Travaux d'Etudes de Licence d'Informatique

Les Architectures Orientées Services

GEAY Sébastien REPETTO Pierre VICARD Sébastien

juin 2005

Table des matières

Introduction

Lancée par Gartner Group, la notion d'AOS (pour architecture orientée services) définit un modèle d'interaction applicative mettant en oeuvre des connexions en couplage lâche entre divers composants logiciels (ou agents). Un service désigne une action exécutée par un composant "fournisseur" à l'attention d'un composant "consommateur", basé éventuellement sur un autre système.

Modèle de type composant. Permet une interraction entre eux des services qui composent l'application. Ceux-ci doivent pouvoir communiquer entre eux sans restrictions d' OS ou quoique ce soit d'autre.

Chapitre 1

Définition d'une architecture orientée services

1.1 Architecture

La création d'applications dans l'entreprise est très souvent pilotée par les besoins à très court terme : « Je suis capable de développer telle application sous tel délai avec telles fonctionnalités ». Pas ou peu de place est faite aux problématiques de montée en charge à long terme et d'évolution des besoins fonctionnels. Il en résulte une forte disparité dans les applications en place.

Le choix en amont d'une architecture va guider la manière dont les applications vont être conçues, tant sur la manière de concevoir les composants et les services, que sur leur interaction dans le cadre du système d'information global. C'est de plus la seule option valable pour rendre les Services Web natifs dans l'ensemble des applications créées dans l'entreprise.

Dans tous les cas, quels que soient les choix effectués (installation de progiciels, développements spécifiques ou combinaison des deux) et la méthodologie d'implémentation utilisée, il en résultera toujours une architecture. Soit elle sera imposée par les contraintes court terme de type délais, coûts, fonctionnalités exclusivement, soit elle sera stratégiquement décidée en amont.

1.2 Les services

Il a fallut plusieurs années depuis l'apparition de la notion de AOS (dont l'appellation revient, à Gartner), avant que l'on ne commence à voir ressortir une unanimité sur ce qui définit un service. Il y a quatre critères à satisfaire pour faire partie du club :

1.2.1 frontières explicites

Un service définit clairement ses points d'entrée. Aucune communication par effet de vases n'est permise. Un Service ne peut pas partager l'état avec un autre service (ou autre système, application, ou ce qu'on veut). Ainsi, un service ne permet pas qu'on s'intègre à lui en partageant sa base de donnée, en appelant directement ses implémentations interne (sous forme de composants métier ou autres), ou de quelque autre manière. Il définit clairement ses points d'entrées, et bloque toute tentative de pénétration par un quelconque autre accès.

1.2.2 echanges par contrat

Les échanges se font par contrat : Un service ne donne aucune indication sur la manière dont il est implémenté, de la technologie avec laquelle il est créé, ou de la plateforme sur la quelle il est déployé. Il expose simplement un contrat qui stipule se que sont les formats de données échangés, les points d'entrée, quelles informations sont acceptée en entrée pour chaque point d'entrée, et quelles informations sont données en réponse. Un service ou système qui appellerait ce service ne peut pas assumer une quelconque connaissance sur l'implémentation du service, et ne doit pas avoir de problème pour parler à un autre service qui expose le même contrat si le service est remplacé par un autre. Cela permet l'évolution. Si une règle métier change, tant que le contrat continue à être respecté, aucun autre service ou système n'est touché.

1.2.3 autonomie

Un service doit être autonome. Si les frontières explicites forcent déjà qu'un service ne communique pas de façon transversale avec un autre service, et qu'il ne partage aucun état avec un autre service, l'autonomie va encore plus loin.

Un service doit "posséder" ses propres données. Cette possession est exclusive. Un service ne peut en aucun cas accéder à une base de données à laquelle un autre service peut accéder, même si cette base est en lecture seule. Par exemple, un service ne peut pas renvoyer la clé d'un enregistrement et laisser le service qui l'a appelé utiliser la clé pour récupérer l'information sur la base.

De même, un service ne peut pas partager d'état entre ses propres instances. Par exemple, un service ne peut pas stocker d'information dans des variables statiques. Un service ne peut pas avoir un cache utilisé par toutes les instances. Un service ne peut même pas avoir de variables persistées entre un appel et le suivant. C'est ce que l'on appelle communément ne pas avoir d'état. Il doit pourvoir à récupérer l'information sur l'appel pour savoir à quelle étape de l'exécution il doit se situer (si l'exécution comprend plusieurs appels successifs).

données propres

La compatibilité est négociée en fonction de stratégies : Un service se doit de définir toutes les méthodes et protocoles d'échange qu'il permet en terme de stratégies. Si le service permet des séquences d'appels successifs, ils doivent être définis dans ces stratégies. Ainsi, un autre service, en utilisant simplement

les contrats et les stratégies, peut définir s'il peut appeler ce service, et quelles méthodes et protocoles d'échange utiliser.

1.3 AOS

La base d'une AOS repose sur des services répondant, notamment, aux critères suivants :

1.3.1 Faiblement couplée

Un service est en couplage faible quand il n'est pas autorisé à appeler directement un autre service. Il délègue cette responsabilité à un traitement spécialisé que l'on nomme fonction d'orchestration.

Grâce au couplage faible, on peut réutiliser un service sans devoir reprendre des services qui lui sont liés.

A partir de cette première définition, on ajoute d'autres principes techniques qui contribuent à découpler les services entre eux :

1. Un service est activable indépendamment de sa technologie. Pour ce faire, l'activation se réalise par l'envoi (et la réception) d'un message XML (il ne s'agit donc pas d'une activation d'objet distribué comme en Corba ou DCOM).
2. Un service peut être activé suivant un mode asynchrone. Dans ce cas, le service s'abonne à un événement auprès de la fonction d'orchestration (ce qui renvoi aux principes de l'architecture EDA - Event Driven Architecture).

1.3.2 distribution

Là où l'entreprise devait maintenir certaines fonctions et données comme les tarifs de fournisseurs, elle pourra interroger directement, via son applicatif, le service du fournisseur sans se préoccuper de sa mise à jour.

1.3.3 invocation et publication

Les services doivent être invocables et publiables quels que soient les systèmes utilisés.

1.3.4 orientation métiers

Les services permettent de gérer les applications avec une approche fonctionnelle, par l'intermédiaire de processus métiers intégrés à l'architecture, permettant de piloter les applications sans développement conséquent.

Chapitre 2

Théorie

2.1 Pourquoi s'orienter vers une AOS ?

L'entreprise qui gagne est une entreprise agile.

L'agilité des entreprises est devenue un impératif. Elles doivent s'adapter en permanence et rapidement à des situations telles que des changements technologiques, fusions, acquisitions, scissions... Mais les entreprises se sont habituées à ce que leurs systèmes d'information ralentissent, voire bloquent la mise en oeuvre de leurs décisions. C'est évidemment l'activité qui doit piloter l'informatique et non l'inverse.

Pour être agile il faut un système d'information intégré et réactif Le premier frein des systèmes d'information réside dans la difficulté de leur intégration. Faire dialoguer deux systèmes différents d'une manière flexible et facilement évolutive est un problème persistant. Ce dont l'entreprise a besoin, et qui n'était pas possible auparavant, est une intégration de type « faiblement couplée », présentant une capacité d'amélioration des applications en place.

Cette intégration doit être globale ; elle doit couvrir :
- L'intégration entre partenaires, clients...
- L'intégration entre applications
- L'intégration à l'intérieur même des applications.

2.1.1 Pourquoi une AOS ?

Avec ce type d'architecture, les processus métiers, les présentations, les logiques applicatives et les données sont séparés dans des couches distinctes et faiblement couplées. L'architecture décompose ainsi les applications en services, disponibles et réutilisables.

Le système d'information est globalement intégré. Il en résulte des fonctions modifiables en temps réel, des applications évolutives notamment grâce à leur

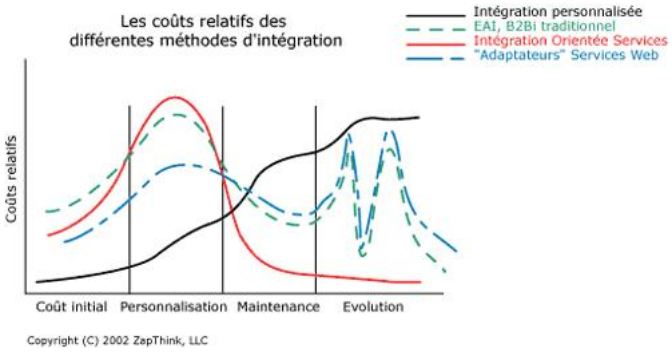


FIG. 2.1 – Coûts relatifs

indépendance matérielle, des utilisateurs fonctionnels maîtres des processus métiers et une meilleure séparation entre les tâches techniques et fonctionnelles dans les processus de développement.

C'est, d'après une étude de Zapthink, la solution économiquement la plus performante :

2.2 Quels en sont les bénéfices ?

La AOS est l'aboutissement de plusieurs années de corrections des erreurs rencontrées dans les applications. Ce n'est pas à proprement parler une révolution technologique.

Elle permet de créer de manière incrémentale de nouvelles fonctions, de les combiner aux services existants pour créer des applications composites.

L'expérience montre que cela minimise les risques car elle capitalise sur les fonctions existantes : livraisons plus rapides, appui sur la robustesse des fonctions déjà stabilisées.

Elle permet de modifier les applications en temps réel. Une application peut, en effet être, modifiée sans intervenir sur les processus métiers eux-mêmes. Le travail du développeur est simplifié et plus sécurisé.

Le découplage préserve l'avenir : les applications sont conçues indépendamment de l'infrastructure matérielle. Lorsque celle-ci évolue, le faible couplage entre « hardware », services et processus métiers limite les risques liés aux modifications des fonctions et permet notamment la compatibilité avec toute interface actuelle et future : Windows, Internet, téléphone mobile... L'indépendance de l'emplacement physique du service permet également d'assurer performance, scalabilité et haute disponibilité des applications.

La réactivité est possible grâce à l'affranchissement des contraintes techniques. L'orientation métier donne aux responsables fonctionnels la capacité de faire évoluer les applications sur la base des demandes et contraintes métiers, sans préoccupations technologiques.

Sur la partie développement :

- Le degré d'implication des concepteurs fonctionnels est accru par leur maîtrise des processus métiers.
- Les développeurs se concentrent sur des fonctions élémentaires sans se préoccuper de l'infrastructure d'accueil. Ils sont plus efficaces et offrent une meilleure valeur ajoutée.
- Le développement collaboratif est également plus simple. Le travail sur plusieurs projets simultanés et l'affectation des ressources à des fonctions plus élémentaires facilitent le travail du chef de projet.
- Les tests unitaires sont facilités et la détection d'erreurs au plus tôt dans le processus de développement limite les risques.

La réutilisabilité des composants ouvre naturellement les services à une consommation par des applications externes.

L'indépendance des fonctions par rapport aux processus métiers et à l'architecture d'accueil, préserve l'évolutivité, la flexibilité et l'intégration.

Vers plus de sécurité : Le système de sécurité est basé au niveau du service. C'est le point central qui permet de renforcer la sécurité (multi niveaux d'authentification, authentification forte,...).

2.3 Les benefices techniques

La construction de processus commerciaux est plus rapide et moins cher :
– les services existants peuvent plus facilement être réutilisés
– les applications peuvent exposer leurs services de façon plus standart
Les applications peuvent être exposées plus facilement à divers clients :
– clients Windows, ASP .NET/JSP, etc...

2.4 Les bénéfices financiers

Les commerciaux comprennent les services
– les personnes externes peuvent plus facilement dialoguer avec eux.
Les processus commerciaux deviennent explicites.
– ils peuvent ainsi être plus facilement compris et améliorés.
Les applications ou les processus commerciaux peuvent être plus facilement exportés
– Parce qu'ils ont été bien définis.

Une des clés est l'interopérabilité, qui est la possibilité d'utiliser les fonctions entre n'importe quelle sorte de plateforme, sans s'intéresser au langage de programmation, au système d'exploitation, au type d'ordinateur, etc...

Dans l'exemple ci-dessus, la fonction de « contrôle d'inventaire » a pu être écrite comme un service qui était requis pour UNE application, par exemple une qui surveille l'inventaire et qui le réordonne automatiquement quand cela est demandé, mais nous pourrions trouver plus tard que le même service peut être utilisé sans modification pour supporter un outil de surveillance d'inventaire basé sur le Web (Web-based, je sais pas si c'est en théorie —architecture— ou en pratique —physiquement hébergé—) utilisé par un employé humain.

Intérieurement, la réutilisation des fonctions d'application(*) est un bénéfice clé, car elle amène à réduire les coûts de développement. Une implication à long terme de la réutilisation des services est la réduction des fonctions redondantes dans l'entreprise, une simplification de son infrastructure, et ainsi un moindre coût de maintenance du code. En organisant les applications comme des utilisateurs de services, nous obtenons un modèle d'intégration bien plus flexible et agile, nous autorisant à rapidement mettre à jour le modèle du processus commercial (business process model), comparé aux techniques de programmation traditionnelles.

Chapitre 3

En pratique

3.1 Fonctionnement

La AOS est une collection de services distribués au travers du réseau.

Comme les processus métiers, les présentations (écrans), les logiques applicatives et les données sont séparés dans des couches distinctes et faiblement couplées, il faut des outils complets de gestion de chacune de ces composantes :

- Les processus métiers (Gestion des processus métiers)
- Les présentations (Réalisation d'interfaces utilisateurs)
- Les logiques applicatives (Réutilisation de composants)
- L'accès aux données.

Principe de fonctionnement

Un consommateur (une application, un poste utilisateur Windows ou web...) demande une fonction. Le serveur de service gère la requête en appelant le service stocké dans un référentiel, se charge de son exécution puis renvoie la réponse au consommateur initial sous le format désiré.

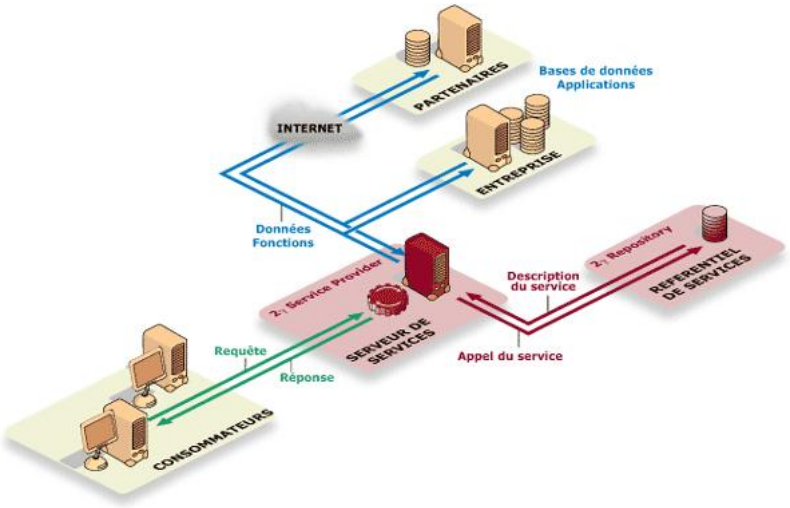


FIG. 3.1 – Principe de fonctionnement

Fonctionnement d'un service

Un service est dédié à une fonction simple et n'est pas alloué à un utilisateur en particulier mais à une multitude de clients qui le partagent. Il est défini par une interface qui peut être invoquée par un autre service ou une application. Il ne maintient pas une conversation avec le client. L'aspect « données » est invisible pour le client. Au-delà, un service invoqué ne connaît pas les requêtes exécutées précédemment ni celles qui le seront en aval. Un service a seulement besoin des paramètres qu'il requiert en entrée. Il s'exécute ensuite en une seule phase car il ne stocke rien d'une invocation à une autre.

Cette organisation rend les systèmes capables de supporter des charges élevées. Le fonctionnement sous forme de services simples, sans mode conversationnel, facilite la « répartition de charge » de l'exécution et la surveillance des erreurs. Il devient plus facile d'augmenter la puissance du système d'information par l'augmentation du nombre d'instances ou de serveurs.

Caractéristiques techniques d'un service

Le but fondamental d'un service est de représenter une unité logique et complète d'un processus métier. Ainsi, il est composé d'un service d'interface et d'un service d'implémentation :

- L'interface définit le service et toute la logistique pour son invocation
- L'implémentation contient un ou plusieurs composant(s) réalisant le travail
- L'interface et l'implémentation doivent être décrites séparément car l'interface définit logiquement le service et l'implémentation le définit techniquement.

Fonctionnement - Organisation	
Réponse	Asynchrone ou Synchrone
Couplage faible	Indépendant de l'architecture matérielle et standard en terme d'interface
Granularité grossière	Encapsulation des composants techniques en composants métiers. C'est cet assemblage qui permet d'assurer une fonction moyennant l'utilisation combinée de multiples services.
Indépendance de l'implémentation	Modification en temps réel du fournisseur de services. Performance (répartition des charges, haute disponibilité, scalabilité).

Orientation "Gestion des processus métiers" ("Méthodologie centrée métier")

L'objectif de l'organisation autour du « business model » est de représenter graphiquement une grande variété de processus partagés, en vue de leur réutilisation ; chacun de ces processus répondant à des contraintes différentes. En appliquant la bonne contrainte au bon niveau, et non d'une manière physique prématurée, les processus sont orientés vers l'activité et non vers la technologie. C'est la clé de l'agilité.

Cette construction requiert les concepts suivants :
- Compréhension : Alignement de la terminologie par la compréhension de la sémantique métier et la signification réelle du vocabulaire utilisé, pouvant être extrait et agrégé pour être partagé.
- Métier : Compréhension des objectifs de l'entreprise que les processus métiers doivent permettrent d'atteindre. Les règles métiers doivent être le reflet de la logique d'entreprise, en permettant d'analyser les impacts d'un changement, les possibilités de réutilisation et la définition des besoins fonctionnels.
- Extension : Niveau nécessaire à la collaboration avec les partenaires mais aussi avec les systèmes internes (historiques). Il fixe les spécifications de base à la collaboration.
- Implémentation : Analyse en profondeur des besoins techniques. C'est le niveau où les objets métiers se concrétisent physiquement sur des bases techniques préalablement définies.

Ainsi, le principe des "business processes" dans la AOS est de répondre à :
- La modélisation des processus
- L'automatisation des processus
- L'orchestration des services.

3.2 Conception

3.2.1 Décomposition AOS

On distingue trois niveaux de décomposition AOS :

Le premier est la découverte des opérations exposées par les services métiers à partir de la modélisation des processus. Il s'agit d'un regroupement d'activités formant le périmètre fonctionnel que l'on souhaite exposé aux consommateurs.

Le processus est la description des traitements au niveau du modèle organisationnel et modélisé soit sous la forme d'un MOT (Merise) soit un Diagramme d'activités UML. Le processus assure une fonction d'orchestration. Il orchestre l'appel aux opérations des services métiers et aux phases. C'est à partir de la modélisation du processus que les services métiers et leurs opérations sont découverts et spécifiés.

Une opération correspond à un traitement exposé par un service métier. Dans l'approche AOS, on admet qu'un service métier puisse exposé plusieurs opérations. L'opération est découverte à partir de l'analyse du processus organisationnel.

Les parties du processus organisationnel qui ne sont pas exposées sous la forme d'opération d'un service métier sont regroupées sous le concept de phase.

En final, un processus organisationnel est donc découpé en une série d'opérations et de phases. Le concept d'opération reste facultatif car il n'est pas toujours nécessaire de spécifier des services métiers, ce qui n'empêche pas d'utiliser la démarche AOS plus tard lors de la découpe des phases sous la forme de services attachés aux catégorises.

Le second niveau consiste à décomposer les opérations et les phases découvertes lors de la modélisation du processus sous la forme de services rattachés aux catégories. Chaque opération et phase devient ainsi un orchestrateur d'appel vers les services exposés par les catégories (service de type externe).

Le concept de catégorie est pivot dans la démarche AOS. Une catégorie est un agrégat de classes du diagramme de classes (ou entités/association en Merise) qui respecte des propriétés strictes de stabilité, de continuité et de consistance métier. Historiquement le concept de catégorie est aussi nommé objet-métier ou sujet-métier. Il a été introduit la première fois par Grady Booch dans le cadre de UML.

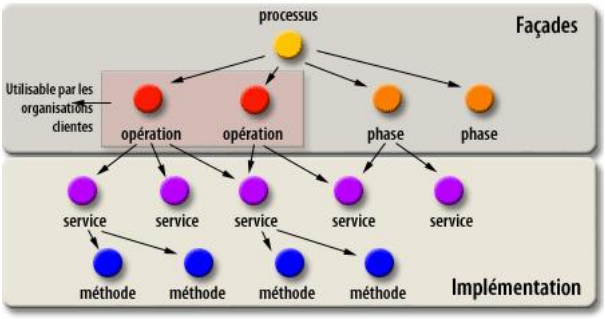


FIG. 3.2 – Shéma

La catégorie permet de décomposer la logique des opérations et des phases sous la forme de services rattachés aux différentes catégories du système d'information. Il s'agit de services de type externe et qui réifie le motif d'architecture applicative AOS. Selon un principe de base de la taxonomie, on impose qu'un service appartienne à une et une seule catégorie. La catégorie est donc à la fois un régulateur de la granularité des services (on décompose les traitements autour des catégories) mais aussi de classificateur (on range chaque service dans une et une seule catégorie)

Le troisième niveau dépend de l'usage ou non d'un langage orienté objet. Il s'agit de décomposer chaque service exposé par une catégorie sous la forme de méthodes attachées aux classes qui constituent la catégorie d'appartenance. Cette décomposition se fait uniquement sur les classes de la catégorie d'appartenance, jamais sur les classes d'autres catégories (principe d'isolation des catégories entre elles).

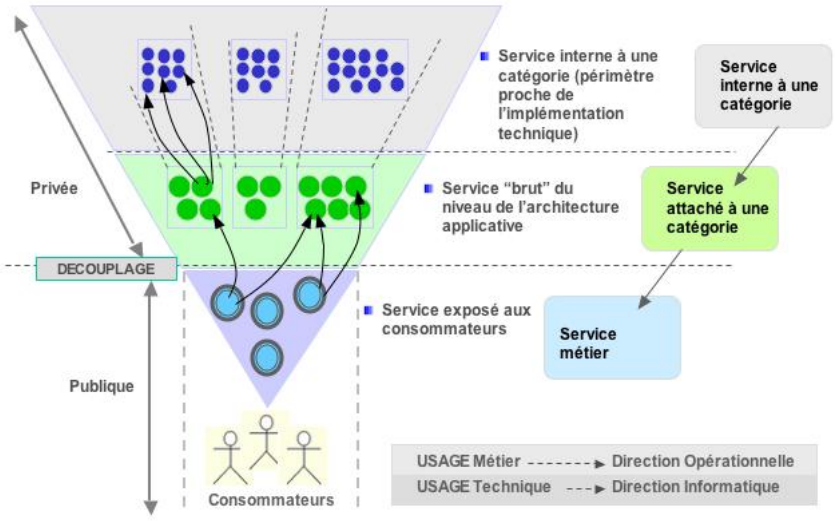


FIG. 3.3 – Shéma

3.2.2 Service

Un service est une unité de traitement qui respecte les propriétés de couplage faible, d'exposition d'un contrat de service et qui correspond à l'un des types suivants :

Service métier dont le périmètre correspond à une fonctionnalité que l'on souhaite exposer à un consommateur. Les fonctionnalités exposées sont regroupées sous le terme d'opération. Un service métier expose donc une ou plusieurs opérations aux consommateurs. L'opération est un regroupement d'activité au niveau de la modélisation du processus. Le service métier est un concept qui fait sens pour les maîtrises d'ouvrage et maîtrises d'oeuvre du fournisseur et du consommateur.

Service externe attaché à une catégorie dont le périmètre est issu du motif d'architecture applicative AOS. Ce type de service correspond à une préoccupation de niveau génie logiciel. Le service externe est un service exposé par une catégorie. Le service externe est un concept qui fait sens pour le fournisseur et qui n'est pas visible du consommateur.

Service interne à une catégorie qui permet l'implémentation des services externes. Un service interne n'est visible que dans le périmètre de sa catégorie d'appartenance. Le service interne est un concept qui fait sens uniquement pour la maîtrise d'oeuvre du fournisseur. Il n'est pas visible du consommateur.

Le contour du service est logique. L'implémentation physique du service s'appuie sur le concept de composant.

3.2.3 Composant

Un composant correspond à une unité de traitement exécutable. Le périmètre d'un composant est directement régulé par le pattern d'architecture applicative AOS et à ce titre il réifie obligatoirement soit :

- Une façade d'une catégorie.
- Une partie ou la totalité de l'implémentation interne d'une catégorie.
- Un service métier.
- Un processus.

Un composant est typé selon les couches de l'architecture N-Tiers. Il est, de manière exclusive, spécialisé soit dans la couche de présentation, soit dans celle des règles applicatives soit dans l'accès aux données.

3.3 Réalisation

Pour encadrer la spécification et la mise à disposition des services métiers, il faut définir deux plans-types importants . Il s'agit d'une part du cahier des charges AOS, qui contient la spécification complète de chaque service métier et qui peut faire l'objet d'une communication partielle aux consommateurs. Et d'autre part du plan-type du contrat d'utilisation du service qui reprend certaines parties du cahier des charges (notamment les préconditions, postconditions et exceptions) et ajoute d'autres informations indispensables pour la description des modalités d'usage du service et des devoirs des consommateurs.

3.3.1 cahier des charges d'une AOS

Dans le cahier des charges on spécifie les services de type métier. La découverte de ces services se mène à partir de la modélisation des processus organisationnels (MOT en Merise, Diagramme d'activités en UML). La maîtrise d'ouvrage est fortement impliquée dans ce travail et il faut prendre en compte les spécificités suivantes des services métiers :

- Contour fonctionnel mettant en relation un fournisseur avec plusieurs consommateurs.
- Conception des variantes du service métier, c'est-à-dire des besoins d'adaptation du service selon les différents contextes d'usage connus et probables. Une variante de service représente un comportement adapté du service par rapport à sa version d'origine. Il peut s'agir d'adaptation de nature très variée : présentation (marque blanche...), métier (offre commerciale...), technique (environnement d'exécution recette, exploitation...), qualité de services (plages de disponibilité, temps de réponse garanti...). La gestion des variantes s'appuie sur la mise en oeuvre du modèle d'architecture de gestion par les contextes (cf. ci-dessous).
- Spécification précise des règles de préconditions, postconditions et d'exceptions du service métier. Ces règles participent directement à la réalisation du contrat d'utilisation du service, elles sont donc lisibles par le consommateur.

3.3.2 Contrat d'utilisation

Un service est décrit par un contrat d'utilisation qui précise ses conditions d'usage (notamment les préconditions, postconditions, exceptions) et les devoirs que le consommateur doit respecter.

Selon le type du service, le contrat d'utilisation prend différentes formes :

- Service interne à une catégorie : il s'agit d'un service proche de l'implémentation du logiciel et dont l'usage est limité au périmètre d'une seule catégorie. Ce contrat peut s'exprimer à l'aide d'une documentation de niveau API comme par exemple un javadoc.
- Service externe à une catégorie, c'est-à-dire exposé par la catégorie : il s'agit d'un service qui se détache de l'implémentation du logiciel. Ce type de service est administré dans une optique de réutilisation entre les projets. Donc, le contrat d'utilisation ne peut pas se limiter à une simple API. Il faut l'étendre à une description informelle en insistant notamment sur les préconditions et postconditions. Il est nécessaire d'utiliser un outil de gestion des contrats adossé à une base de données. Cet outil implémente le modèle d'architecture de gestion par les contextes afin de gérer des contrats cadres puis des spécialisations successives selon les besoins.
- Service métier. Ce type de service est complètement détaché de l'implémentation du logiciel. Sa valeur est directement perçue par les maîtrises d'ouvrage du fournisseur et des consommateurs. La gestion des contrats d'utilisation nécessite l'usage d'un outil convivial et partagé entre les maîtrises d'ouvrage du fournisseur et des consommateurs. Ici également, cet outil implémente le modèle d'architecture de gestion par les contextes afin de gérer des contrats cadres puis des spécialisations successives.

3.3.3 Modèle d'architecture de gestion par les contextes

Ce motif permet la création de variantes de services grâce à un dispositif de gestion de paramètres. Les paramètres susceptibles de faire varier le comportement d'un service sont localisés dans un modèle d'adaptation.

Un produit logiciel permet de créer différentes valorisations des paramètres selon les contextes d'usage du service : un canal, un partenaire, une filiale... Ce produit dispose d'une console de paramétrage qui permet d'agir sur les variantes sans intervenir dans le code du service (principe même du paramétrage). Dans la pratique, cette console peut être utilisée par les maîtrises d'ouvrage afin d'agir de manière autonome sur certains paramètres du modèle d'adaptation.

L'organisation des variantes d'un service respecte généralement la logique d'un arbre qui permet de bénéficier des mécanismes d'héritage. Plus précisément, à partir d'une première variante d'origine (la racine) on peut créer des variantes filles et sous filles qui héritent les unes des autres. Par exemple, on peut utiliser ce dispositif pour créer un contrat d'utilisation cadre (un contrat est un ensemble de paramètres qui définit ses différentes clauses) puis grâce à l'héritage le décliner en multiple variantes selon les besoins d'exposition vers les types de consommateurs (filiale, partenaire, canaux...) puis les consommateurs au niveau nominatif (une organisation).

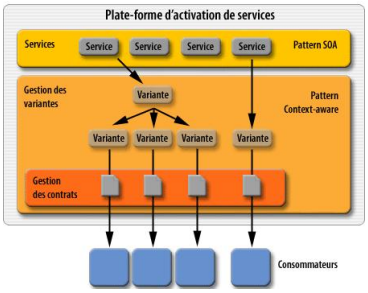


FIG. 3.4 – Shéma

3.3.4 Activation des services

L'activation de services consiste à créer une variante d'un service afin de le configurer conformément aux règles de mise à disposition d'un consommateur : options de fonctionnement, choix du niveau de la qualité d'exploitation... Les règles de mise à disposition forment le contrat d'utilisation du service. Lorsque que le contrat est validé par le fournisseur on considère que le service est activée pour une variante et pour un consommateur.

La plate-forme d'activation de services met en oeuvre le modèle d'architecture de gestion par les contextes (pattern Contexte-Aware) pour la gestion des variantes et des contrats d'utilisation. Cette plate-forme dispose d'une console métier qui permet aux équipes techniques et opérationnelles de paramétrer les données des variantes et des contrats. Certains éléments d'une variante peuvent être directement modifiés par le consommateur si celui-ci dispose des droits délégués par le fournisseur. Il faut alors que la plate-forme d'activation de services autorise le travail collaboratif : plusieurs organisations chez le fournisseur et partage de certains paramétrages avec les consommateurs.

L'activation de services est un domaine clef du AOS car la construction de services répond à une volonté d'exposition de multiples services vers plusieurs consommateurs. La gestion de cette exposition crée un nouvel écosystème de paramétrage des variantes de services et de création des contrats d'utilisation. C'est aussi à ce niveau que les fonctions de monitoring des flux d'exploitation sont mises en oeuvre.

3.3.5 Extensibilité du code des services et réutilisation

Lorsque que l'on souhaite réutiliser un service il est souvent nécessaire de l'adapter au préalable afin qu'il se conforme au nouveau contexte d'utilisation. Cette adaptation ne doit pas nécessiter une duplication du code d'origine, ce qui conduirait à une impasse en génie logiciel (problème d'intégrité et de maintenance de plusieurs codes dupliqués). Pour étendre le code d'origine sans le modifier (principe de fermeture/ouverture du code) il existe trois approches possibles :

- La première consiste à tirer profit des propriétés d'héritage de l'objet et de la programmation par interface. A partir de la classe qui contient le code d'origine du service, on crée une classe fille par variante que l'on spécialise selon les besoins (surcharge ou remplacement des méthodes). Malheureusement ce mécanisme doit être exceptionnel car il conduit à créer des connexions d'héritage qui ne correspondent pas à un objectif de taxonomie au sens de la classification. La maintenance des arbres de classes avec de multiples connexions d'héritage est très difficile.
- La seconde approche consiste à utiliser le modèle d'architecture de gestion par les contextes qui permet de créer des variantes à l'aide d'un paramétrage du code. Cela nécessite une démarche de spécification adaptée ainsi qu'un outillage pertinent (station de paramétrage).
- "La troisième solution est encore prospective mais s'annonce très intéressante. Il s'agit de l'usage des Aspects (AOP : Aspect Oriented Programming). L'AOP complète les langages objets (précompilateur Java, C#) avec des mécanismes qui permettent d'étendre le code sans le modifier. Pour ce faire on définit dans des Pointcuts l'endroit où l'on souhaite ajouter du code en indiquant via des Advices l'événement qui le déclenchera : par exemple " before " l'exécution d'une méthode ou " after " l'initialisation d'une variable. Le code déclenché est écrit sous la forme d'une classe particulière qui est un Aspect (aspect remplace le mot clef class).

L'extensibilité du code et donc la gestion des variantes de service est un sujet déterminant du AOS. Aujourd'hui, la pratique combinée de l'héritage (usage très encadré) et du context-Aware (effort de modélisation et outillage de gestion des paramètres) offre un excellent cadre de développement et d'exploitation des variantes. On ne duplique plus le code, on l'étend. A terme, on pourra ajouter l'AOP pour rationaliser encore plus ces principes.

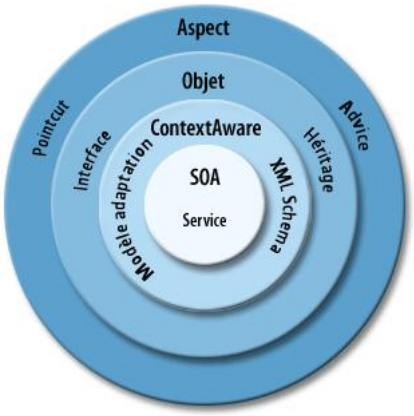


FIG. 3.5 – Shéma

3.3.6 Fonction d'orchestration

La fonction d'orchestration est assurée par les concepts AOS d'opération (celles exposées par le service métier) et de processus mais jamais à d'autres niveaux de l'architecture. C'est un point très important pour rationaliser l'usage des techniques d'orchestration.

Les quatre rôles clefs de la fonction d'orchestration sont indiqués maintenant :

- Gestion de contexte : La fonction d'orchestration prend en charge la conservation de certaines données qui sont nécessaires tout au long de la durée de vie d'un enchaînement. Par exemple, il peut s'agir de mémoriser un résultat qui est réutilisé plus tard dans l'orchestration. Les données mémorisées forme un contexte.
- Gestion transactionnelle : Les éléments assemblés lors de l'orchestration peuvent émettre des mises à jour dans les bases de données. Puisque ces éléments ne se connaissent pas entre eux, ils sont incapables de se synchroniser. C'est la fonction d'orchestration qui prend en charge la gestion d'une unité transactionnelle globale.
- Gestion de la logique applicative : Au-delà des rèles techniques de gestion de contexte et de transaction, la fonction d'orchestration assure un rèle applicatif. Elle contient la logique fonctionnelle d'assemblage des éléments et uniquement cette logique. Les règles métiers restent localisées dans les éléments assemblés. C'est en quelques sortes, la partie « déroulement des opérations » de la fonction d'orchestration.
- Gestion des traitements interactifs : Les opérations et les phases de type IHM enchaînent des écrans. La fonction d'orchestration s'appuie sur un cadre technique de type MVC (Modèle Vue Contrèleur) spécialisé dans la gestion des conversations interactives.

3.4 Les services Web

Les Services Web forment un des éléments d'une AOS. Ils fournissent la base technologique pour faire communiquer les applications entre elles. Ils ont l'im-

mense intérêt de réduire les coûts et la complexité dans la mise en oeuvre d'une AOS. Ils constituent aussi, sans aucun doute, la partie la plus visible du bénéfice d'une telle structure.

Les Services Web communiquent sur la base de standards de communication, éliminant ainsi les frontières des plates-formes technologiques ou des logiciels en se reposant sur des méthodes de communication virtuellement disponibles sur tous les systèmes.

Sur la base d'une AOS, les Services Web deviennent naturels, par opposition à la majorité des progiciels existants, pour lesquels leurs éditeurs doivent mettre à disposition des Services Web en « interface » avec leurs produits. Etant donné que ces applications n'ont pas été conçues autour de services invocables indépendamment, l'utilisateur est tributaire de l'éditeur pour les Services Web qu'il voudra bien lui livrer. Il ne sera pas capable de faire face à tous les besoins internes ou externes en terme d'intégration.

Chapitre 4

Interet actuel et futur

4.1 Comparaison avec un systeme classique

Traditionnellement les organisations des entreprises et des systèmes d'information sont découpées en départements fonctionnels, ce qui engendre des coûts considérables dans la coordination des flux entre eux.

Or la réalité des entreprises est dans la gestion de processus multi départementaux, faisant intervenir différentes responsabilités et applications. Pour répondre efficacement aux demandes des clients, ces flux doivent pouvoir être traités transversalement, indépendamment de l'hétérogénéité technologique et applicative. Au-delà de ces contraintes internes, les processus franchissent également les frontières de l'entreprise. Pour être réactive, l'entreprise doit pouvoir prendre en compte les activités et processus de plusieurs partenaires. Les systèmes d'information doivent donc être adaptés pour répondre à ces challenges. La départementalisation des applications doit pouvoir faire place à une gestion transversale de type métier.

Les systèmes doivent pouvoir absorber ces changements d'où la nécessité croissante d'une intégration du système d'information.

La AOS permet de penser stratégiquement et d'agir tactiquement. En effet, pour absorber les changements, les systèmes ne doivent pas seulement être intégrés mais aussi flexibles.

Une solution d'EAI (Intégration des applications dans l'entreprise) classique (de type « courtier de messages ») ne permet pas d'atteindre l'agilité souhaitée car elle ne repose pas sur des composants faiblement couplés. Les intervenants de chaque côté des systèmes doivent être indépendants. L'entreprise doit se poser la question de quels services exposer plutôt que de quels systèmes vont pouvoir les utiliser.

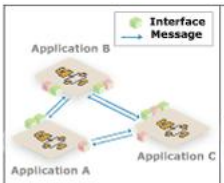
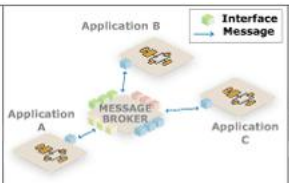
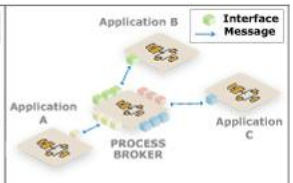
	 Intégration point à point	 EAI traditionnel	 Intégration à base de process
	Point à point	**à base de broker**	**SOA**
Gestion des flux de données	distribué	centralisé	central et distribué
Connexions	points à points	points à points virtuelles	équivalentes à points à points
Interfaces	physiques	physiques	logiques et physiques
Sratégie de métadonnées	aucune	à base de message broker	au niveau entreprise
Réutilisation	peu d'opportunités	centralisée	nombreuses opportunités
Suivi des flux	faible	centralisée	services
Lieu de l'intégration	au point de l'utilisation	au niveau du broker	au point de l'utilisation
Supervision de l'information	conservé par domaine	au niveau du broker	conservé par domaine
Mapping	une seule fois	deux fois ou plus	une seule fois
Bande passante	faible	élévée	faible
Moyens hardware	distribué	central moyens lourds	distribué
Impact des changements	moyens	élevés	faibles
Temps réel	oui	non (asychrone)	oui

FIG. 4.1 – Conclusion

Il convient de se demander au moment de la conception, comment une application tierce pourra utiliser l'application en cours de création au lieu de se demander comment telle ou telle application va interagir avec le développement en cours. Il est fait abstraction des systèmes qui vont ou pourront dialoguer avec l'applicatif.

L'agilité à long terme n'est obtenue qu'en introduisant un niveau d'abstraction et non en se focalisant sur les interfaces ou connecteurs entre différents progiciels. Le Message Broker manque notamment d'un mécanisme central permettant d'appréhender et de visualiser l'ensemble des processus et non pas simplement ceux des interfaces. Il devient alors difficile d'automatiser des tâches fonctionnelles et de les faire vivre. Sur un fonctionnement à base de "Processus métiers", tous les processus de l'entreprise sont centralisés, facilitant maintenance et évolution (changement dynamique).

L'approche AOS apporte :
- Un fonctionnement basé sur les processus métiers
- Des modifications de processus logiques sans re-codage des applications
- Un suivi des flux
- La possibilité de simuler des scénarii avant leur mise en oeuvre
- L'autonomie aux utilisateurs fonctionnels

4.2 Conclusion

Au-delà de la théorie et du mode de fonctionnement, quels sont donc les avantages que l'on peut identifier autour des AOS ?

Premièrement, la nécessité d'un effort conséquent en terme de modélisation se traduit inévitablement par une implémentation de meilleure qualité. Il en découle des facilités en terme de maintenance, de solidité applicative, de distribution, etc. La capacité d'exécuter un traitement, sans avoir connaissance du langage ayant servi à son implémentation, est également un atout indéniable. Tout comme la facilité d'adaptation et de réutilisation de l'implémentation des services.

A l'inverse, la nécessité d'effectuer un effort conséquent en termes de conception représente à ce jour l'un des principaux freins à la banalisation des AOS. De même que le besoin de réalisation d'une couche d'interface supplémentaire, pour laquelle les équipes de développement ne sont généralement pas aguerries.

Avantages	Inconvénients
Obligation d'avoir une modélisation poussée	Coûts de conception et de développement initiaux plus conséquents
Possibilité de découpler les accès aux traitements	Nécessité d'appréhender de nouvelles technologies
Localisation et interfaçage transparents (ouverture accrue)	Existant non AOS dans les entreprises
Possibilité de mise en place facilitée à partir d'une application objet existante	pour des traitements simples (couche supplémentaire) Performances réduites
Réduction des coûts en phase de maintenance et d'évolution	
Facilité d'amélioration des performances pour des applications importantes (répartition des traitements facilitée)	

A l'heure actuelle, les AOS sont encore en plein emergence. Si l'on perçoit déjà l'interet de ce type d'architecture, on est encore loin d'en connaitre tous les atours. La majeur partie des travaux se situent sur les Services Web, notamment parce que cet environnement se renouvelle beaucoup plus rapidement que des environnements plus traditionnels au sein des entreprises. Quelques entreprises,

comme Capgemini, Unilog ou encore Dreamsoft, proposent tout de même des demarches dans ce domaine.

Toute fois, il est fort à parier que les AOS sont l'avenir de la conception informatique, que ce soit par interface Web ou intra-entreprise. Pour s'en assurer, il suffit de voir les chiffres du marché mondial des Services Web, 950 millions de dollars en 2004. Certains prévoit le dépassement des 6 milliards de dollards en 2008.

D'après Gartner Group, d'ici 2008, 60% des entreprises opéreront leurs applications métiers par le biais d'une architecture AOS.