecter et concentrer toutes les données relatives à un objet métier (client, fournisseur, produits…) ainsi que leurs meta-données de structuration (clés d'accès sur la source d'origine maître, la version, un ou plusieurs champs contextuels tels que la dernière commande, etc.) afin qu'elles soient partagées à l'échelle de toute l'entreprise voire du secteur industriel dans un format pivot. Il s'agit principalement des données servant de point d'entrée aux processus métiers.

Chaque valeur est unique et dispose d'un identifiant unique (elle a fait l'objet d'un processus de mise en conformité/qualité). Elle peut être versionnée (via les méta-données) en vue d'une exploitation analytique. Le référentiel du MDM peut lui-même être dupliqué pour des raisons de performances (à condition de rester à l'identique de l'original).

Le MDM peut aussi servir pour la traçabilité des données et permet de soulager les processus d'intégration par les données. Il contribue à une volonté d'amélioration des processus transverses à plusieurs départements dans l'entreprise (par exemple, les éléments tels que contractualisés au départ par une équipe commerciale, puis tels que facturés par une équipe comptable, ou tels que fabriqués par une filiale, etc.). Ceci permet de dégager les données clefs utiles à chaque département métier.

Les caractéristiques différenciatrices du MDM, sont principalement :

- La nature des données centralisées (dédiée produit, client, organisation etc.) avec ou sans meta-données.
- Le type de pivot (à construire ou prêt à l'emploi[1]).
- La qualité des données (c'est le point le plus délicat car il se heurte aux aspects organisationnels : nettoyage de données non utiles, conformité de structure et de valeurs, suppression des doublons, etc.).
- Le type d'administration (batch ou temps réel).

1. Il existe en effet des MDM spécialisés dans la consolidation de données Clients (CDI : *Customer Data Integration*) ou la consolidation de données Catalogue Produit (PDI : *Product Data Integration*).

Le troisième besoin porte sur la nécessité de répliquer les données agrégées pour accroître les performances d'accès à ces informations ou pour garantir la robustesse de la solution métier (cf. partie 3 – chapitre 7). Ce besoin, qui recouvre également le deuxième besoin, est couvert par une nouvelle race d'outils, baptisée MDM – *Master Data Management*. On notera dans ce cas qu'il est utile qu'à la fois les données et les méta-données soient disponibles et unifiées dans le MDM, même si dans ce cas, certaines données dont la fréquence de modification est trop élevée, ne peuvent pas être répliquées.

### L'architecture

La fonction première d'accès CRUD repose sur le concept de méta modèle métier. Ce méta modèle reprend le modèle métier en y ajoutant des informations nécessaires à la mécanique d'accès : par exemple, est-ce que l'attribut « code postal de l'adresse du fournisseur » est utilisé comme critère de Recherche sur la base « fournisseur » (le R de CRUD) ?

La fonction d'agrégation repose sur le concept de Vue de données : ce concept est similaire à celui proposé par les SGBD relationnels. Une vue définit une information « virtuelle » via une agrégation de données. Une vue décrit donc :

- Les types de données de base, agrégés pour former la vue (par exemple : la vue « fournisseur » est l'agrégation d'une « personne physique » et d'un « contrat »).
- La correspondance entre attributs (par exemple, nomDuClient = customerName, etc.).
- Les informations d'accès technique aux données agrégées (par exemple : la « personne physique » est accédée via une clé primaire SQL, le « contrat » est accédé via un identifiant SAP).

La fonction de réplication repose sur les concepts de Cache de réplication et de Politique de Réplication. Le Référentiel « Cache » contient les données répliquées, considérées comme des données maîtres (d'où le nom de *Master Data Management*), et constitue un cache d'informations unifiées. Une politique de réplication définit la fréquence et les modalités de réplication entre chaque référentiel source de données, et le référentiel « maître ».

La figure 20.6 présente en conséquence l'ensemble de l'infrastructure associée à un tel outil d'accès aux référentiels. Cette infrastructure comprend des outils de modélisation, une console d'administration, et un moteur d'exécution. En pointillé, les fonctions spécifiques d'un outil de type MDM.

Chaque connecteur permet d'accéder à un type de référentiel en utilisant le dialecte attendu par ce type. Le dialecte supporté en standard est bien sûr SQL. L'accès à des progiciels reconnus peut également être proposé. Certains outils EII/MDM prétendent également offrir l'accès à des fichiers plats (fichiers Excel, etc.).

La fonction de réplication exécute les réplications de données en appliquant les politiques de réplication définies au préalable et déployées via la console d'administration de l'outil. Elle s'appuie sur les connecteurs de données.

La fonction de transformation agrège les données en utilisant la définition des Vues de données modélisées dans l'atelier logiciel et déployées via la console d'administration de l'outil.

La fonction de gestion du référentiel maître assure la gestion de ce cache persistant. Elle propage éventuellement les modifications apportées aux informations sur ce référentiel vers les référentiels sources.

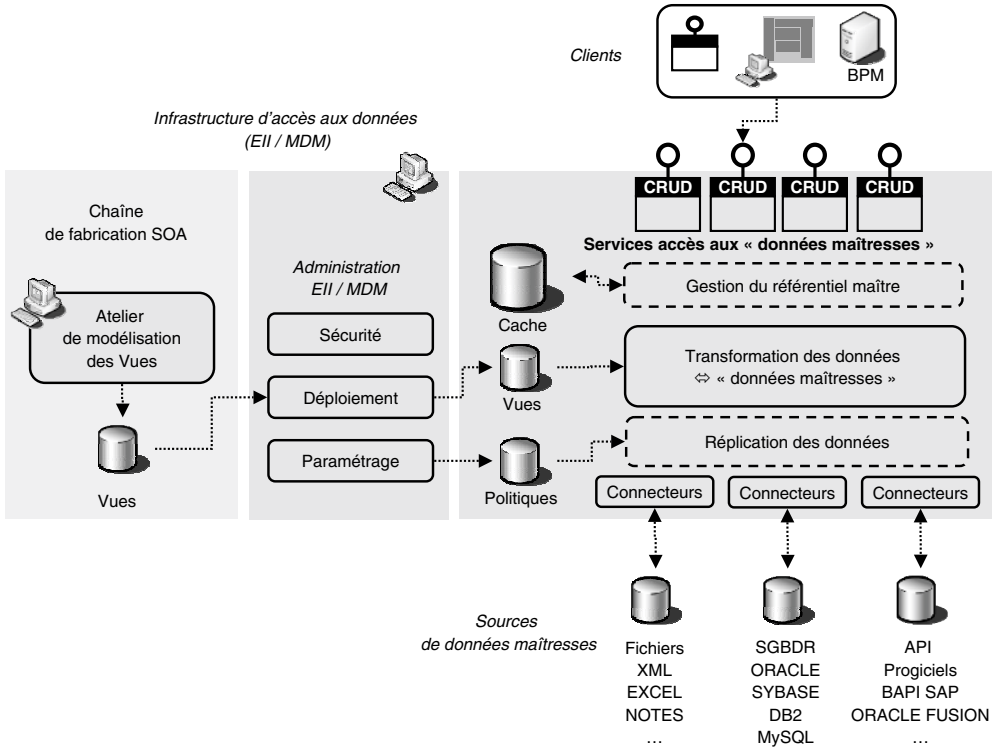



**Figure 20.6** — Zoom sur l'accès aux référentiels

Les services encapsulent cette mécanique EII/MDM, et offrent les opérations CRUD utilisables par les clients. En fonction de l'outil, soit ces services accèdent directement au transformateur de données, qui s'appuie sur la définition des vues pour cibler en temps réel les référentiels source des données à agréger, soit ils accèdent uniquement au référentiel maître.

### *Le déploiement*

Les différences entre un outil MDM et un outil EII tiennent au fait que l'EII accède en temps réel aux référentiels sources, et n'a pas besoin d'offrir de réplication de données et donc pas de cache de consolidation : il a donc un champ d'application plus large en terme d'accès aux données (tout type de données est accessible via un

EII, y compris les données fortement transactionnelles), même s'il reste nécessaire de veiller aux performances d'accès.

Certains outils proposent de générer le code des services CRUD, ce qui est moins souple en terme de déploiement, mais plus sûr en terme de qualité de code et d'optimisation des performances. Il n'est pas besoin de rappeler combien ces performances d'accès sont critiques pour la performance globale de toute solution métier.

### 20.1.4 Le moteur BPM

#### *Les besoins*

Le besoin principal porte sur la capacité à modéliser les processus métiers, et à pouvoir utiliser ces modèles pour paramétrer directement le moteur d'exécution de processus.

Les processus concernés sont principalement des processus e-business, plus ou moins automatisés, dans lesquels il doit être possible de faire intervenir des acteurs humains.

#### *L'architecture*

La figure 20.7 présente l'architecture du moteur et les outils associés (atelier de modélisation, console d'administration, console de monitoring BAM).

Les outils liés au workflow humain ont été décrits au chapitre 9.

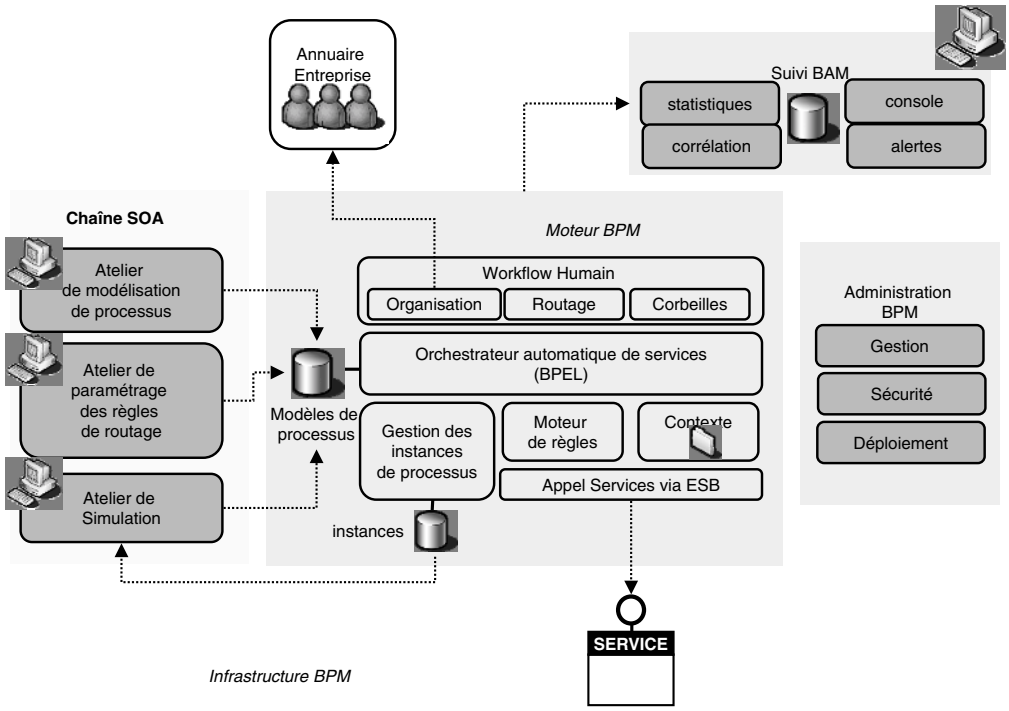



**Figure 20.7** — Zoom sur le BPM

### 20.1.5 Les consoles de suivi SOA

*Les besoins*

Comme tout logiciel, les solutions métiers SOA doivent faire l'objet d'une surveillance en production, ou monitoring. Le besoin de monitoring spécifique de SOA est double :

- Paramétrer et visualiser des indicateurs sur le comportement des services : c'est le besoin SAM – *Service Activity Monitoring*.
- Paramétrer et visualiser des indicateurs sur le comportement des processus : c'est le besoin BAM – *Business Activity Monitoring*.

Via la console SAM, un exploitant ou un responsable de service pourra visualiser en temps réel le comportement de chaque service : courbe du nombre d'accès en fonction de l'heure, temps de réponse moyen/minimal/maximal, volumétrie des paramètres d'entrées/sorties, erreurs détectées, etc. Il pourra également visualiser le dépassement de seuils et ainsi détecter des alertes.

Ces données SAM sont historisées, ce qui permet de revenir sur le comportement d'un service sur plusieurs jours, et de surveiller par exemple une éventuelle dégradation progressive de la qualité de service.

Via la console BAM, un administrateur de processus, un responsable d'une solution métier ou un responsable de la maîtrise d'ouvrage pourront visualiser en temps réel le comportement d'un processus (date de démarrage, état actuel, durée d'exécution de chaque activité, exception levée, etc.). Ils pourront également visualiser le dépassement de seuils (exemple : un processus tarde à se terminer) et ainsi émettre des alertes.

Ces données BAM sont historisées, ce qui permet d'établir des statistiques sur le comportement d'un ensemble de processus : par exemple, quel est le temps moyen d'exécution d'un type de processus, quelle est la fréquence d'une exception donnée, etc.

Le besoin général en matière de paramétrage BAM/SAM porte donc sur :

- La politique de recueil des données temps réel (fréquence, type de trace…).
- Le choix des indicateurs à visualiser en agrégeant les traces recueillies.
- La définition d'alertes sur dépassement de seuil critique.
- La définition d'historique consolidant les données temps réel.
- L'export des données dans un format approprié (Excel…).
- La définition de rapports d'analyse décisionnelle (ceci pouvant être délégué à des outils décisionnels existants, à condition qu'il soit possible d'exporter les données nécessaires).

Dans un futur plus ou moins proche, les outils BAM/SAM devraient permettre d'anticiper sur l'évolution des ressources du système, en offrant la possibilité :

- d'effectuer des simulations de charge à partir des statistiques obtenues, de la description de l'architecture (services) et des modèles de processus concernés,

- de définir des règles d'utilisation automatique des ressources « *on demand* » par segmentations et auto ajustement de ces ressources, en plus ou en moins.

### *L'architecture*

Les caractéristiques essentielles sont :

- Une instrumentation de chaque composant et de l'infrastructure (ESB, moteur BPM), par des sondes (ou agents) distribuées.
- Un ou plusieurs serveurs de recueil et d'archivage des traces recueillies par les sondes.
- Une console d'administration du monitoring, permettant de paramétrer les niveaux de sensibilité des contrôles, avertissements, erreurs et options d'instrumentation.
- Les consoles BAM/SAM elles-mêmes, incluant les outils de restitutions, allant du simple indicateur aux outils de statistiques et présentation personnalisables. Elles peuvent inclure des vues graphiques, des tableaux, des visualisations d'alertes, etc..
- Des interfaces d'intégration avec des outils d'exploitation systèmes et réseaux, avec une solution décisionnelle (notamment via l'usage d'un serveur de corrélation d'événements – CEP), voire avec une solution de facturation de l'utilisation des services, etc.

## 20.2 LA CHAÎNE LOGICIELLE SOA

Les ateliers de développement s'enrichissent de plus de plug-ins et frameworks permettant de compléter des actions spécialisées nécessaires à chaque rôle et peuvent être intégrés pour former une chaîne de fabrication logicielle[1] la plus déterministe et automatisée possible. Ainsi SOA n'échappe pas à la nécessité d'harmoniser les tâches nécessaires autour des services et de leur description formelle. En outre apparaissent progressivement des accélérateurs pour :

- La modélisation visuelle de processus métier et la codification explicite de la logique de séquence.
- La génération d'interfaces ou de squelettes de services à partir de diagrammes UML.
- La modélisation des formulaires pilotés par les grammaires évitant la saisie brute de fragments XML.
- La gestion en configuration de variantes de services, soit pour agir dés la conception, soit pour permettre à l'exécution le fonctionnement de plusieurs variantes en bonne cohabitation.

1. *Software Factory* en terminologie anglo-saxonne.

- Les tests d'intégration continue de services, permettant à tout fournisseur de service souhaitant tester au plus tôt son implémentation, de solliciter celle-ci sans attendre la réalisation des implémentations de services dont il dépend. Il utilise pour se faire, en lieu et place, des services « bouchons » permettant de fournir un résultat au travers de jeux d'essai répondant aux interfaces convenues[1].
- Etc.

### 20.2.1 L'atelier de modélisation

L'objectif de l'atelier est de proposer des éditeurs (graphiques ou textuels) permettant de modéliser les différents composants logiciels d'une même solution métier. Il permet d'entamer une démarche MDA (cf. encart MDA chapitre 11).

Il doit comprendre au minimum :

- Un modeleur UML.
- Un modeleur XML.

#### *L'atelier UML*

Le modeleur UML permet de modéliser et gérer les différents modèles métiers (modèle de processus, modèle d'objets métiers, modèle de cas d'utilisation, etc.), encore appelé PIM (*Platform Independant Model*).

Le modeleur UML offre aussi dans l'idéal de gérer des documents textuels décrivant ces composants (cas d'utilisation, fiche descriptif des services, etc.).

Cet atelier supporte si possible la notation BPMN pour modéliser les processus métiers d'une façon lisible par la maîtrise d'ouvrage et transformable en WS-BPEL par l'architecte. Il peut aussi offrir un mécanisme d'export XPDL, lui aussi exploitable par les équipes techniques.

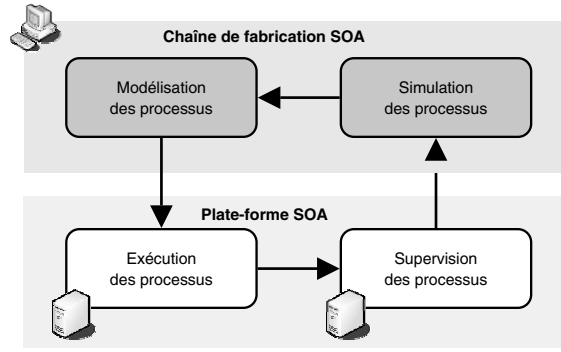



**Figure 20.8** — Modélisation et simulation de processus

1. Il est une pratique fréquente de proposer que chaque fournisseur produise en même temps que le contrat du service, le jeu d'essai permettant de tester le service rendu.

Certains éditeurs proposent également un simulateur de processus. Un tel outil simule l'exécution de processus associés à un ou plusieurs modèles de processus, en tenant compte, d'une part, des performances mesurées au préalable par l'outil de surveillance, d'autre part, d'hypothèses sur les événements métier à traiter, et enfin, d'hypothèses sur les capacités de l'infrastructure.

Cela permet, en théorie, d'analyser l'impact d'une évolution du modèle de processus, avant de déployer cette évolution. Cela n'avait que peu d'intérêt dans le cadre d'une gestion de workflow, mais devient intéressant pour anticiper d'éventuels problèmes ou apprécier le gain de réactivité dans le cas de processus automatisés par des machines.

#### *L'atelier XML*

Le modeleur XML permet de définir et gérer les composants logiciels à l'aide de vocabulaires XML spécialisés, ou DSL. Ces modèles sont des PSM (*Platform Specific Model*) car ils ciblent une architecture ou une technologie particulière. Le modeleur UML est associé à des générateurs assurant la transformation PIM (UML) → PSM (DSL).

Les DSL ciblés pour les outils basés sur WS-* sont :

- BPEL pour les processus métier;
- WSDL pour les contrats de base des services métiers;
- WS-SecurityPolicy pour les directives de sécurité;
- WS-Policy pour les autres directives;
- BOXML (*Business Objects XML*) pour les Objets Métiers et la génération des services CRUD;
- XFORMS, XUL ou XAML pour les Vues des Applications Interactives;
- xxx-config.xml (xxx = Struts, JSF ou équivalent) pour les Contrôleurs de navigation;
- SCA.xml pour les dépendances entre services.

Le premier avantage d'un tel atelier est donc de supporter les phases d'expression de besoin et d'analyse des solutions métiers, en permettant de modéliser services, processus métiers et applications composites, comme l'a illustré la partie 3.

Le deuxième avantage de cet atelier est d'apporter le niveau de productivité nécessaire.

### 20.2.2 L'atelier de fabrication

L'atelier de fabrication comprend d'une part un atelier de codage « manuel » et d'autre part des générateurs de code.

#### *Atelier logiciel de codage*

Cet atelier n'est pas spécifique à l'approche SOA. Il s'appuie donc sur les ateliers existants (ECLIPSE, IBM RAD, SUN NetBeans, Visual Studio.Net, etc.).

#### *Générateurs automatisés de code*

L'atelier de fabrication inclue également des générateurs assurant la transformation PSM → code. Ces générateurs cibleront :

- Les formulaires de consultation ou modification, écrans/pages *back office*, etc.
- Le code des services CRUD.
- Le code des proxys d'accès aux services via l'ESB, etc.

### 20.2.3 L'atelier d'assemblage et de déploiement

L'atelier d'assemblage permet de compiler et construire les différents livrables composants une solution métier. Ce processus d'assemblage est piloté par une description de ces livrables et par une description du processus de livraison lui-même.

L'atelier permet notamment de rassembler automatiquement les services en modules de déploiement. Pour cela, un éditeur permet de décrire les dépendances entre services et de spécifier la composition des modules grâce aux fichiers SCA (cf. la description de SCA à la partie 4).

Une fois les livrables construits, l'atelier les déploie sur la plate-forme visée (plate-forme d'intégration, plate-forme d'homologation, plate-forme de recette, etc.)

### 20.2.4 L'atelier d'homologation

L'atelier d'homologation permet d'effectuer l'ensemble des tests nécessaires à la mise au point des solutions métier et des services : tests d'assemblage, tests de bon fonctionnement, tests de performance, etc.

Idéalement, un tel atelier est organisé autour d'un outil de gestion des tests, qui assure d'une part la planification des tests, d'autre part permet de lancer l'exécution de ces tests et d'archiver les résultats obtenus. Cet outil de gestion collabore avec les différents outils de tests disponibles.

Les outils de tests de service permettent de spécifier et d'exécuter des scénarios de tests de services, ce qui implique de :

- Définir un ensemble de messages WSDL de requêtes, ainsi que la réponse attendue pour chaque requête.
- Définir les scénarios de tests, c'est-à-dire l'enchaînement de messages à envoyer.
- Définir des configurations de tests (il faut simuler N consommateurs simultanément actifs, par exemple).
- Lancer une campagne (on appelle campagne une exécution d'un ou plusieurs scénarios associés à une configuration particulière).

Les outils de tests de Processus SOA permettent de la même façon de spécifier des scénarios d'injection de messages dans un processus BPEL, avec simulation pos-

sible de charge. La figure 20.9 décrit le cycle complet mettant en relation modélisation, tests, supervision et simulation. La mise en place progressive de ce cycle est l'un des éléments clés permettant de mesurer la maturité d'une démarche SOA.

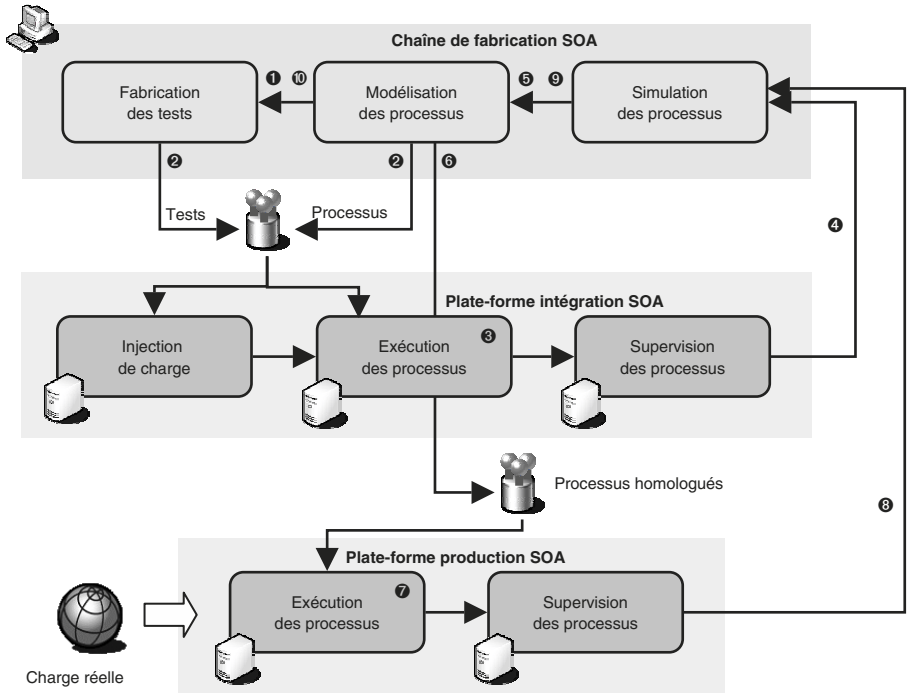



**Figure 20.9** — Tests des processus

## En résumé

Le déploiement de solutions SOA repose sur une infrastructure complète. Il est donc important de bien comprendre le rôle et la place de chacun des composants de cette infrastructure.

L'ensemble des briques potentielles d'une *Application Platform Suite* reste à ajuster en fonction du contexte de chaque architecture mise en œuvre. SOA ne peut s'implémenter au travers d'un seul et même outil, produit voire même fournisseur. Plusieurs composants de la plate-forme sont à sélectionner en fonction des points d'agilité les plus critiques à résoudre entre les métiers et l'informatique. De nombreuses autres techniques traditionnelles d'intégration demeurent toujours valables mais dans des cas d'utilisation en général plus restreints.

# 21

# Aide au choix

## Objectif

Dans la partie 4, on a pu constater que les standards autour des Web Services sont encore peu stabilisés, à l'exception du socle de base. Par conséquent, les solutions du marché font aujourd'hui leurs propres choix d'implémentation. De plus, compte tenu de la richesse de l'infrastructure SOA décrite au chapitre précédent, tous les offreurs ne peuvent ou ne souhaitent pas offrir l'ensemble des modules de cette plate-forme.

Ainsi est-il commun de rencontrer d'importants écarts de périmètre dans les plates-formes disponibles.

L'objectif de ce chapitre est de proposer une aide au choix parmi les solutions SOA du marché. Le lecteur qui désire déployer une infrastructure en support à son architecture SOA trouvera ici des critères d'analyse lui permettant de faire un choix raisonné.

## 21.1 INTRODUCTION À L'ANALYSE PAR GRILLE DE CRITÈRES

Le principe général de l'analyse par grille de critères consiste à mener une étude en vue du choix d'un outil, et ce sur la base de listes de critères permettant de qualifier et de comparer de manière détaillée une famille de solutions.

Cette démarche passe par les étapes suivantes :

- **Définition des critères de choix de la solution sur la base des besoins identifiés par les équipes impliquées dans le projet.** Ces équipes sont générale-

ment constituées de la maîtrise d'ouvrage du projet, des architectes en charge de la mise en œuvre de la solution (équipe infrastructure et équipe architecture applicative, cf. chapitre 11). La grille de critères pourra aussi être récupérée auprès d'un tiers ayant effectué un choix de solution similaire (autre maîtrise d'œuvre, cabinet de conseil, institution de recherche, etc.).

- **Pondération de la grille de critères.** Il s'agit ici d'attribuer à chaque critère une pondération sur une échelle généralement de 1 à 4 (priorité critique, haute, moyenne, ou basse). Ainsi les sponsors du projet pourront quantifier de manière extrêmement précise les critères les plus importants à leurs yeux.
- **Analyse des solutions :** il est ensuite temps de confronter les offres du marché à la grille. Cette étape peut s'effectuer au travers de la documentation de la solution (disponible sur Internet ou auprès de l'éditeur), ou bien en rencontrant les éditeurs pour les soumettre aux questions de la grille. On donnera alors une note pour chacun des critères (par exemple 1 point si telle caractéristique, 1 autre point si tel supplément sur la caractéristique et zéro point autrement).

| N° | Intitulé du critère | Coefficient de pondération | Note | Commentaires |
|---|---|---|---|---|
| 1. | Pérennité de l'offre | Haute | 1 | |
| 2. | Interface de consultation associée | Moyenne | 2 | |
| 3. | Incorporation à une architecture plus globale | Basse | 0 | |
| 4. | Nb clients, principales références | Critique | 2 | |
| 5. | Coût fixe proposé | Haute | 2 | |
| 6. | Coût récurrent proposé | Moyenne | 1 | |

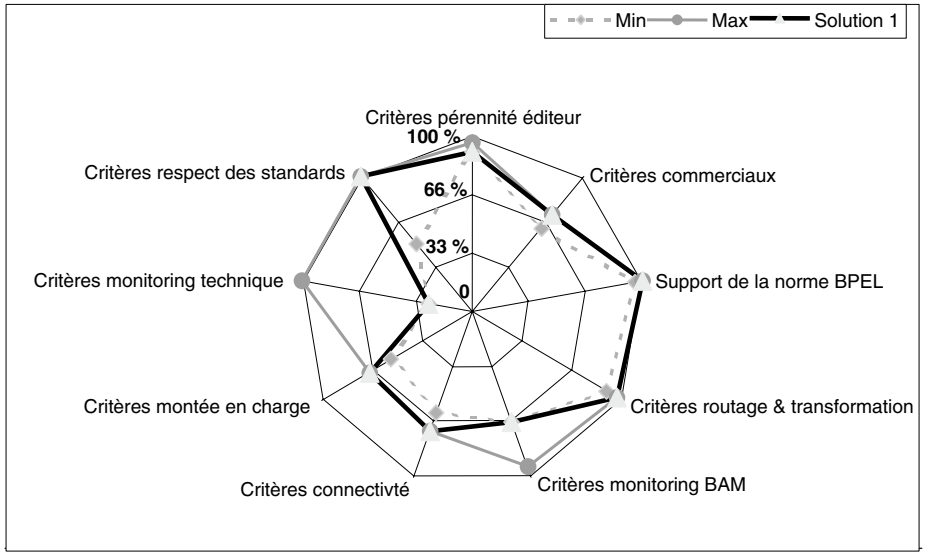



**Figure 21.1** — Outils d'analyse par grille de critères

- **Agrégation des résultats :** cette étape consiste à analyser les notes et les pondérations afin d'en sortir des indicateurs. Ces indicateurs, graphiques si possible, seront les outils d'aide à la décision pour arrêter la solution cible**.**

La figure 21.1 propose un exemple d'extrait de grille de critères et un exemple de graphe d'aide à la décision.

# 21.2 LA GRILLE DE CRITÈRES SOA

Ce paragraphe présente les éléments à prendre en compte dans la grille de critères dans le cadre d'une démarche de choix de plate-forme SOA.

Les critères présentés ici sont des méta-critères, c'est-à-dire qu'ils offrent une vision de premier niveau de la grille. Ils devront être affinés en fonction des besoins précis identifiés par les équipes impliquées dans le projet de déploiement de plate-forme SOA.

## 21.2.1 Les critères généraux

Les critères présentés ici revêtent un caractère généraliste : ils sont en effet pertinents pour toutes les démarches de choix d'outils. Ils relèvent du bon sens et peuvent être utilisés pour faire un premier filtrage dans le cas où trop de candidats initiaux se présenteraient.

**Tableau 21.1** — Critères généraux pour une présélection d'offres SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| G1 | Références | Nombre de références en production attestant de la stabilité et de l'expérience de la solution en situation de production. |
| G2 | Appropriation | Maturité du fournisseur et recul sur les enjeux et problématiques. |
| G3 | Prise en main | Ergonomie des interfaces destinées aussi bien aux concepteurs, développeurs, administrateurs ou aux métiers. |
| G4 | Richesse fonctionnelle | Prise en compte et couverture de l'état de l'art. |
| G5 | Évolutivité | Existence d'une API publique et documentée permettant de faire évoluer la solution en cas de besoins spécifiques. |
| G6 | Portabilité | Capacité de la solution à se déployer sur des infrastructures préexistantes (serveurs d'application, base de données). ➢ |

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| G7 | Interopérabilité | Capacité d'intégration avec l'existant, connectivité, caractère ouvert de la solution pour permettre l'échange de modules. |
| G8 | Maintenance | Homogénéité des outils et consoles d'administration d'une part et suivi d'autre part. Capacité à gérer la production et l'exploitation selon les besoins. Aspect non « boite noire ». |
| G9 | SAV | Service après vente et support du fournisseur disponibles. |
| G10 | Organisation | Impact organisationnel. Séparation des rôles dans la solution. |
| G11 | Pérennité | Stabilité financière du fournisseur de solution, caractère stratégique de la solution dans son offre. |
| G12 | Coûts de développement | Lorsque la prise en main nécessite une part de développement spécifique, facilité et rapidité de ce développement. |
| G13 | Coûts directs | Coûts d'acquisition en licences. |
| G14 | Coûts indirects | Coûts de support, de formation, de maintenance. |
| G15 | Standards | Respects des normes et standards (cf. partie 4 pour les standards Web Services). |
| G16 | Fiabilité | Robustesse, disponibilité, résistance aux pannes de la solution et capacité à répondre aux politiques de sécurité et d'intégrité (pouvant nécessiter un POC[1]). |
| G17 | Performance | Capacité à monter en charge (pouvant nécessiter un POC). |

1. POC signifie *Proof Of Concept*. C'est un prototype permettant de valider les fonctions avancées par le fournisseur.

### 21.2.2 Les critères techniques

Cette famille de critères se décompose en sous-rubriques, qui s'appuient sur la description du chapitre 20 :

- **Socle :** il s'agit de qualifier la complétude du socle technique (en premier lieu l'ESB), sa capacité à répondre à toutes les attentes techniques de part la richesse de ses frameworks de base.
- **Infrastructure de Coordination :** il s'agit ici de l'invocation de services au sein de l'ESB, qu'elle soit issue d'un échange point à point, composé, orchestré ou collaboratif. En particulier la capacité de le faire sans recourir à du code :
  - **Protocoles de Transports :** on qualifie ici les protocoles de transport pris en charge par la plate-forme à la fois en entrée ou en sortie.

  - **Routage :** on qualifie ici les modes d'acheminement des messages pris en charge par la plate-forme.
  - **Transformation :** il s'agit ici de traduire un message d'un format dans un autre.
  - **Connecteurs directs :** il s'agit de la capacité de la plate-forme à s'interfacer directement avec des sources d'informations (sans recours à un serveur applicatif tiers ou à un service).
- **Infrastructure d'informations :** il s'agit des caractéristiques essentielles du distributeur de requêtes (EII) et/ou du réplicateur d'informations (MDM).
- **Registre de services :** on qualifie l'annuaire de services et sa richesse fonctionnelle en tant que gestionnaire du référentiel SOA.
- **Administration :** il s'agit ici d'assurer une gestion cohérente unifiée des référentiels d'infrastructure SOA (annuaire de services, méta-données, vues logiques, etc.) et de la sécurité. Ceci doit pouvoir se réaliser au sein d'équipes réparties.
- **Suivi :** cette rubrique traite du suivi des échanges par l'informatique et les décideurs métiers c'est-à-dire du SAM et BAM.
- **Chaîne de fabrication :** il s'agit ici de l'ensemble des outils fournis par la solution pour modéliser, générer ou implémenter, tester et déployer les services et les compositions.
- **Présentation de compositions :** les composants permettant la mise en œuvre rapide d'interfaces utilisateurs pour applications composites interactives ou semi-automatisées, comme par exemple un portail web, un client riche, etc.
- **Périmètre métier :** cette rubrique se rapporte à la fourniture complémentaire de solutions métiers déjà SOA (cf. chapitre 22).

Les tableaux suivants étoffent la description de ces différentes sous-familles de critères.

### *Socle de la plate-forme*

**Tableau 21.2** — Les critères de couverture du socle SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| S1 | Développement | Proposition d'une API complète permettant de développer des extensions à la plate-forme. |
| S2 | Test | Outillage pour mener des simulations, des tests d'exécution avant connexion aux applications réelles. |
| S3 | Exécution | Environnement d'exécution distribuable (si possible basé sur un serveur d'application du marché). |
| S4 | Stockage | Intégration d'un référentiel consolidé pour le fonctionnement de la solution (paramétrage). |
| S5 | Topologie | Choix d'une topologie de bus versus topologie de hub. |

### *Coordination*

**Tableau 21.3** — Principaux critères d'une infrastructure de coordination SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I1 | Orchestration | Fourniture d'un moteur capable d'exécuter des orchestrations de processus (généralement suivant la spécification BPEL). |
| I2 | Règles de coordination | Fourniture d'un moteur de règles capable de traiter des algorithmes complexes de routage d'activités dans un processus. |
| I3 | Workflow interactif | Fourniture d'un moteur capable d'exécuter des séquences par interventions humaines. |
| I4 | Contexte Métier | Fourniture d'un framework de gestion de contexte métier. |
| I5 | Collaboration | Support d'événements et de protocole de communication directe entre processus. |

### *Protocoles de Transport de messages*

**Tableau 21.4** — Les critères techniques des transports SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I-P1 | Mode Query/Reply HTTP et HTTPS | Fourniture d'un adaptateur de protocoles capable d'acheminer des messages (entrants et sortants) en HTTP et version sécurisée. |
| I-P2 | Mode Message SOAP+WS-Security | Fourniture d'un adaptateur de protocoles capable d'invoquer des services en technologie SOAP et variante WS-Security. |
| I-P3 | Mode Message Passing SMTP et SMTPS | Fourniture d'un adaptateur de protocoles capable d'acheminer des messages SMTP (*Simple Mail Transport Protocol*) et version sécurisée. |
| I-P4 | Mode Fichier FTP et SFTP | Fourniture d'un adaptateur de protocoles capable d'acheminer des fichiers en FTP et variante sécurisée. |
| I-P5 | Mode Message IIOP | Fourniture d'un adaptateur de protocoles capable d'invoquer des services en technologie CORBA IIOP. |
| I-P6 | Mode Asynchrone MOM | Fourniture d'un adaptateur de protocoles capable d'acheminer des messages en technologie de Middleware Orienté Message du marché (MQSeries, MSMQ,TibcoMQ, etc.) incluant le mode publication/abonnement d'événements. |
| I-P7 | Message Asynchrone JMS | Fourniture d'un adaptateur de protocoles capable d'acheminer des messages en API Java unifiée JMS (*Java Message Service*). |
| I-P8 | Mode Query/Reply RMI | Fourniture d'un adaptateur de protocoles capable d'invoquer des services distants en technologie Java RMI. ➢ |

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I-P9 | Mode Messages .NetRemoting | Fourniture d'un adaptateur de protocoles capable d'invoquer des services distants en technologie Microsoft.NetRemoting. |
| I-P10 | Mode Query/Reply DCOM | Fourniture d'un adaptateur de protocoles capable d'invoquer des services distants en technologie Microsoft DCOM. |
| I-P11 | Transactionnel Tuxedo, CICS, IMS | Fourniture d'un adaptateur de protocoles capable d'acheminer des messages suivant des moteurs transactionnels du marché (BEA, IBM, etc.). |

### *Routage de messages*

**Tableau 21.5** — Les critères techniques du routage SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I-R1 | Règles de flux Explicites/ Implicites | Fourniture d'un moteur capable de prendre en charge différentes typologies de règles de routage des messages incluant l'utilisation de l'en-tête ou du contenu du message. |
| I-R2 | Garantie de délivrance | Fourniture d'un moteur garantissant la livraison et l'acheminement des messages. |
| I-R3 | Publication/Abonnement | Fourniture d'un moteur proposant un service de souscription d'événements et de leur propagation. (cf. support de WS-Notification[1]). |
| I-R4 | Optimisation de messages | Support de MTOM et/ou d'un accélérateur XML. |
| I-R5 | GRID | Fourniture d'un moteur assurant l'utilisation d'infrastructures parallèles. |

1. WS-Notification au sein de l'OASIS semble rallier les éditeurs pour amorcer la standardisation Web services des concepts de publication/abonnement évoqués au chapitre 20.

### *Transformation de messages*

**Tableau 21.6** — Les critères techniques pour la transformation en SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I-T1 | Extensibilité de messages | Utilisation interne d'un meta-language, du protocole SOAP pour l'enrichissement de messages ou la prise en compte de protocoles propriétaires ou nouveaux. |
| I-T2 | Gestion d'un dictionnaire de messages | Fourniture d'un moteur de transformation des formats de messages sur le principe d'un format pivot (souvent via XSLT). ➢ |

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I-T3 | Outils de Mapping | Mise en correspondance d'un message entrant avec un message sortant (fabrication visuelle, génération de scripts, etc.). |
| I-T4 | Convertisseur Texte Plat ↔ XML | Fourniture d'un moteur de transformation des messages ASCII en binaire. |
| I-T5 | Convertisseur Binaire ↔ XML | Fourniture d'un moteur de transformation des messages binaires en ASCII (généralement base 64). |

### *Connecteurs directs*

**Tableau 21.7** — Principaux connecteurs directs SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I-C1 | Utilisation de données du SI | Fourniture de connectivité vers des bases des données relationnelles, annuaire d'entreprise, etc. |
| I-C2 | Utilisation d'applications du SI | Fourniture de connectivité vers des progiciels métiers ou des technologies spécifiques (SAP, Oracle Applications, Java, .NETnatif…). |
| I-C3 | Utilisation de données à l'extérieur du SI | Fourniture de connectivité d'échanges inter entreprises (protocoles EDI, B2B). |
| I-C4 | Utilisation générique technique | Fourniture de connectivité standard bas niveau (HTTP, FTP, SMTP, etc.). |

### *Intégration d'information d'entreprise*

**Tableau 21.8** — Principaux critères d'une infrastructure de services d'information SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I-I1 | Extraction, Injection, Dédoublonnage | Fourniture d'un moteur capable d'analyser et de mener des transformations en masse sur des sources de données hétérogènes et réparties. |
| I-I2 | Requêteur unifié | Utilisation des langages génériques (Xpath, Xquery, Xupdate) pour la formulation de requêtes CRUD distribuées. |
| I-I3 | Constructeur de vues unifiées | Outils graphiques de fabrications de vues logiques sur plusieurs sources. |
| I-I4 | Cache persistant de données | Réplication d'informations, stockage et gestion de données ou méta-données maîtres (MDM). ➢ |

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| I-I5 | Extensibilité du cache | Utilisations transverses pour de multiples entités d'information (client, produit, RH, etc.) et prise en compte de méta-données personnalisées. |
| I-I6 | Qualité des données | Nettoyage de données non utiles, conformité de structure et de valeurs, gestion des doublons, etc. |
| I-I7 | Multiples APIs pour client de données | Utilisation des données d'entreprise restituables sous plusieurs APIs (SQL, XPath/XQuery, Web Services, JDBC, ODBC…). |

### *Registre de services*

**Tableau 21.9** — Principaux critères du registre SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| R1 | Découverte | Capacité du registre à offrir des fonctions d'annuaire UDDI. |
| R2 | Référentiel | Capacité du registre à offrir des fonctions de référentiel public de services (Classification, contrat étendu, etc.). |
| R3 | Analyse d'impact | Capacité du registre à offrir des fonctions d'analyses d'impacts du changement d'un contrat de service ou d'un SLA. |
| R4 | Publication | Capacité du registre à offrir des fonctions d'administration, notamment en matière de règles de validation et publication. |
| R5 | Contribution collaborative | Capacité du registre à offrir des outils de contribution collaboratif (gestion de versions, de configurations. modification par des intervenants différents). |

### *Sécurité*

**Tableau 21.10** — Principaux critères de la sécurité pour SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| S6 | Annuaire/Registre de sécurité | Fourniture d'un registre pour la définition de politique de sécurité. Souplesse de niveau d'applicabilité (par opération, par service, par groupements de services etc.). |
| S7 | Authentification | Fourniture d'outils pour authentifier les accédants (X509, Token, etc.). |
| S8 | Habilitations | Fourniture d'outils pour gérer les droits des accédants (suivant les spécifications XACML, WS-Authorization). ➢ |

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| S9 | Propagation du contexte de sécurité | Fourniture d'outils pour propager le contexte de sécurité des accédants entre services (suivant les spécifications SAML, WS-Trust, WS-Federation, Liberty, etc.). |
| S10 | Non répudiation | Fourniture d'outils de signature électronique pour assurer la non-répudiation des actes des utilisateurs des services (suivant la spécification XML Signature). |
| S11 | Confidentialité | Fourniture d'outils pour assurer la confidentialité des messages échangés entre services (suivant les spécifications SSL, XML Encryption, WS-SecureConversation). |

*Interfaces d'administration*

**Tableau 21.11** — Principaux critères de l'administration SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| A1 | Gestion des configurations | Fourniture d'une interface de configuration/paramétrage de la plate-forme. Cette interface permet aussi le déploiement des services (cf. SCA décrit dans la partie 4). |
| A2 | Gestion de registre | Fourniture d'une interface de gestion du registre des services (contrats métier, technique). |
| A3 | Gestion de directives | Fourniture d'une interface de gestion et contrôle des politiques de la plate-forme : politique de sécurité, politique sur les garanties d'acheminement, SLA, etc. |
| A4 | Interaction en équipes réparties | Ajustement et délégation de plusieurs rôles d'administration. Disponibilité d'un accès 100 % Web en cas d'hébergement distant. |

*Suivi de performances*

**Tableau 21.12** — Principaux critères du suivi de performance SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| P1 | Suivi opérationnel des services | Fourniture d'un agent ou mécanisme de suivi technique remontant l'état (en attente, occupé, saturé, arrêté, déficient, etc.), les performances et l'exécution des services (messages et exécution). |
| P2 | Suivi opérationnel des processus | Fourniture d'un outillage de suivi remontant l'état, les performances et les erreurs d'exécution des processus. ➢ |

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| P3 | Traces | Persistance des valeurs d'indicateurs et API d'accès aux données historisées. |
| P4 | Monitoring | Intégration avec des outils de surveillance temps réel (console Tivoli, Patrol, HP, etc.). |
| P5 | Outils de reporting, BI | Fourniture ou intégration avec des outils de génération de synthèse graphiques, tableaux, textes et d'agrégations décisionnelles.<br>Choix d'un rendu et d'un niveau de détails. |
| P6 | Suivi Métier | Fourniture d'une interface de suivi des processus, avec choix des indicateurs et alertes lisibles par les métiers remontant l'état, les performances et l'exécution de processus et la corrélation d'événements (CEP). |
| P7 | Ajustement de ressources | Mécanisme de vérifications de seuils par règles et directives permettant d'ajuster les ressources en plus ou en moins.<br>Mécanisme de régulation des demandes. |

### *Atelier de conception*

**Tableau 21.13** — Principaux critères de l'atelier de conception SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| F1 | Modèles de processus métiers | Fourniture d'un outil graphique (type BPMN) permettant de modéliser les processus métiers à exécuter sur la plate-forme SOA. |
| F2 | Modèles de compositions interactives | Fourniture d'un outil permettant de définir (puis générer) les représentations utilisateurs ainsi que leur cinématique d'enchaînement (en particulier, dans le cadre de Workflow). Il peut comporter un éditeur textuel ou graphique du langage de représentation graphique (type XML, XHTML, XAML, XUL…). |
| F3 | Modèles de services | Intégration ou interfaçage avec un outil UML. |
| F4 | Modèles d'architecture | Fourniture d'un outil graphique permettant de modéliser l'architecture globale de l'environnement SOA de l'entreprise. |
| F5 | Modèles d'infrastructure | Fourniture d'un outil graphique permettant de modéliser les infrastructures physiques qui sous-tendent l'architecture SOA (machines serveurs et environnement réseau, comme des DMZ[1]). |
| F6 | Modèle de traitements | Intégration ou interfaçage avec un outil UML. ➢ |

1. DMZ signifie Zone Démilitarisée : il s'agir de zones réseaux spécifiques, servant à héberger des serveurs semi-protégés.

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| F7 | Modélisation de données et de vues sur les données | Fourniture d'un outil graphique permettant de travailler sur les modèles de données, en particulier dans le cadre de mise en œuvre de services CRUD (outillage EII/MDM). |
| F8 | Simulation de processus | Capacité à simuler l'exécution de processus métier pour évaluer l'impact d'une modification (performance notamment). |

### *Atelier de fabrication*

**Tableau 21.14** — Principaux critères de l'atelier de fabrication SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| F9 | Composition de services | Fourniture d'un outil de composition de services supportant ou non la norme SCA. |
| F10 | Développement de composants | Intégration ou interfaçage avec un atelier de développement de code (Java, C#…). |
| F11 | Génération automatique du code | Fourniture de générateurs du code à partir de modèles (UML ou XML). |
| F12 | Fourniture de générateur de documentation | Fourniture de générateurs du documentation à partir des descripteurs de contrats (fiches de services, catalogue de services, indicateurs de processus, guide d'utilisation des services à l'usage du consommateur, guide de sécurisation, guide de déploiement à l'usage des équipes d'intégration, guide de supervision métier (SLA), destiné aux maîtrises d'ouvrage, etc.). |
| F13 | Support des Règles de coordination | Capacité de développement de règles IF…THEN…ELSE. |

### *Portails Composites*[1]

**Tableau 21.15** — Principaux critères de l'interface de présentation SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| M1 | Composition IHM | Fourniture de composants de présentation des services sous forme d'IHM composite.<br>Il s'agit généralement d'un portail compatible avec WSRP.<br>L'interface composite est alors un agrégat de « Portlets », fragments d'interface du portail. ➢ |

1. Critères uniquement spécifiques à SOA.

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| M2 | Multi-Canal | Fourniture de composants de présentation pour divers types de terminaux (PC, PDA, téléphone, etc.). ces composants pourront générer des contenus formats du standard XHTML ou Xforms. |
| M3 | Support de composition Web2 | Fourniture de composants graphiques Ajax, support des formats RSS et outil de conception visuelle pour l'aggrégation de flux XML, support du protocole ATOM. Support des widgets UWA. |

*Périmètre métier*

**Tableau 21.16** — Principaux critères de couverture métier SOA

| N° | Intitulé du critère | Commentaires |
|---|---|---|
| M3 | Gestion d'informations d'Entreprise (ECM[1]) | Progiciels SOBA de gestion de contenu, de documents, de connaissances. |
| M4 | Gestion de données internes d'entreprise (GRH, GRC, Gestion Projet, ERP…) | Progiciels SOBA de gestion des ressources humaines, de la relation client, des produits, des projets, etc. |
| M5 | Gestion de données externes d'entreprise (SCM, eCommerce…) | Progiciels SOBA de gestion des commandes, de la chaîne de sous-traitance, etc. |

1. ECM – *Enterprise Content Management* est la terminologie anglo-saxonne usuelle.

## En résumé

Ce chapitre a donné une vision de premier niveau de la démarche de choix d'outil dans le cadre de la mise en œuvre d'une plate-forme SOA.

Les critères proposés ici sont un point de départ pour une analyse d'ensemble. Ces grilles de choix devront être adaptées à chaque démarche SOA.

Les critères aussi bien organisationnels, financiers et techniques devront être détaillés pour permettre une comparaison globale juste et pertinente.

# 22

# Tous vers SOA !

## Objectif

La communauté des prétendants est immense. Ils sont issus des brokers Corba, de l'EAI ou de l'ETL, du monde des applications métiers, de la gestion d'informations au sens large, voire même du poste client, de la sécurité ou de la gestion de la performance ou même tout simplement encore du rachat de logiciels.

C'est aujourd'hui l'ensemble de la communauté des éditeurs, qu'ils soient à caractère commercial ou libre, qui s'intéresse donc à fournir tout ou partie des briques nécessaires à la plate-forme SOA précédemment décrite.

Ceci remet en cause progressivement l'ensemble des outils de réalisation et de production en place dans les SI et les équipes informatiques mais apporte aussi de nouveaux outils pour les métiers. Une nouvelle typologie semble se dégager progressivement à l'échelle du marché.

## 22.1 LES COMMUNAUTÉS RIVALISANT

Un livre entier ne suffirait pas à décrire l'offre de chaque éditeur. C'est pourquoi n'est présenté ici qu'un bref aperçu des acteurs les plus fréquemment rencontrés. Le détail de chaque offre n'est pas fourni du fait de changements encore trop fréquents dans la stratégie et le packaging des fournisseurs logiciels et/ou matériels.

L'illustration de la figure 22.1 synthétise les principaux acteurs du moment en distinguant cinq catégories principales :

- Les vendeurs « **généralistes** » : Il s'agit d'offres à caractère commercial mais ne constituant pas, loin sans faut, l'intégralité du portefeuille du fournisseur logiciel. On retrouve principalement ici IBM, Microsoft ou Sun.
- Les vendeurs « **spécialistes** » : Bien que toujours à caractère commercial, ces solutions constituent cette fois la grande majorité du portefeuille d'un spécialiste logiciel, ou du moins le cœur de sa stratégie logicielle. Ce sont, pour la plupart, des acteurs historiques ayant su anticiper sur l'enrichissement de leur produit d'infrastructure d'intégration respectif. Ils sont issus du monde du transactionnel et des serveurs applicatifs, du MOM événementiel ou de l'EAI tels que BEA, TIBCO ou WebMethods.

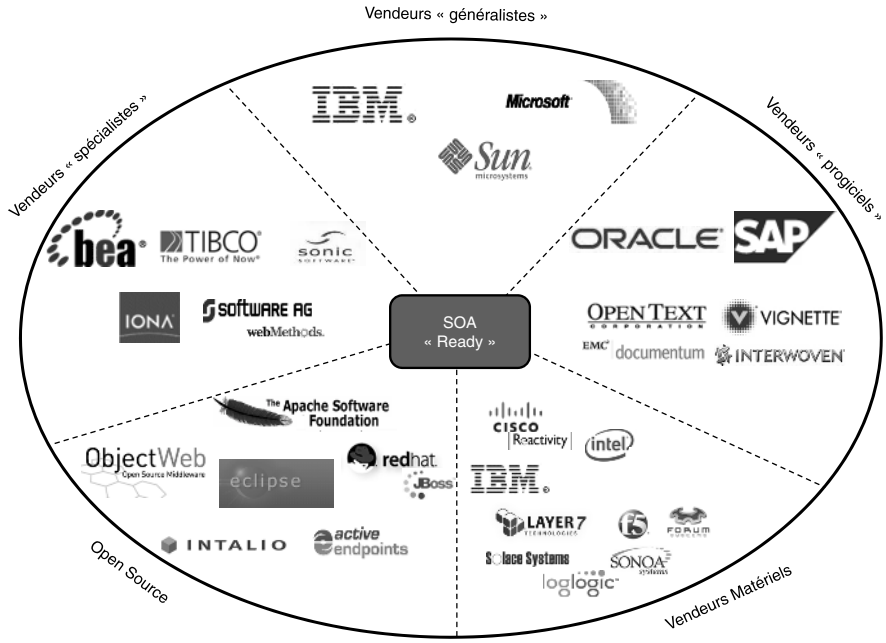



**Figure 22.1** — Les principaux acteurs du marché

- Les éditeurs de **progiciels** : Il s'agit principalement des éditeurs de progiciels applicatifs comme SAP ou Oracle. Se retrouvent également dans cette catégorie les grands acteurs de la gestion de contenu au niveau entreprise (EMC-Documentum, OpenText ou Interwoven).
- Les communautés « **Open Source** » : Avec l'essor des normes et standard (cf. partie 4), des communautés reconnues de développement de logiciels libres abritent désormais de nombreux sous-projets visant notamment certaines briques centrales d'infrastructure SOA comme le bus, l'orchestrateur ou la

chaîne de fabrication logicielle. Se distinguent notamment les communautés Apache, Eclipse, ObjectWeb et RedHat.

- Les constructeurs « **matériels** » : La recherche de l'optimisation des performances est un axe faisant aussi émerger un ensemble d'acteurs proposant directement des boîtiers matériels en guise de modules pour une plate-forme SOA. Cela couvre notamment l'accélération de traitements XML, XSLT, XQuery ou protocoles WS-*, le routage et la transformation de messages de services (bus de services), la sécurité et le respect de directives, mais aussi le suivi des services et jusqu'au contrôle du flux amont (messages, ou événements) des applications clientes. Ce marché se consolide actuellement via de nombreux rachats significatifs comme celui de Datapower (par IBM), Reactivity (par Cisco), Sarvega et Conformative (par Intel), ou d'autres restant encore spécialistes indépendants et souvent pionniers, à l'instar de Layer 7 Technologies, F5 Networks, Forum Systems, Solace Systems, Sonoa Systems, LogLogic, Xambala…

On notera que de nombreux fournisseurs SOA commerciaux trouvent aujourd'hui intérêt à investir plus ou moins dans l'open source, soit pour revendre des distributions plus avancées basées sur souche open source (exemple IBM), soit pour fournir des distributions en double licences (exemple Active EndPoints), soit enfin pour fournir des distributions avec assemblage, support et maintenance propre (exemple RedHat, Iona, etc.). Si le modèle économique open source a le mérite de réduire le coût initial d'acquisition, il ne réduit pas toujours pour autant le coût total. Il peut même se révéler moins économique à long terme pour certaines organisations. Il convient donc dans ce processus de sélection de plates-formes de ne pas faire de raccourcis trop hâtifs.

### 22.1.1 Le camp des vendeurs SOA généralistes

#### *IBM*

Classé (selon les analystes) premier ou second éditeur de logiciels à l'échelle mondiale, IBM n'annonce pas moins de 30 produits pour SOA répartis sur quatre gammes : Lotus, WebSphere, Information Management et Tivoli. L'éditeur est par ailleurs capable de commercialiser des solutions globales : matériels, logiciels, services, conseil et financement.

Au sein de la gamme WebSphere, se trouve principalement WebSphere Business Integration qui regroupe la famille des produits commerciaux de la plate-forme SOA IBM, basés sur socle d'édition Eclipse et socle d'exécution du serveur applicatif WebSphere (« WAS »)[1]. On y retrouve notamment :

1. WAS – « *WebSphere Application Server* » dispose d'extensions (Features Packs) permettant le support de fonctionnalités comme EJB3 SCA/SDO, les protocoles WS-* et Web2 sans attendre le support intégral d'une nouvelle version JEE.

- « *WBM – WebSphere Business Modeler* » est l'atelier de modélisation des processus et règles métiers. Il permet aussi la définition fonctionnelle d'écrans, formulaires ou indicateurs de performance. Il apporte en outre, des moyens de simuler et d'optimiser les processus, via la remontée de mesures capturées à l'exécution (cf. plus bas « WBM »). L'outil permet l'export XPDL2 et BPEL2 pour alimenter d'autres chaînes (par exemple le moteur de workflow « *FileNet* » ou l'environnement pour développeurs « *WID* »). La version étendue « *WebSphere Publishing Server* » fournit quant à elle, un portail Web collaboratif pour réunir plusieurs rôles échangeant autour de la modélisation métier.
- « *WID – WebSphere Integration Developer* » est l'atelier dédié à la construction de solutions métiers SOA. Il intègre en outre la reprise des conceptions métiers issues de WBM, selon des choix techniques, notamment génération d'écrans WorkPlace, JSF ou Portlets, prise en compte d'extensions BPEL2 spécifiques pour les tâches humaines, la gestion de contexte, etc.). Il permet d'effectuer l'assemblage des composants au travers d'outils d'édition graphique assistée (XML, XSD, WSDL, SCA, etc.) en vue du déploiement sur les serveurs BPM ou ESB de l'éditeur (cf. plus bas). WID peut, le cas échéant, proposer un cycle de conception itératif en approche descendante (par analyse de différences et fusion) ainsi qu'une traçabilité d'impacts potentiels sur la conception métier initiale, en cas d'approche remontante.
- « *WebSphere ESB* »[1] est le socle ESB Java/SCA/SDO tandis que WebSphere Message Broker constitue un ESB alternatif basé sur ESQL et le middleware historique de l'éditeur. Ces deux déclinaisons disposent de mécanismes de clustering et d'exploitation différents du fait de leur socle WAS et MQ respectif. En outre ils bénéficient désormais de « *WTX – WebSphere Transformation eXtender* », issu du rachat Ascential, permettant de produire visuellement des transformations complexes et de les exposer en services. On notera que IBM, à l'instar d'autres éditeurs commerciaux comme BEA ou Oracle, mise sur la norme SCA et non sur JBI considéré comme trop restrictif à Java. Le recours possible à l'environnement de développement Rational permet de fluidifier la fabrication de services Java, notamment au travers d'ateliers dédiés à l'architecture, la conception et à la génération de squelettes des services.
- « *WSRR – WebSphere Service Registry and Repository* », issu du rachat BuildForge et porté sur socle WAS, fournit un référentiel et un registre de services. Il gère le stockage des métadonnées de services, leur classification et jusqu'au cycle de vie via une machine à états permettant de maintenir un statut et un processus de validation. Il peut être utilisé directement depuis Eclipse pour de l'analyse d'impacts, ainsi qu'à l'exécution avec WESB pour effectuer du routage dynamique par genération java. Il peut aussir servir au monitoring pour des métadonnées dynamiques, tel un temps de réponse, exploitable ensuite via Tivoli ITCAM for SOA (cf. plus bas).

1. À noter qu'il existe aussi une offre ESB « WebSphere Message Broker » sur socle WebSphere MQ en lieu et place de WAS, ainsi qu'une offre ESB matérielle sur système hardware XI50.

- « *WebSphere Process server* » constitue l'environnement serveur d'exécution BPM clusterisable et distribuable de Big Blue. Construit sur WESB, il fournit désormais plusieurs types de composantes sous forme SCA (BPEL/WS-BPEL, tâches humaines, machine à états, moteur de règles). L'ensemble peut en outre s'appuyer sur des services d'informations métiers (maps SDO), une sélection dynamique des services ainsi que « *WebSphere Virtual Member Manager* » pour implémenter un service organisation (cf. Chapitre 9 – Router les processus).
- « *WebSphere Business Monitor* » possède deux déclinaisons pour le BAM incluant un serveur d'événements. L'une reste simplifiée sur seule base de tableaux de bords Web2.0, socle WAS et capture d'événements au format CBE sur JMS/MQ. L'autre permet une analyse décisionnelle beaucoup plus riche en ajoutant DB2, Cube Views, AlphaBlox ainsi que WebSphere Portal permettant un requétage interactif multidimensionnel et une personnalisation par profils.

Cette offre WBI peut encore s'étendre au-delà avec :

- « *WebSphere Adapters* » qui fournit un ensemble de connecteurs techniques gratuits (FTP, email, etc.) et de connecteurs payants notamment pour les progiciels reconnus du marché. Chaque connecteurs se veut bidirectionnel et peut communiquer avec le BAM sous forme événementielle. L'offre « *WebSphere Partner Gateway* » spécifique aux adaptateurs B2B (et nécessitant encore une intégration bas niveau « manuelle » via JMS ou MQ) devrait, à terme, rejoindre ce modèle SOA événementiel.
- « *WebSphere Business Fabric* », alias Webify s'inscrit au-delà de l'infrastructure pour fournir des frameworks de productivité métiers déclinés verticalement par secteur tel que la banque, l'assurance, la santé, les télécommunications, etc. Ce produit apporte ici les premières briques d'un édifice métier pré-packagé sur base SOA/BPM. En outre cela permet de disposer de modèles de services, modèles de processus, modèles d'indicateurs, modèles de directives, modèles de tableaux de bord, etc. Bien que récent, ce type de produits, apporte une réelle plus value sur la notion d'industrialisation de solutions métiers et rejoint désormais les initiatives de passage à SOA des grands progiciels de gestion type ERP ou CRM (SAP, Oracle, etc.)

Pour la sécurité SOA, IBM place désormais en avant son offre matérielle « DataPower ». Celle-ci, se découpe à ce jour en trois types de boitiers :

- « *XA35* » pour l'accélération XML;
- « *XS40* » pour le pare-feu XML, les Web Services, l'habilitation, l'encodage, et WS-Security;
- « *XI50* » embarque quant à lui les possibilités précédentes, et permet aussi le mixage de protocoles, la transformation any-to-any grâce à WTX et un déploiement via console Web ou Eclipse pour satisfaire des besoins de configuration ou de développement. On notera ici que les chaines de transforma-

tions WTX peuvent générer directement du micro code ou du code C++ et sont exposables comme des services.

Les produits dédiés à la présentation dans le cadre Web2/SOA/BPM ne sont pas en reste chez ce géant du logiciel. On y retrouve notamment « *WebSphere Portal* » qui demeure l'interface utilisateur pour accueillir la présentation Web Java native des processus et la prise en compte éventuelle de WSRP. IBM propose aussi un « *Mashup Starter Kit* » avec un serveur d'alimentation et d'agrégation de flux d'informations ainsi qu'un outil RAD d'assemblage baptisé « *QEDwiki* » pour la réalisation rapide de mashups d'entreprise sur base de services de contenus ou de données. Sur le socle Eclipse RCP et Lotus8, IBM reformule également son offre de collaboration, WorkPlace, et propose une déclinaison client riche composite (« *Expeditor* ») ainsi que son offre de gestion documentaire SOA (*FileNet Web Services/OpenClient*). La gamme Lotus s'étoffe aussi en outils de développement et d'exécution avec un environnement unifié d'interfaces utilisateurs composites pouvant accueillir composants NSF, composants Java/Eclipse, composants portlets JSR168, etc.

Dans sa gamme Information Management, IBM étend son portefeuille d'infrastructure pour les services d'information. Outre la proposition SDO – *Service Data Object* visant à fournir un mécanisme unifié de représentation et d'accès à des sources hétérogènes de données. L'ancien produit DB2 Information Integrator (*Websphere Federation Server*) est également intégré à WebSphere afin de fournir des services d'accès en temps réel aux sources de données de type EII. Deux briques MDM (issues du rachat successif de Trigo et DWL) apportent respectivement la structuration de produits (*WebSphere Product Center*) et de clients (*WebSphere Customer Center*) malgré des philosophies d'utilisation non harmonisées à ce jour. L'éditeur dispose aussi d'une mécanique ETL robuste et de connecteurs B2B/EDI (issu de l'acquisition Ascential) ainsi que d'une offre de services d'analyse décisionnelle via le récent rachat de Cognos.

Enfin, la gamme Tivoli fournit les outils de surveillance bas niveau, la gestion d'identité (« TIM[1] » complété du rachat MetaMerge), la sécurité des Web Services (« TAM[2] »), et le monitoring d'applications composites (ITCAM[3]).

### *Microsoft*

La firme de Redmond n'est plus à présenter. Depuis longtemps reconnu pour ses logiciels sur le poste client, elle continue à se valoriser sur le serveur et apparaît de plus en plus complète, notamment dans le cadre de l'évolution de sa technologie .NET. Son socle technique, depuis sa version 3, s'enrichit pour SOA, et dispose désormais d'un mécanisme unifié de communication baptisé *Windows Communication Foundation – WCF* ainsi que d'un moteur de workflow baptisé « *Windows Workflow Foundation – WWF* ».

1. *Tivoli Identity Manager.*
2. *Tivoli Access Manager.*
3. *Tivoli Composite Application Manager for SOA.*

- WCF fournit un modèle programmatique permettant de découpler l'implémentation des choix de déploiement de services (choix de protocoles http ou tcp, choix de format d'encodage binaire ou soap, choix de ports, niveau de sécurité, mode transactionnel, asynchronisme, etc.). Il harmonise l'ensemble des technologies Microsoft (COM, DCOM, MSMQ, Enterprise Services, .NETRemoting, Web Services, REST XML), notamment en monde distribué. Il utilise les capacités d'annotations du langage .NET en proposant une externalisation et un découpage du contrat de base et du contrat étendu. WCF peut aussi générer des traces WMI en évènementiel, appliquer par AOP des pre/post-traitements transversaux (behaviors) à différents niveaux de granularité, et dispose d'une console technique baptisé « *TraceViewer* ».
- Le moteur WWF, également intégré au framework souche, fournit quant à lui, un modèle programmatique et un serveur d'exécution de workflows applicatifs.

Au-delà et dans l'optique d'architecture nécessitant la prise en compte d'un médiateur devant gérer une variété de transformations de messages et de routages, Microsoft propose son produit Biztalk Server, entièrement réalisé en interne, et actuellement dans sa cinquième version (2006 R2).

- Celui-ci peut désormais bénéficier de nombreux connecteurs réécrits selon le mécanisme WCF, et directement packagé au sein de Biztalk, notamment pour des échanges techniques (type fichier, mail etc.) mais aussi pour les formats des progiciels du marché et pour le B2B dont la norme AS2 pour l'EDI sur Internet.
- Biztalk offre une plate-forme complète avec un orchestrateur de processus basé sur WWF et pouvant tirer parti d'un moteur de règles techniques paramétrables (« BRE ») ainsi qu'un outillage visuel de modélisation de processus et une gestion de formulaires pouvant s'appuyer sur le produit de conception graphique dédié MSInfoPath et sur le portail MOSS 2007.
- L'offre comprend en outre, un serveur d'intégration orienté messages (basé sur MSMQ) avec transformation des données. Ce serveur s'appuie sur des pipelines prédéfinis facilitant la réalisation de règles de transformations et de routage de données XML. Le référentiel des données ou des messages peut s'appuyer en outre sur les nouvelles capacités de réplications et de gestion de files d'attente ou d'intégration par messages offertes par « SQL Server 2005 SSB ».
- L'éditeur dispose aussi de réponses sur les besoins d'administration et exploitation unifiée au travers d'une cartouche pour « *Microsoft Operation Manager* ». Malgré une dépendance certaine à la base de données SQL Server pour le stockage des messages, Microsoft offre aussi un module BAM au travers de SQL Server 2005 « *Integration Services* » pour la collecte d'indicateurs, et d'extensions OLAP « *Analysis Services* » ou de rapports « *Reporting Services* » directement exploitables dans MSExcel ou au travers de portlets « WebParts MSSharePoint ». Ce BAM dispose d'intercepteurs configurables permettant notamment d'étendre le contrôle de directives de services via des produits tiers

spécialisés (par exemple AmberPoint ou SOA Software). Enfin le récent rachat de Stratature, laisse entrevoir l'émergence progressive d'une solution MDM à part entière, notamment pour l'analyse consolidée de données. Ce produit pourra aussi compléter judicieusement l'offre de progiciels Office Business.

- Basé actuellement sus le registre Systinet, Microsoft vise l'implémentation d'un référentiel de services, processus et données ainsi qu'un suivi des ressources IT au sein de la prochaine version de son système d'exploitation Windows. Ceci permettra d'en faire tirer profit à l'ensemble des applications serveurs écrites avec MS .NET, ainsi que les prochaines versions de son offre collaborative MOSS2007 ou de son offre progiciel de gestion, mais aussi Biztalk pour l'historisation des traitements et transactions sur messages.

Microsoft ne présente donc pas Biztalk comme une réponse à l'ESB, mais fournit pour cela un référentiel de pratiques architecturales baptisé « ESB – Guidance » sous la forme d'une communauté open source, affichant clairement autour de .NET et Biztalk, une volonté nouvelle d'interopérabilité multi-technologiques vers PHP, Java[1], etc.

Sur le plan du développement, et déjà à l'initiative de SOAP, le géant du logiciel poursuit une volonté affichée d'un usage répandu du méta langage XML dans tous ces produits (cf. GXA) et implémente le support étendu de Web Services via WSE. Au travers de Dynamic Systems Initiative, l'éditeur soutient aussi SML dans le but de décrire, mesurer dynamiquement et contrôler le contrat établi sur les composants d'une infrastructure. Microsoft investit également sur la productivité des équipes de conception, développement, tests et déploiement, notamment au travers des extensions apportant une intégration très forte avec Visual Studio Team System, sa chaîne de fabrication logicielle, et l'enrichissement progressif de designers de DSL[2] dédiés à chaque rôle dans le but d'industrialiser la conception SOA.

Microsoft n'est pas en reste sur la présentation de services, notamment avec la montée en puissance de Silverlight faisant converger approche Web2, RIA et 3D mais aussi d'outils de fabrication rapide, graphique, et hébergé de mashups composites comme PopFly.

Au-delà des produits de cette plate-forme SOA, Microsoft a déjà prévu d'utiliser les nouvelles moutures plus modulaires d'Office et des Smarts Documents pour combiner approche Service et mode hébergé via ce que l'éditeur démarque en « Software + Service » ou « S+S »[3]. Cela permettra d'une part d'envisager l'usage de multiples outils Microsoft directement exécuté et packagé à distance, mais aussi de combiner de manière transparente les avantages d'interfaces riches, du Web et de la puissance de serveurs hébergés. On peut déjà entrevoir de consommer un simple service de messagerie combiné à un service de base de données, jusqu'à un service final de type CRM.

1. On y découvre notamment l'existence de JNBridge pour envisager la connectivité de Biztalk avec un serveur JMS du marché.
2. Voir à ce titre, l'arrivée de produits tiers comme SOA Factory chez Avanade.
3. Voir par exemple Microsoft Live Services ou Online Services from Microsoft (Exchange Hosted Services, Microsoft Biztalk Services, etc.).

### SUN

L'offre est issue du rachat de Seebeyond et transforme une nouvelle fois le portefeuille de logiciels détenu par SUN.

Malgré l'homogénéité actuelle de la suite rachetée, rebaptisée « *JCAPS* » (*Java Composite Application Platform Suite*), l'éditeur californien ne cache plus son ambition en matière open source. Il a d'ailleurs fourni lui-même la première implémentation JBI[1] OpenSource baptisée « OpenESB ». L'environnement *NetBeans*, rival déclaré de l'IDE *Eclipse*, et notamment apprécié pour sa productivité avec le serveur d'application JEE de Sun *JSAS*, fait également parti de cette offre pour en constituer le socle.

SUN préserve donc dans JCAPS et de manière commerciale l'ESB *eGate* incluant un référentiel et un registre de services UDDI, le BPM *eInsight* capable de réalisé des invocations BPEL de web services ou de composants Java ou EJB et muni d'un modeleur BPMN, ainsi que des facilités pour le B2B (*eWay Adaptors, eXchange Integrator et eXpressway Integrator*) issus de ce rachat. La suite est supportée sur plusieurs systèmes d'exploitation (Solaris, Windows, autres Unix et Linux).

Peuvent aussi s'y ajouter l'atelier de supervision BAM *eBAM Studio*, le portail *JSPS* et un outil de composition d'interface *CWI* ainsi qu'un gestionnaire de références croisées *eView Studio* orienté client. En termes d'infrastructure, SUN propose *JSDS* un annuaire reconnu sur le marché, *JSAM* pour la gestion d'accès et la fédération d'identité selon Liberty Alliance ainsi que *Identity Manager* pour le contrôle de directives et l'approvisionnement d'identités. On notera également *eTL* pour la synchronisation de données.

SUN distribue avec JCAPS un environnement unifié pour la conception de compositions, baptisé *Enterprise Designer*. Ce dernier regroupent dans un même outil, les objets de présentation (écrans et formulaires, flot de navigation, feuilles de styles, etc.), les services quelque soit leur granularité et les processus métiers. Enterprise Designer propose aussi l'accès à la configuration dans les différents modules de JCAPS ainsi qu'un mappeur de collaboration permettant de réaliser des intégrations entre systèmes basées sur des objets communs (format pivot) reliés via des fonctions (règles métiers écrites par un profil technique) et sur lesquels pourront se baser les services métiers puis les services composites ou les orchestrations en processus métiers. Il permet enfin, par ce référentiel centralisé, la gestion de projets et celle du déploiement multi-environnements (tests, production, etc.). Une console Web baptisée *Enterprise Manager* apporte un suivi et une administration SAM également centralisée pour les plateformes d'exécution, le suivi des SLA sur les messages de services ainsi qu'un suivi graphique de l'exécution des processus métiers. L'éditeur fournit enfin des frameworks accélérateurs de SOA, notamment *Common Service Framework – CSF* pour le suivi des services et *Service Governance Framework – SGF* pour le

1. Pour faire très bref, on dira que JBI vise à standardiser la messagerie interne de l'ESB, telle que la figure 18.3 l'illustre, et surtout la façon dont un module ESB (traducteur, routeur, connecteur…) dialogue sur cette messagerie interne.

contrôle de conformités à des directives. Du côté des moteurs graphiques, SUN propose une alternative open source à Microsoft Silverlight et Adobe Flex baptisé JavaFX, offrant un même langage de scripts disponible aussi bien pour des postes clients riches, clients Web ou Mobiles consommateurs de services Web2 ou SOA.

### 22.1.2 Des spécialistes SOA vendeurs d'infrastructure[1]

#### *BEA*

Acteur depuis seulement 1995, la réputation de BEA n'est plus à faire dans le monde des middlewares (moniteur transactionnel Tuxedo racheté à Novell, MOM JMS, serveur d'application, broker d'intégration, etc.). Faisant suite à son approche de frameworks d'applications via la gamme « WebLogic », BEA a lancé depuis près de trois ans, une nouvelle gamme baptisée « *AquaLogic* » et dédiée à l'approche SOA. Aqualogic souhaite aujourd'hui couvrir un périmètre de bout en bout (baptisé « *entreprise 360°* ») pour une infrastructure globale destinée à des applications composites transverses et multi-technologiques dans l'entreprise :

- Présentations composites :
  - « *ALUI – AquaLogic User Interaction* » est une plate-forme (issue du rachat Plumtree) pour la réalisation, l'exécution et l'administration d'applications de type portail, Widgets Ajax et Web flow avec contexte de composition.
  - « *ESC – Enterprise Social Computing* » est une plate-forme pour la réalisation, l'exécution et le suivi d'applications de type mashups Web2 pour les besoins d'utilisateurs finaux et de développeurs de Widgets (il permet en outre le mixage rapide de flux RSS, WS-* avec un outil de recherche collaborative et un mécanisme de sécurisation et de statistiques sur ces flux).
- Métiers composites : destinés à servir de plate-forme d'intégration pour les processus et services métiers, « *AL BPM – AquaLogic Business Process Management* », issu du rachat de Fuego, est un orchestrateur de processus automatisés et interactifs aujourd'hui pleinement intégré au bus de services « *ALSB – AquaLogic Service Bus* ». Ces deux serveurs peuvent en outre respectivement bénéficier de consoles BAM et SAM (cf. *Workspace 360°*). Deux ateliers de modélisation complémentaires, sur socle Eclipse, séparent d'une part, l'activité d'analyse et optimisation métier des processus, et d'autre part l'intégration de services aux processus. À noter également, « WebLogic Event Server », permettant de faire de la corrélation statistique d'évènements afin de remonter des évènements « synthétiques » pour la console BAM. Les tableaux de bords BAM peuvent être exposés à travers une présentation composite.
- Données composites et Services composites : « ALDS – *AquaLogic Data Services* » *et* « *WLI – WebLogic Integration* » fournissent l'infrastructure SOA pour les équipes techniques. Ils permettent respectivement la réalisation, l'exécu-

1. Cette catégorie étant vaste, le livre ne peut retenir certains éditeurs « ultra spécialistes » actuellement en pointe sur des modules particuliers tel que Amberpoint, HP/Systinet ou SOA Software pour la gouvernance, Intalio sur le BPM, ou Orchestra Networks sur le MDM.

tion, l'administration et la sécurisation de services de données (cf. EII) ou l'orchestration de services de traitements, via des connecteurs pour les protocoles techniques standards et les systèmes patrimoniaux. Ce socle respecte désormais la spécification SDO.

- Déploiement et virtualisation.

Dans cette vision, BEA mise désormais au-delà de JBI (Java) et propose déjà le support de SCA pour l'interopérabilité multi-language et le choix tardif du mode de déploiement. Un nouveau module (« *Service Assembly Manager* ») doit d'ailleurs y être dédié. A noter dans ce même cadre, l'initiative « mSA – micro Service Architecture » ainsi que l'essor de produits de virtualisation (« Barre Metal ») intégrant la JVM et le serveur Java de BEA.

Par ailleurs, BEA s'efforce d'enrichir une usine logicielle pour couvrir l'ensemble du cycle de vie d'applications composites. Cette initiative, baptisée « Workspace 360° », vise un environnement unifié et collaboratif adressant à la fois, les analystes métiers, les architectes, les développeurs et jusqu'aux exploitants et responsables d'applications. Bien que les ateliers frontaux soient basés sur socle Eclipse open source, ils restent soumis à licences payantes contrairement à d'autres fournisseurs concurrents. Les consoles d'administration sont également unifiées sur base des fonctionnalités disponibles dans les modules de présentation composite.

Cette vision à 360° permet donc de couvrir des préoccupations d'analyse et d'optimisation, de conception (réutilisation, dépendances, configuration, etc.), de réalisation (implémentation et intégration) mais aussi de production (suivi) et de pilotage (retour sur utilisation, etc.) autour d'un référentiel central baptisé *« ALER (AquaLogic Enterprise Repository) »*. Ce dernier, issu du rachat de Flashline par BEA, permet une gestion électronique du cycle des artefacts de composition (contrats de services, directives de conception etc.) avec de nombreuses ouvertures pour une consultation ou une publication pour la panoplie d'outils présents dans l'entreprise sur le développement, le test, la gestion de configuration, la gestion documentaire, le registre d'exécution des services[1] etc. En matière de gouvernance des services, BEA étend désormais cette gamme en fournissant « ALES (*Aqualogic Enterprise Security*) » pour servir de console centrale d'autorisation, de distribution des directives et de vérification de leur déploiement. Enfin, « *ALSM – Aqualogic Service Manager* » permet le suivi technique de la mise en service (SAM) et fourni une cartographie du trafic à l'aide d'agents par technologie de messages.

BEA participe très activement par ailleurs, aux côtés de Microsoft et d'IBM, à la définition des standards dédiés à la mise en œuvre des Web Services ainsi qu'aux efforts de normalisation pour SOA en contexte hétérogènes de technologies (SCA, etc.).

À noter enfin, que l'infrastructure Aqualogic peut aussi s'intégrer avec un registre tiers comme celui de Systinet (Mercury) ou un BAM spécialisé comme celui de AmberPoint ou un MDM orienté client comme celui de Purisma.

1. BEA propose d'ailleurs un OEM baptisé ALSR (*Aqualogic Service Registry*) fournissant un registre de services notamment pour le contrôle de directives à l'exécution.

### Tibco

Est-il besoin de rappeler la signification du nom TIBCO (*The Information Bus COmpany*) ? Cet acteur historique du monde de la finance grâce à son robuste middleware asynchrone orienté publication/abonnement « RendezVous », reste aujourd'hui un *pure-player* très présent.

L'EAI historique de l'éditeur s'est vu initialement étendu d'un conteneur Web Service « BusinessWorks ». Cette dualité s'estompe pour laisser place à un véritable ESB unifiant la transformation et le routage de messages ou d'évènements. Les connecteurs directs EAI/B2B font également la forte valeur ajoutée de BusinessWorks. Un module registre « *ActiveMatrix Registry* » propose la solution Systinet en OEM mais d'autres annuaires restent possibles, tandis que *ActiveMatrix PolicyManager* propose une traçabilité des entrants/sortants ainsi que des fonctionnalités de cryptage pour la sécurisation des services. Le module « XML-Canon » permet l'import/export XML de tout composant du référentiel opérationnel d'exécution et assure une organisation cohérente et sécurisée des données, des services et des processus.

Le module d'orchestration « iProcess » est désormais intégré à BusinessWorks et dispose d'une très riche supervision métier BAM bientôt capable de proposer des assistants pour accélérer le choix d'indicateurs notamment grâce à une sélection de type d'entrants selon la performance des services par consultation directe de ActiveMatrix. Tibco BAM peut aussi bénéficier d'un gestionnaire d'événements métiers complexes *BusinessEvents* proposant un cache distribué pour la corrélation en mémoire d'une très grande quantité d'événements. iProcess supporte la gestion de transactions et de processus longs et bénéficie de « *TBR* » comme moteur de règles. Tibco propose gratuitement BusinessStudio comme outil de modélisation métier sur socle Eclipse avec support de notation BPMN, import/export XPDL et prochainement BPDL permettant de combiner processus et règles. Il permet aussi des simulations à destination de maitrise d'ouvrage et un affinage de conception par un architecte réalisant le câblage des processus sur les services par interrogation d'un annuaire UDDI ou du bus BusinessWorks. BusinessStudio permet un déploiement direct dans iProcess et Tibco Asset Central qui apporte un versionning au niveau fichiers (XPDL, WSDL…) d'un même projet.

« *Collaborative Information Manager* » est la réponse MDM de Tibco. Issu de l'acquisition de Velosel, initialement dédié à un référentiel produit, il peut désormais jouer le rôle d'un meta-catalogue et tire parti d'une intégration avec BusinessStudio, BusinessEvents et le B2B. TIBCO dispose aussi d'une solution ETL « DataExchange ».

L'éditeur propose la fabrication de portails avec « PortalBuilder » mais aussi un environnement de composition RIA « General Interface » avec un environnement intégré de développements (IDE) open source[1], des composants Ajax sur étagères et un Bus Ajax qui permet une communication multiplexée, filtrable et voire streamée pour pousser des flux de données vers la couche de présentation accédant aux services. Ce bus de présentation permet en outre de corréler les évènements de la cou-

1. Voir également la communauté en ligne Tibco Developer Network.

che cliente et ceux de la couche serveur afin de produire des tableaux de bord temps réel sur de gros volumes d'informations par exemple.

Enfin, Tibco amorce également « ActiveMatrix Service Grid » pour une virtualisation de services dans un seul conteneur pouvant accueillir des services composants SCA et JBI distribuables en multi-technologies avec gestion de leur disponibilité, sécurité et une administration centralisée

### *Progress/Sonic Software*

Désormais filiale de Progress Software, Sonic Software est le premier acteur à avoir revendiqué une solution en topologie distribuée. Le cœur de transport « Sonic MQ » repose sur un bus asynchrone distribué, développé en interne, sur lequel « *Sonic ESB* » propose des fonctionnalités de routage et transformations techniques de messages ainsi que d'un conteneur pour WS-*. On peut aussi y adjoindre *XML Server* pour un traitement (XSLT, XQuery) optimisé et un stockage XML natif ou *Sonic Database Service* pour accélérer la réalisation de services CRUD sur des accès SQL à des bases de données relationnelles. Le discriminant majeur de l'offre étant la souplesse de distribution en segments du bus toute en gardant des capacités de configuration et d'administration centralisées.

Bien qu'il soit possible de lui reprocher une frontière trop légère entre outils de développements et outils d'administration, ainsi qu'une dépendance au serveur applicatif fourni, Sonic dispose d'une infrastructure évolutive pour la montée en charge.

L'offre peut se compléter avec :

- *DataXtend* qui apporte une infrastructure pour concevoir et exécuter des services d'informations sur des sources de données hétérogènes. Il propose notamment un environnement de conception sur base Eclipse pour réaliser des vues logiques (par transformation, agrégation, validation et application de logiques métiers sur des données) et fournit un framework d'exécution capable d'auditer les services ainsi générés et déployés en EJB, Web Services, etc. L'ensemble peut reposer sur un socle unifié de connectivités via *DataDirect*, et apporte aussi des capacités d'accès temps réel (en lecture ou mise à jour), de synchronisation et mise en correspondance de données (type EII) ainsi que la possibilité de construire des hubs locaux de données pour des accès plus performants sur de multiples référentiels.
- *Apama*, un moteur de corrélation d'événements (type CEP) pour l'analyse et le suivi temps réels d'èvènments techniques ou métiers. Ce moteur peut s'utiliser avec *Apama BAM* qui fournit des tableaux de bords accédant aux régles du moteur ainsi qu'un environnment RAD de réalisation de consoles adhoc.

L'éditeur propose désormais également, la solution TeamWorks BPM de Lombardi en OEM qui se retrouve intégrée par Sonic Software au bus Sonic ESB. Ceci permet la réalisation d'orchestration métiers sur des systèmes fortement distribués,

tout en apportant un environnement de modélisation confortable pour des acteurs métiers.

On notera enfin l'ajout récent du portefeuille Actional au sein de Progress Software qui positionne désormais cet acteur dans les leaders de l'administration de services et du contrôle de directives de services exposés au sein du SI de l'entreprise.

L'éditeur propose enfin *OpenEdge* une plate-forme de génie logiciel unifiée pour la conception et l'exécution de solution SOA basée sur *ABL – Advanced Business Language*. Reprenant les concepts de Progress 4GL, il s'étend en outre d'un atelier dédié aux architectes, d'un Studio pour développeurs accélérant la fabrication d'interfaces utilisateurs et d'applications composites interactives sur des objets logiques, en passant par la localisation et jusqu'au déploiement, la gestion de configuration et la génération automatique de code en particulier autour des contrats et de la composition de services.

### *Software AG / WebMethods*

Acteur historique depuis 35 ans des bases de données légataire avec ADABAS et le langage NATURAL sur Mainframe, l'éditeur allemand trouve désormais un nouvel essor suite à l'acquisition de WebMethods qui vient enrichir fortement sa plateforme SOA et former ainsi une nouvelle division dédiée. Constitué au départ de son référentiel natif XML (« Tamino »), la suite d'abord baptisée « Crossvision » et dédiée notamment aux services d'informations, devient WebMethods Product Suite. Elle est fournie par défaut sur le serveur applicatif JBoss, mais reste portable sur d'autres serveurs applicatifs.

Au sein de la plate-forme, « *WebMethods BPMS* « apporte un orchestrateur BPEL de processus ainsi qu'un bus de services « *WebMethods ESB* » pour le routage, la transformation de messages et la connectivité à des référentiels patrimoniaux variés. Pour le BAM, l'éditeur met l'accent sur les indicateurs de performances et l'optimisation des processus et de l'infrastructure par analyse de patterns au sein d'un historique. Ce principe de mesures et surveillance métier se retrouve également décliné pour les échanges B2B ou SAP. La suite regroupe enfin un EII, baptisé « *Webmethods Information Integrator* », pour les vues unifiées temps réel ainsi que « *WebMethods MDM* », un MDM généraliste.

Software AG dispose par ailleurs d'un moteur de workflow ciblé pour les interactions humaines et, d'un environnement de composition Ajax ou SWT disponible en mode autonome ou plug-ins sur Eclipse. Enfin, « Legacy Integrator » permet d'ouvrir les données et applications, notamment du mainframe, en les exposant comme des services Web.

Un registre et référentiel, baptisé CentraSite, est utilisable par chaque module et sert à la fois pour l'enregistrement et le déploiement de modèles de données, de services, ou de descriptions de processus. Il constitue l'offre centrale de gouvernance SOA pour Software AG et s'accompagne d'une initiative communautaire rejointe par plusieurs autres éditeurs (Fujitsu, AmberPoint, Novell, etc.).

#### Iona

Faisant parti des leaders pour son broker Corba, Iona a cette fois choisi de parier sur un moteur fortement paramétrable par les adaptateurs de protocoles, baptisé « Artix ». En outre Iona mise sur l'implémentation facilitée de nombreux patterns d'intégrations. Une branche de développement séparée, « Celtix », basée sur un principe similaire, fait désormais partie du projet Open Source CXF (cf. Apache plus loin).

#### Encore beaucoup d'autres acteurs de niche

Force est de reconnaître, que de nombreux outils ou utilitaires sont aujourd'hui fournis par des sociétés beaucoup moins connues, notamment pour compléter la chaîne logicielle. S'il reste impossible d'en faire ici la liste exhaustive, il est classique de rencontrer par exemple ceux de Altova, Stylus Studio, Aris, Mega, MetaStorm pour la modélisation des processus et des services, ou d'autres plus ciblés comme OnMap, BlueOxide qui facilitent la cartographie ou la modélisation collaborative, etc.

On peut également citer HP pour la gestion des performances et des services.

### 22.1.3 La revanche des Solutions Métiers commerciales

#### SAP

Le leader des progiciels de gestion intégré a tracé une route faisant sa force depuis son origine, le tout « paramétrable ». Une évolution majeure le conduit actuellement à une refonte progressive de son offre selon deux axes :

- en la rendant pilotable par les processus, autrement dit en externalisant ces processus de l'implémentation des fonctionnalités de gestion ou d'accès aux données;
- et composable avec des modules métiers non-SAP.

Cette nouvelle stratégie est baptisé *Enterprise SOA*. L'ambition de cette flexibilité amorcée en 2003, doit aussi reposer sur une infrastructure que l'éditeur allemand entreprend aussi de fabriquer en interne sous le nom « Netweaver ». Il s'agit plus particulièrement, d'une infrastructure pour les métiers. Elle est composée d'un Portail, d'un système BPM et BAM ainsi que d'un cache MDM avancé (disponible séparément), même si l'éditeur n'envisage pas de rivaliser directement avec les fournisseurs d'infrastructure IT. L'utilisation de ce MDM avec cache permet d'utiliser un format pivot logique et normalisé d'informations SAP et non-SAP reposant sur des référentiels physiques distribués de données.

Au cœur, SAP fait évoluer, son serveur d'intégration initialement baptisé XI et servant à relier des modules SAP en utilisant le protocole propriétaire ALE[1], pour en faire un bus *Process Integration – PI* capable d'accueillir une variété (surtout technique) de protocoles du marché, donc en particulier des composants métiers non SAP expo-

1. ALE établit un dialogue entre modules SAP distribués et serveurs d'EDI.

sés en services. On notera néanmoins que ce bus souffre encore du manque de certaines fonctionnalités clé, comme le mécanisme de publication / abonnement.

Même s'il ne faut pas attendre la disparition de l'ABAP, cette évolution majeure offre ainsi un premier niveau d'ouverture, d'autant qu'un référentiel de services baptisé *Enterprise Services Repository – ESR* permet également une définition centralisée et cohérente des processus métiers, des services SAP ou non-SAP ainsi que des objets macroscopiques d'informations – *business objects* – accédant aux principales entités de données gérées par SAP. Ce référentiel utilise une approche MDA pour générer également le squelette des interfaces d'implémentations en ABAP Objet ou Java. À noter que la modélisation visuelle de processus s'envisage à ce jour via des outils tiers en OEM, notamment Aris IDS Sheer, rebaptisée *ARIS for SAP Netweaver capable d'accéder à ESR et fournir une définition externalisée du procesus*. De même que la gestion d'identité est proposée via la solution Siemens par exemple, ou les connecteurs directs via iWay ou Seeburger B2B.

SAP franchit aussi le pas de la modélisation visuelle d'interfaces homme machine composant des services d'entreprise et fournit un atelier baptisé *Visual Composer*, minimisant le recours à l'écriture de code pour la réalisation d'applications composites simples.

Après les premières livraisons de l'ERP « R/3 » refondu sur Netweaver, SAP poursuit en même temps la généralisation de l'usage du méta-langage XML et des normes WS-* (y compris au-dessus de ses langages de communications propriétaires IDOC, RFC et BAPI). Il généralise à ce titre l'utilisation de définition sémantiques globales des structures d'information SAP, stockées au sein de ESR, et partagées par des Web Services baptisés « Enterprise Services » minimisant ainsi les transformations de formats nécessaires pour les consommateurs de plusieurs de ces services. Même si la tendance générale est de fournir des implémentations natives ABAP et des compositions Java[1], ce découplage permet également de masquer la dualité des implémentations du fournisseur qu'elles soient ABAP et/ou Java. Dans le même temps, SAP modernise aussi l'environnement de conception des services Java via *SAP Netweaver Composition Environment* basé sur Eclipse. Enfin, *ES Workplace* permet de découvrir les solutions métiers SAP basées sur ces Enterprise Services mais aussi de construire un assemblage spécifique. SAP prépare ainsi une bascule de son modèle économique en amorçant une approche SaaS[2] avec des annonces comme sa prochaine solution baptisée *BusinessByDesign (nom de code A1S)* destinée aux entreprises de taille moyennes.

### *Oracle (PeopleSoft/Siebel)*

Outre Oracle Applications, les rachats successifs de grands progiciels tel que PeopleSoft puis Siebel engage actuellement Oracle dans un vaste chantier stratégique d'intégration et de ré-implémentation progressive de l'ensemble des périmètres fonctionnels de ce patrimoine logiciel étendu. Encore plutôt reconnu pour son SGBDR,

1. Évitant ainsi à la base installées des clients historiques SAP de trop gros efforts de migrations purement techniques.
2. SaaS Software As A Service.

cette vision baptisée « Fusion » pérennise désormais doublement le positionnement de l'éditeur sur SOA. En effet, il doit à la fois investir massivement dans l'infrastructure d'une plate-forme SOA complète (« *Fusion middleware* ») du fait de la diversité des offres à intégrer, mais doit dans le même temps, démontrer les nouveaux bénéfices d'agilité métier qui résulte d'une capacité de construction flexible et productive de multiples solutions métiers modulaires (« *Fusion Applications* »). Ces solutions sont ainsi construites sur une nouvelle architecture baptisée *AIA* fournissant des modèles métiers pivots servant de référence par métier via des schémas XSD publics et ouverts et des processus métiers BPEL pré câblés. L'éditeur envisage d'adresser également de manière pré-packagée dans AIA, une vingtaine de secteurs d'entreprise sur plusieurs années. Les premières apparaissant actuellement dans le monde des télécommunications et de la gestion de la relation clients. A noter aussi que AIA restera ouvert aux intégrateurs, partenaires et éditeurs de logiciels tiers afin d'envisager par co-développement d'autres compositions (modèles et processus) de ces applications modulaires sur des systèmes externes selon des directives et un programme formalisé par Oracle.

L'ambition affichée est forte. Il s'agit, non seulement, de rassembler ses offres de gestion ERP (Finances, CRM, RH…) mais aussi de les déployer sur la base d'une infrastructure déjà reconnus sur le marché, notamment avec :

- l'indépendance du socle serveur applicatif JEE, même si des facilités d'administration s'ajoutent en cas d'usage du serveur applicatif *Oracle AS* ;
- la généralisation de l'uage de « *Coherence* » un cache java distribué issu du rachat Tangosol offrant des capacités de traitements massivement parallèles et d'allocation dynamique de ressources (cf. *Grid 10g et Fusion Mesh 11g*) pour plusieurs modules de l'éditeurs (orchestrateur, workflow, moteurs de règles, ESB, etc) et également disponible séparemment pour des solutions à développer en spécifique, par exemple dans les domaines à forte contraintes de performances comme la finance, les télécommunicatons, les jeux, etc.
- *« BPEL Process Manager »*, un moteur d'orchestration de processus très complet (issu du rachat du spécialiste Collaxa) est capable de prendre en compte les processus longs et les processus humains. Il permet un maintien en cohérence de la définition des processus durant les allers-retours de la conception métier et technique via *BPA Suite* et l'envrionement *jDevelopper*. BPA Suite embarque en OEM plusieurs modules Aris[1] de IDS Sheer, une solution de BPA[2] leader du marché. Cet environement propose ainsi un serveur référentiel des artefacts de conception, la possibilité de définir des indicateurs métiers[3] via la mise en place de sondes pour la solution BAM, puis de remonter les mesures collectées vers la simulation.

1. Notament pour la modélisation de processus métiers en représentation EPC ou BPMN, la modélisation d'architecture, la simulation de processus et la publication de modèles.
2. BPA – *Business Performance Analysis* ou outils de modélisation et simulation.
3. KPI *Key Performance Indicator.*

- Enfin, « *Oracle ESB* », qui est un bus de services incluant une compression binaire des échanges SOAP avec Oracle BPM L'éditeur prévoit dejà le support de SCA et SDO.

À noter également une harmonisation progressive des consoles d'administration au sein de *Oracle Enterprise Manager* ainsi que des améliorations ergonomiques du fait de l'usage de composants graphiques de haut niveau (jsf), un suivi métier et technique de chaque instance de processus et la prise en compte progressive d'administration de contrats étendus jusqu'au SLA. Oracle dispose aussi d'une offre complète de gestion d'identité (issue des rachats d'Oblix, Thor, et Octet-String. Cette infrastructure se retrouve donc déjà capable de rivaliser avec les spécialistes, même si les *Fusion Applications* ne peuvent encore en tirer les pleins bénéfices. Sans surprise, la firme de Larry Ellison possède désormais plusieurs types de MDM orientés client (notamment issu du rachat SIEBEL « UCM »), mais aussi orientés produits et finance, qui pourraient faire l'objet d'une prochaine globalisation d'infrastructure de données distribuées dynamiques.

L'éditeur n'est pas en reste, sur l'approche ROA[1] dans le cadre de la présentation de services d'informations grâce d'une part à son framework *ADF* permettant de construire rapidement des mises en correspondances de composants graphiques de haut niveau (jsf) et de multiples sources d'informations de type Web Services ou bases de données. Sur cette base, Oracle ajoute déjà une nouvelle plate-forme baptisée *WebCenter* mixant une approche Web2 et portail collaboratif pour assembler et consommer des services ROA et reconstruire ainsi efficacement de multiples espaces Web d'échanges sur des informations ou services distribués en interne, sur Internet ou chez des tiers réunis selon des techniques simples et désormais éprouvées de marquage (tags) et flux RSS.

Oracle s'attaque également au marché de la gestion de contenus d'entreprise avec la refonte de ses anciennes offres (« iFS » et « Files ») et le rachat du portefeuille logiciel Stellent pour viser l'unification des contenus accessibles par services et Web Services. Cette nouvelle suite de produits « *Oracle Content Management Solutions* » servira désormais de socle pré-intégré de persistance des contenus pour la plateforme *WebCenter* du même éditeur (cf. plus haut) en proposant des interfaces orientées services d'accès au contenus via la norme JCR.

### *Les éditeurs ECM en SOA*

Les éditeurs de solutions d'*Enterprise Content Management* (ECM) cherchent à unifier les fonctionnalités de leurs différents modules de gestion de contenus (contenus documentaire, contenus web, contenus multimédia, collaboration sur les contenus, etc.). L'adoption d'une architecture SOA pour structurer les produits leur permet :

- soit une approche consistant à exposer des services autour des opérations disponibles sur le contenu (cf. Interwoven, Oracle, etc.);

1. ROA (*Resource Oriented Architecture*) désignant un e approche SOA simplifiée à base d'une interface réduite à un simple CRUD dans le but de manipulation d'informations (ou resource en anglais – cf. Partie 6).

- soit une approche de construction d'un modèle unifié encapsulant tout contenu et exposant des « nouveaux » services d'informations (cf. Vignette, BEA, IBM FileNet, EMC Documentum, etc.).

La granularité des interfaces et leur valeur ajoutée métier peuvent varier :

- soit par la fourniture de services techniques de type CRUD XML (SOAP RPC/DOC), Java ou .NET (c.f. Day, Alfresco, etc.);
- soit par l'ajout de services administratifs grâce à l'utilisation d'un moteur d'intégration des contenus, ou d'approvisionnement de sources de contenus réparties (cf. BEA, Interwoven, Oracle, Vignette, etc.) dans l'optique d'un référentiel virtuel ou consolidé;
- soit par des services organisationnels transverses de sécurité (droits d'accès front ou back-office), ou de conformité à des normes de qualités, de législations, etc. (cf. Documentum, OpenText, etc.). Il est possible d'envisager que certains CMS produisent directement des rendus de contenus composables à l'aide de langages comme XUL ou XAML.

Ainsi SOA apporte des réponses pour proposer un modèle d'architecture de mise en œuvre des concepts ECM[1] pour :

- faciliter l'accès homogène vers des référentiels hétérogènes (par cloisonnement des implémentations) = Approche « **Library** » de l'ECM;
- apporter une flexibilité à (re)assembler plusieurs sources/fonctionnalités existantes en créant plus de valeur (ex : recherche unifiée…) = Approche « **Best-of-breed** » de l'ECM;
- fournir un suivi extensible de la performance des sources et processus de contenus intégrés = Approche « **Factory** » de l'ECM.

### 22.1.4 L'essor des généralistes Open Source

#### *Apache*

La communauté Apache fournit depuis plus de trois ans une implémentation open source de référence pour l'invocation de Web services baptisée « *Axis* ». Si Axis n'a pas encore atteint le niveau d'unification des transports disponibles dans les ESB commerciaux, la nouvelle version Axis 2, incorpore un support étendu de WS-* (WSDL2, asynchronisme, support multi-transports dont JMS, génération multi-proxy C ou Java, multi-bindings dont XMLBeans, JAX-WS et WSIT[2], WS-Policy etc.). La commu-

1. Cf. les travaux de AIIM (*Association for Information and Image Management*) pour l'interopérabilité entre solutions ECM, baptisé « iECM ».
2. WSIT (*Web Services Interoperability Technologies*) permet l'interopérabilité des Web Services entre Java et WCF (le conteneur de services Microsoft intégré à .NET 3.x).

nauté à également repris Xfire (anciennement CodeHaus) et Celtix (anciennement Iona/ObjectWeb), désormais rebaptisé « *CXF* », pour automatiser le développement de services en Java à la fois sur WS-* et REST et fournir un serveur de Web Services allégé voire XML « maison ».

« *ServiceMix* », à ce jour toujours en phase d'incubation auprès de Apache, souhaite adopter la philosophie d'ESB modulaire (via JBI, configuration Spring et serveur JMS ActiveMQ embarqué) tout en supportant plusieurs conteneurs de Web Services (dont Axis, CXF, JAX-WS,…). Son stade de maturité n'est à ce jour pas encore atteint. On peut aussi y adjoindre :

- « ODE » (*Orchestration Director Engine*) orchestrateur de processus pouvant à la fois supporter le standard WS-BPEL 2.0 et les versions BPEL 1.x. Il est issu de Fivesight/*PXE* (repris aussi par Intalio). Ce projet a récemment terminé son incubation Apache. Il supporte en outre le framework WSIF.
- « *Synapse* » (contribution de Sonic) dédié à la médiation de Web Services (notamment transformation, routage ainsi que mesure et traceabilité). Synapse n'inclut pas de connecteurs ou adaptateurs de protocoles et suppose donc une focalisation Web Services (SOAP). Il peut en revanche se présenter comme composant JBI.
- « *Tuscany* » incubation Apache des spécifications OSOA[1] (i.e. SCA, SDO et DAS) déclinées pour plusieurs technologies (notamment Java, C++ et PHP) et pour laquelle des plugins Eclipse de productivité apparaissent via le projet STP.
- « Camel » framework facilitant l'unification de médiations et transformations selon des patterns d'intégration d'entreprise types.

Enfin n'oublions pas que Apache, délivre aussi « *Geronimo* », serveur applicatif Java complet et modulaire, certifié à ce jour JEE 1.5, supportant plusieurs conteneurs Web (Tomcat ou Jetty), plusieurs conteneurs Jax-WS (Axis2 et CXF) et pouvant accueillir ServiceMix, ainsi que « *ActiveMQ* » (pour le support de JMS).

L'ensemble constitue ainsi une implémentation de référence, désormais en concurrence avec d'autres suites SOA Open Source comme RedHat/Jboss+jBossESB ou Sun/Glassfish+OpenESB ou encore ObjectWeb/JonAS+Petals.

### *ObjectWeb/OW2*

Fondé en 2002 par Bull, l'Inria et France Telecom, ObjectWeb délivre aujourd'hui des solutions open source reconnues comme « Enhydra » (serveur d'application Java/XML), « JORAM » (bus à messages conforme JMS), « JonAS » (implémentation des spécifications JEE TM).

1. OSOA (*Open Service Oriented Architecture*) Collaboration est un groupe d'industriels regroupés en vue de standardiser la définition de conteneurs de services composites multi-technologies pour les traitements ou les données.

Dans le cadre du consortium OW2, la communauté ainsi élargie, a logiquement fortement misé sur JBI au travers de « Petals » initialisée par EBM WebSourcing (SSLL Toulousaine) conjointement avec une entreprise Brésilienne, Fossil E-Commerce. Cet ESB a pour ambition de prendre en compte les problématiques d'environnements d'intégration hautement distribués (via JORAM) capables de traiter de plusieurs dizaines à plusieurs centaines de containers de services, notamment dans un contexte de fournisseur d'hébergement (*Integration Service Provider*). Cette infrastructure peut en outre bénéficier du serveur de traitements parallèles (Grid) baptisé *ProActive* disponible auprès de la même communauté.

L'ESB « Petals » fournit, à ce jour, plusieurs Binding Components (notamment pour les protocoles techniques FTP, JMS, Soap, Mail, etc.) ainsi que plusieurs Service Engines (CSV, Pojo, RMI, XSLT…) mais reste néanmoins dans une approche très technique à réserver aux intégrateurs spécialisés.

### *RedHat*

À ce jour, moins avancé que les deux communautés vues précédemment, Jboss pourrait rattraper son retard par le choix de l'acquisition de Rosetta et de son rachat par RedHat. Le support sur JEE 1.5 retardé à fin 2007, proposera ainsi un serveur applicatif étendu doté de fonctionnalité de messagerie synchrone ou asynchrone unifiée par JMS mais néanmoins capable d'exploiter :

- un moteur de transformation XSLT et Smooks (Java);
- son moteur de règles (« Drools ») par programmation via assertions pour un routage dynamique basé sur le contenu;
- le moteur de transition d'états (« jBPM ») qui bénéficie déjà d'un plugins de productivité dans Eclipse;
- un registre de service Jax-R et UDDI;
- un référentiel persistant des événements.

RedHat semble aussi se rapprocher de solutions comme ActiveBPEL (cf. plus loin) pour supporter une suite plus complète incluant l'orchestration de processus.

### *D'autres spécialistes du « libre »*

On notera enfin la maturité progressive de plusieurs communautés pour certains modules de la plate-forme SOA en logiciel libre (Open Source).

- « Mule » est un bus de services Java aujourd'hui reconnu sur le marché et se présente en challenger direct des grandes communautés Open Source précédemment citées. Sa prise en main se révèle facile. Il dispose en outre d'une documentation très complète (guide utilisateur, guide du développeur, cookbook, javadoc, etc.), d'une communauté active (CodeHaus) et d'un support professionnel payant (MuleSource). Son principe de fonctionnement nécessite un serveur de messagerie JMS externe (plusieurs sont possibles ActiveMQ,

JBoss MQ, Joram OpenJms). Sa configuration s'effectue par fichiers XML sous forme de composants Java beans Spring.

- « ActiveBPEL » est un orchestrateur de processus disponible en déclinaison Java et .NET. Il inclut un modeleur également en open source. Des extensions payantes permettent de disposer de fonctionnalités avancées comme le support de WS-Security, des facilités d'administration, de suivi et de tolérance aux pannes.
- Plusieurs autres communautés délivrent par ailleurs certaines briques du puzzle, notamment pour la chaîne de fabrication décrite au chapitre 20.
  - La fondation Eclipse fournissait déjà *WTP* (*WebTools Project*) aujourd'hui intégré à *Eclipse Europa* qui propose l'outillage nécessaire au développement de services Web sur base Eclipse. Il permet en outre la création de WSDL à partir de code Java ou l'inverse et offre des outils d'aide à l'édition et la mise au point des descripteurs et des messages. « BPELDesigner » devrait apporter prochainement un éditeur BPEL.
  - Le projet STP (*SOA Tools Platform*) initialisé par la fondation Eclipse, vise à fournir les outils nécessaires à la mise en œuvre d'architectures orientées services en se conformant aux travaux SCA. Il réutilisera l'éditeur de fichiers WSDL fourni par WTP, mais ambitionne de couvrir l'ensemble du cycle de vie incluant la conception, la configuration, l'assemblage, le déploiement et la supervision.
  - Le projet ATF (*Ajax Toolkit Framework*) également en incubation au sein de Eclipse, propose un framework extensible pour enrichir des environnements de développements Web2 au dessus de différents moteurs d'exécution Ajax (comme Dojo, Zimbra, Rico, Script.aculo.us, etc.). Il offre notamment des fonctionnalités d'éditions et mise au point (debug) de Javascript, feuilles de styles CSS, arbre DOM avec contrôle des syntaxes respectives, etc.
  - *NetBeans* est un environnement de développement Java libre soutenu par SUN. Il concurrence directement Eclipse. Il propose en plus d'un IDE Java complet, des perspectives de modélisation utiles pour une approche BPM/SOA avec notamment un modeleur productif pour la construction de schémas XSD, ainsi que des facilités pour la modélisation de services, WSDL et de processus métiers BPEL.

## 22.2 Une classification des solutions se dégage

Ainsi qu'en témoignent les descriptions précédentes, plusieurs catégories d'outils se structurent progressivement pour couvrir l'ensemble du cycle de vie d'une architecture SOA. Il est d'ores et déjà possible de les grouper en sept familles ainsi que l'illustre la figure 22.2.

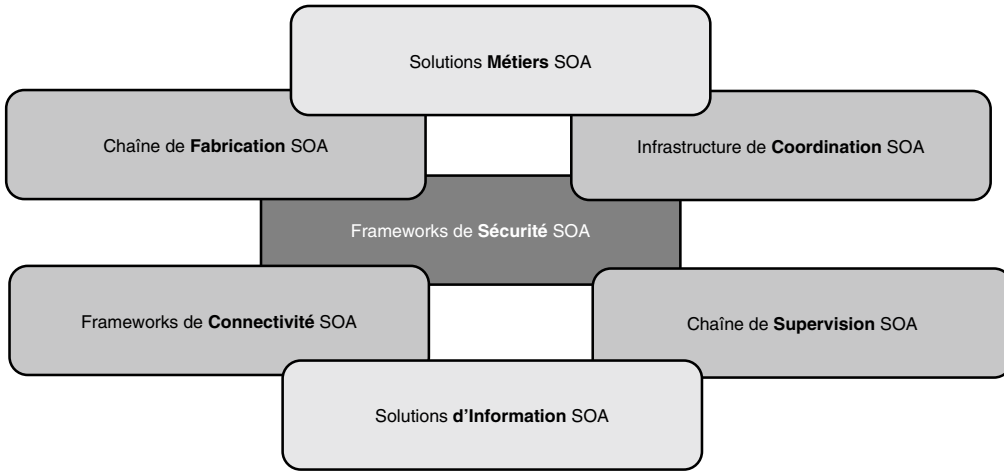



**Figure 22.2** — Les sept familles de solutions en route vers SOA

## En résumé

L'unanimité des acteurs ne doit pas cacher l'étendue du chemin restant à parcourir pour la grande majorité des offres.

Si le marché est loin d'être aujourd'hui parvenu à une maturité, une tendance de spécialisation reste prévisible à l'échelle du marché des solutions.

# Annexes

## GLOSSAIRE

**A**

| | |
|---|---|
| AJAX | Asynchronous JavaScript And XML |
| AOP | Aspect Oriented Programming |
| APS | Application Platform Suite |

**B**

| | |
|---|---|
| BAM | Business Activity Monitoring |
| BI | Business Intelligence |
| BPA | Business Performance Analysis |
| BPM | Business Process Management |
| BRM | Business Rule Management |
| BOXML | Business Object eXtensible Markup Language |
| BPEL | Business Process Execution Language |
| BPMN | Business Process Modeling Notation |

**C**

| | |
|---|---|
| CDI | Customer Data Integration |
| CEP | Complex Event Processing |
| CICS | Customer Information Control System |
| CMMI | Capability Maturity Model Integrated |
| CMS | Content Management System |
| CORBA | Common Object Request Broker Architecture |
| CRUD | Create, Read, Update, Delete |

**D**

| | |
|---|---|
| DMZ | DeMilitarized Zone |
| DSL | Domain Specific Language |
| DCOM | Distributed Common Object Model |

**E**

| | |
|---|---|
| EAI | Enterprise Application Integration |
| ECI | Enterprise Content Integration |
| EDA | Event Driven Architecture |
| EDI | Échange de Données Informatisées |
| EII | Enterprise Information Integration |
| EIMS | Enterprise Information Management System |
| EJB | Enterprise Java Beans |
| ESB | Enterprise Service Bus |
| ETL | Extract, Transform and reLoad |

**F**

| | |
|---|---|
| FTP | File Transfer Protocol |

**G**

| | |
|---|---|
| GXA | Global XML Architecture |

**I**

| | |
|---|---|
| IDE | Integrated Development Environment |
| IECM | Interoperable Enterprise Content Management |
| IIOP | Internet Inter-ORB Protocol |

| | |
|---|---|
| IMS | Information Management System |
| ISP | Integration Service Provider |

**J**

| | |
|---|---|
| JBI | Java Business Integration |
| JEE | Java Extended Enterprise |
| JMX | Java Management Extension |
| JMS | Java Message Service |

**M**

| | |
|---|---|
| MOM | Message Oriented Middleware |
| MDA | Model Driven Architecture |
| MDM | Master Data Management |
| MQ | Message Queuing |
| MIME | Multipurpose Internet Mail Extensions |
| MTOM | Message Transmission Optimization Mechanism |

**O**

| | |
|---|---|
| OASIS | Organization for the Advancement of Structured Information |
| OCL | Object Constraints Language |
| OOP | Object Oriented Programming |
| OMG | Object Management Group |

**P**

| | |
|---|---|
| PDI | Product Data Integration |
| POC | Proof of Concept |
| POP | Post Office Protocol |
| POA | Process Oriented Architecture |
| PSM | Platform Specific Model |
| PIM | Platform Independant Model |

**R**

| | |
|---|---|
| REST | Representational State Transfer |
| RMI | Remote Method Invocation |
| ROA | Resource Oriented Architecture |

**S**

| | |
|---|---|
| SaaS | Software As A Service |
| SAM | Service Activity Monitoring |
| SCA | Service Component Architecture |
| SDO | Service Data Object |
| SI | Système d'Information |
| SLA | Service Level Agreement |
| SOA | Service Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| S/MIME | Secured Multipurpose Internet Mail Extensions |
| SMTP | Simple Mail Transfer Protocol |
| SOBA | Service Oriented Business Application |
| STP | SOA Tools Plateform |

**U**

| | |
|---|---|
| UDDI | Universal Description and Discovery Interface |
| UML | Unified Modeling Language |

**W**

| | |
|---|---|
| W3C | World Wide Web Consortium |
| WMI | Windows Management Instrumentation |
| WSDL | Web Service Description Language |
| WSRP | Web Services Remote Portlet |

**X**

| | |
|---|---|
| XHTML | eXtensible Hypertext Markup Language |
| XML | eXtensible Markup Language |
| XSLT | eXtensible StyleSheet Language Transformation |
| XAML | eXtensible Application Markup Language |
| XUL | eXtensible User-interface Language |

# LE WSDL DOCUMENT LITERAL WRAPPED AU FORMAT WSDL 2.0

Ce document a été réalisé automatiquement par une transformation XSLT.

```
<?xml version="1.0"?>
<wdsl:description xmlns:wsdl="http://www.w3.org/2006/01/wsdl"
     xmlns:si="http://www.distribution.com/ServiceIntervention"
     targetNamespace="http://www.distribution.com/ServiceIntervention">
     <wdsl:types>
          <xsd:schema xmlns:bo="http://www.portos.org/bo"
               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
               xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://www.portos.org/bo">
               <xsd:element name="recupererEtatDemande">
                    <xsd:annotation>
                         <xsd:documentation>le type wrapped de recupération de
l'état de la demande</xsd:documentation>
                    </xsd:annotation>
                    <xsd:complexType>
                         <xsd:sequence>
                              <xsd:element name="NumeroDemande" type="xsd:int"/>
                              <xsd:element name="NumeroDistributeur"
type="xsd:int"/>
                         </xsd:sequence>
                    </xsd:complexType>
               </xsd:element>
               <xsd:element name="recupererEtatDemandeResponse">
                    <xsd:annotation>
                         <xsd:documentation>le type wrapped de la réponse
                         </xsd:documentation>
                    </xsd:annotation>
                    <xsd:complexType>
                         <xsd:sequence>
                              <xsd:element name="EtaDemande" type="xsd:string"/>
                         </xsd:sequence>
                    </xsd:complexType>
               </xsd:element>
          </xsd:schema>
     </wdsl:types>
     <wdsl:interface name="ServiceInterventionPort">
```

```
          <wdsl:operation name="recupererEtatDemande" pattern="http://
www.w3.org/2006/01/wsdl/in-out">
               <wdsl:input xmlns:bo="http://www.portos.org/bo"
element="bo:recupererEtatDemande"/>
               <wdsl:output xmlns:bo="http://www.portos.org/bo"
element="bo:recupererEtatDemandeResponse"/>
          </wdsl:operation>
     </wdsl:interface>
     <wdsl:binding xmlns:wsoap="http://www.w3.org/2006/01/wsdl/soap"
          name="ServiceInterventionBinding"
          interface="si:ServiceInterventionPort"
          type="http://www.w3.org/2006/01/wsdl/soap"
          wsoap:version="1.1"
          wsoap:protocol=http://www.w3.org/2006/01/soap11/bindings/HTTP>
          <wdsl:operation ref="si:recupererEtatDemande"
wsoap:soapAction="urn:#recupererEtatDemande"/>
     </wdsl:binding>
     <wdsl:service name="ServiceIntervention" interface="si:ServiceInterventionPort">
          <wdsl:endpoint
               name="ServiceInterventionPort"
binding="si:ServiceInterventionBinding"
               address="http://services.portos.com/wsdl/ServiceIntervention"/>
     </wdsl:service>
</wdsl:description>
```

# Index

Photocomposition : **SCM**, Toulouse

INFOPRO

TYPE D'OUVRAGE

| L'ESSENTIEL | SE FORMER | RETOURS D'EXPÉRIENCE |
|---|---|---|

MANAGEMENT DES SYSTÈMES D'INFORMATION

APPLICATIONS MÉTIERS

ÉTUDES, DÉVELOPPEMENT, INTÉGRATION

EXPLOITATION ET ADMINISTRATION

RÉSEAUX & TÉLÉCOMS

Xavier Fournier-Morel, Pascal Grojean
Guillaume Plouin, Cyril Rognon
*Préface de Luc Fayard*

# SOA

## Le guide de l'architecte du SI

*2e édition*

**Cet ouvrage s'adresse** aux responsables des systèmes d'information, aux maîtrises d'ouvrage et maîtrises d'œuvre, aux équipes d'exploitation.

Les architectures orientées services (SOA) offrent un nouveau modèle qui permet de construire des systèmes informatiques **évolutifs** et rapidement **adaptables**.

Cet ouvrage en présente les concepts et les usages de manière détaillée. Il se propose de guider le lecteur dans la **mise en œuvre d'une architecture SOA** en décrivant une méthodologie et en présentant les outils indispensables à leur concrétisation.

- La première partie dresse le cahier des charges d'un SI idéal, moderne et « agile ».
- La deuxième explique en détail l'approche SOA.
- La troisième traite d'abord de la modélisation des services et des processus métier, puis de l'impact de SOA sur la gestion de projet.
- La quatrième montre comment les standards et outils associés aux Web Services s'inscrivent dans une démarche SOA.
- La cinquième détaille certains aspects techniques d'un cas réel.
- La sixième positionne SOA vis-à-vis de Web 2.0.
- La dernière partie dresse un panorama de l'offre du marché.

Les auteurs font partie de SQLI Consulting, département conseil du groupe SQLI, spécialiste depuis 15 ans du conseil et de l'intégration des nouvelles technologies. PASCAL GROJEAN anime SQLI Consulting, et CYRIL ROGNON en est le directeur « Capitalisation et R&D ». GUILLAUME PLOUIN est responsable de la veille technologique du groupe SQLI. XAVIER FOURNIER-MOREL est responsable du pôle architecture de SQLI Suisse.

Leur expérience les a particulièrement sensibilisés aux problématiques d'urbanisation, de construction et de maintenance de systèmes complexes.

ISBN 978-2-10-053549-1

www.dunod.com

DUNOD