UNIVERSITÉ DU QUÉBEC À MONTRÉAL

DÉFINITION DES BASES D'UN GUIDE SUR L'ARCHITECTURE ORIENTÉE SERVICE

« En corrélation avec le guide SWEBOK »

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

MARTIN-PIERRE DUMOUCHEL

JUIN 2008

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.01-2006). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

AVANT-PROPOS

Intérêts personnels de l'auteur

L'auteur est un informaticien qui a été formé au niveau théorique par l'UQÀM, mais au niveau pratique par IBM. Son intérêt découle directement de son expérience associée aux solutions EAI/AOS de IBM et de ses compétiteurs (i.e.: webMethods, BEA, Microsoft, etc.). Il était, au moment de la rédaction de ce mémoire, chez Hydro-Québec à titre d'architecte technologique et de *lead* développeur dans l'unité d'informatique en mesurage et relève. Il a travaillé dans le groupe *"logiciels IBM"* de 2000 à 2001. De plus, il a débuté sa carrière de consultant dans une petite équipe, chez IBM, qui répondait au nom de "*affaires électroniques et intégration*". Lors de son passage (2001-2005) dans cette équipe, il a eu la chance de participer à des projets en clientèle ayant directement trait avec des technologies découlant de l'AOS et de l'EAI. [GAVI2003], [COXP2000], [WHAL2003], [WACK1999]. De là provient son intérêt pour cette sphère d'activité du domaine informatique. Il est certain qu'à titre d'ancien *IBMiste*, il a probablement une vision quelque peu biaisée au niveau technologie. Ce biais, il tentera de ne pas le faire transparaître dans cet ouvrage, voire de l'effacer. Pour ce qui est de son intérêt pour le SWEBOK, il a été développé par la lecture et l'approfondissement de ce document dans le cadre de ses cours de maîtrise à l'ÉTS et à l'UQÀM. Il a apprécié la tendance à l'amélioration continue qui caractérise ce genre d'ouvrage. L'auteur a donc décidé de joindre ces deux intérêts personnels afin d'en générer un mémoire de maîtrise que voici. On se demandera probablement pourquoi l'AOS? C'est probablement lié au cheminement de carrière de l'auteur. Très tôt, il a été confronté à des technologies à très forte tendance d'intégration de système. Alors, il a vite développé un intérêt pour ces technologies. Mais en plus, il a vu,

comme tous, l'industrie informatique se tourner vers ces technologies d'intégration de système. En plus d'avoir développé un intérêt, il a aussi été parachuté à plus d'une reprise dans un environnement où l'intégration de système était au cœur de l'action. L'auteur a donc été forcé de réagir, comme les autres, de s'adapter, etc.

Objectifs Généraux du mémoire

En jetant les bases de ce mémoire, des objectifs généraux ont servi à l'orienter. Tout d'abord, un premier objectif général est de déterminer la portée précise de l'ouvrage. C'est-à-dire qu'il faut que le lecteur comprenne la portée du document. Ensuite, le public cible a été pris en considération quant à la formule de présentation à choisir. Un style plus littéraire que technique a été choisi, selon les recommandations du directeur de recherche, Dr. Ghislain Lévesque, professeur au département d'informatique de l'Université du Québec à Montréal. Il faut donc que le document s'adresse bel et bien à son public cible. Puis, et non le moindre, l'identification du sujet lui-même découle de l'objectif voulant que ce mémoire soit avant tout un document de référence (ou du moins, humblement, les bases d'un éventuel document de référence) sur l'architecture orientée-service.

À qui s'adresse cet ouvrage

Le public cible a influencé l'orientation du contenu et de la présentation de cet ouvrage. Le public cible comporte l'ensemble des membres d'une équipe de développement et/ou de maintenance. De plus, certains membres de la communauté enseignante en informatique peuvent y trouver intérêt. Il a donc fallu adapter le

contenu à un public cible plus large que seulement une communauté de développeurs. C'est donc grâce à cet objectif que cet ouvrage a pris une tangente plus littéraire quant à la présentation du contenu, plutôt que de demeurer un document purement technique. Ainsi tous les membres d'une équipe de développement et/ou de maintenance peuvent se référer, au besoin, à cet ouvrage. L'enseignant ou l'étudiant, le chef de projet, en passant par l'architecte, l'analyste, le développeur et le testeur; Cet ouvrage demeure assez général pour accommoder tous ces styles de lecteurs éventuels.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS ET ACRONYMES

AOS	Architecture orientée-service
EAI	Enterprise application integration
EJB	Enterprise java bean
ETL	Extract, transform and load
FTP	File transfer protocol
J2EE	Java 2 enterprise edition
JMS	Java messaging service
MDB	Message-driven bean
MQ	Message queuing
RMI	Remote method invocation
RPC	Remote procedure call
RUP	Rational unified process
WS	Web services

RÉSUMÉ

Ce mémoire aborde un sujet du domaine de l'intégration d'applications d'entreprise qui est basé sur la notion de services. Le nom exact de ce domaine est *architecture orientée-service*. L'architecture orientée-service est abordée par ce mémoire comme étant une nouvelle approche de l'intégration d'applications, qui est en soit un problème qui date de plusieurs années déjà. Des survols historiques et théoriques sont donnés afin de fournir l'état de l'art, en plus d'approfondir la matière avant d'entrer dans le cœur du sujet qui est, comme le titre l'indique, les bases d'un guide sur l'architecture orientée-service basé sur le SWEBOK. Pour ce faire, le chapitre 4 est divisé en sections faisant directement référence au SWEBOK mais où l'on retrouve plutôt des contributions de l'auteur sur chacun des domaines abordés. Une suite d'objectifs est donnée en début de chapitre 4. Les contributions doivent répondre à un ou plusieurs de ces objectifs. Aux contributions, sont donc directement attribués des objectifs, ainsi qu'une courte explication qui décrit comment chacune des contributions répond aux divers objectifs. Une analyse sommaire des relations entre contributions et objectifs suit afin de déterminer si des hypothèses et généralisations découlent de cet exercice. La conclusion porte, d'une façon générale, sur l'ensemble du mémoire, mais aussi principalement sur le contenu du chapitre 4.

Mots-clés: architecture, orientée-service, architecture orientée-service, intégration d'applications d'entreprise, conception d'application, services web, conception de logiciel, swebok, génie logiciel, soa

INTRODUCTION

1. Le mémoire

L'architecture orientée-service est un vaste sujet tant commercial que technique. Dans une vision des sciences de la gestion ou d'études commerciales, on pourrait remarquer qu'il s'agit d'un phénomène à part entière de l'industrie informatique, de la même ampleur que l'intégration d'applications d'entreprise par exemple, ou encore que le paradigme client-serveur. Avec une vision plus scientifique de ce sujet, on peut y trouver un grand intérêt tant le côté technique est intéressant, voir bouleversant pour le domaine de l'informatique.

Ceci étant dit, il a fallu concentrer la vision, focaliser vers un ensemble d'objectifs clairement définis et réduire la portée de l'étude pour finalement produire un livrable qui formerait un mémoire de maîtrise en science informatique avec spécialisation en génie logiciel.

Le résultat visé est la base d'un ouvrage de référence utilisable par une équipe de développement et/ou de maintenance d'un système à architecture orientée-service. Le but n'est pas d'étudier à fond le côté technique et scientifique de l'architecture orientée-service, ou d'une de ses implémentations, mais plutôt de créer les bases d'un guide de génie logiciel à saveur architecture orientée-service.

À titre d'ouvrage de référence, une mise en situation historique sur l'architecture orientée-service sera faite pour le lecteur, ainsi qu'une revue théorique sur l'architecture orientée-service (i.e.: les différentes techniques par exemple), avant de finalement entrer dans le cœur du sujet, c'est-à-dire de situer le guide SWEBOK

[ABRA2004] par rapport au domaine de l'architecture orientée-service. Selon cette approche, il serait utile au lecteur de prendre connaissance du guide SWEBOK afin de mettre en perspective cet ouvrage de sorte à en tirer le maximum.

2. Objectifs du mémoire

Dans un contexte plus formel, voici les principaux objectifs du mémoire, qui reprennent certains aspects présentés dans la section précédente.

- Faire un état de l'art sur l'AOS

Tout d'abord, le sujet principal, soit l'architecture orientée-service, doit être traité en effectuant un état de l'art. Ainsi le chapitre I (l'AOS un survol) et II (Aspects Théoriques) devront y répondre.

- Présenter l'évolution des technologies d'intégration de système

Le chapitre III (Situation Historique) propose une réponse à ce deuxième objectif ayant pour but de présenter l'évolution des techniques et technologies d'intégration de systèmes. L'aspect historique va de pair avec l'état de l'art mais a, en plus, l'avantage d'approfondir le passé. Il est important de connaître l'aspect historique pour comprendre les racines de l'état actuel.

- Proposer un guide d'application méthodologique et les bonnes pratiques pour assurer le plein potentiel d'exploitation de cette technologie

Le chapitre IV (L'AOS en corrélation avec le SWEBOK) a pour but de proposer un guide, ou du moins une esquisse d'un éventuel guide de référence d'application méthodologique et les bonnes pratiques de l'AOS.

- Procéder à une première validation de ce guide et en déterminer les limites

Afin de valider la qualité des réponses aux objectifs précédents, ainsi que d'en déterminer les limites, une technique de validation du mémoire est défini au chapitre V (Validation).

- Indiquer les étapes et moyens pour en assurer la diffusion et l'évolution

La conclusion amène des perspectives d'utilisation d'un tel guide et des suites à donner afin d'indiquer les étapes et moyens pour en assurer la diffusion et l'évolution.

3. Comment est organisé cet ouvrage

Cet ouvrage est organisé en préface, une introduction, suivie de chapitres. L'introduction introduit le mémoire, en y citant par exemple les objectifs généraux. Le premier chapitre (L'AOS, un survol) a pour but d'introduire le sujet principal de ce mémoire, soit l'architecture orientée-service. Le deuxième chapitre (Aspects Théoriques) définit les grandes lignes théoriques derrière l'architecture orientée-service, sans pour autant s'y attarder longuement ni en détail. Le troisième chapitre (Situation Historique) couvre assez largement la mise en situation historique et l'évolution de l'intégration de systèmes jusqu'à l'AOS. Ensuite, ce même chapitre rapporte certaines des difficultés rencontrées dans le domaine de l'intégration de système à travers le temps et les perspectives qui se dégagent à l'heure actuelle. Le

quatrième chapitre (L'AOS en corrélation avec le SWEBOK) est le cœur du sujet de ce mémoire. Les justifications, objectifs et contributions d'un guide sur l'AOS sont avancés. On y retrouve aussi une section qui explique comment est organisé le chapitre de façon plus détaillée Le cinquième chapitre porte sur la validation en expliquant la méthode utilisée afin de valider cet ouvrage. Une conclusion suit et termine le contenu proprement dit. On y retrouve les références et les annexes en fin de document.

CHAPITRE I

L'AOS, UN SURVOL

Introduction

Voici la définition de l'AOS par l'encyclopédie des internautes www.fr.wikipedia.org :

"L'architecture orientée services (calque de l'anglais Service Oriented Architecture, ou SOA) est une forme d'architecture de médiation qui est un modèle d'interaction applicative qui met en œuvre des services (composants logiciels)."

Mais, on peut aussi avancer que l'architecture orientée-service est une évolution normale de l'architecture client-serveur en conjonction avec le domaine des systèmes répartis. Mais c'est avant tout une solution au problème de l'intégration des systèmes. Ce mémoire porte donc principalement sur l'architecture orientée-service en se penchant sur le problème du manque de support au développement de telles architectures. Il est important de comprendre que l'AOS n'est pas une technologie, mais bien une approche architecturale. De plus, bien que le nom et la prise de conscience de cette approche soit récente, cette approche existe depuis longtemps, mais n'était pas clairement définie ni même reconnue par l'industrie. L'émergence des services web a donné beaucoup d'importance à ce type d'architecture et a aussi donné lieu à un éventail incroyable d'ouvrages de référence sur le sujet. Les références sont présentées à la fin de cet ouvrage. Ce mémoire se penche sur les services web, mais ils sont traités comme une tangente, bien que principale, de l'architecture orientée-

service et non comme étant le centre et le cœur du sujet. Il tente donc d'apporter un certain support au développement orientée-service en se fondant, au niveau de la structure et du contenu, de ce guide sur le génie logiciel qu'est le SWEBOK (*software engineering body of knowledge*).

1.1 Le défi de l'intégration

Voir la note de bas de page. [1]

L'intégration de systèmes hétérogènes est un vieux problème causant beaucoup de soucis financiers et des tracas opérationnels aux entreprises exploitant des technologies de l'information. La très grande majorité des entreprises possèdent plus d'un système d'information et bon nombre d'entre elles doivent éventuellement implémenter un ou des niveau(x) d'intégration entre ces systèmes. Parfois au niveau des données seulement [DENG2005], d'autres fois au niveau des transactions applicatives [NORT2007], l'intégration de systèmes demeure lourde, complexe et difficile.

1.1.1 Les causes

1

N.B.: Cette section du mémoire est une introduction au sujet et est traitée d'une façon plus personnelle de la part de l'auteur que le reste du contenu. C'est donc une forme plus libre et avec moins de références que le reste du contenu de ce document.

Les données, leurs sémantiques, l'utilisation différentes qu'en font les divers systèmes, ainsi que les natures technologiques différentes sont toutes des causes potentielles rendant lourde, complexe et difficile l'intégration de systèmes. On pourrait deviner ces causes à partir des technologies offertes et couramment utilisées dans le marché, parce qu'elles tentent toutes de résoudre un certain ensemble de problèmes. En identifiant les problèmes soulevés par une technologie d'intégration de système, il est facile d'en déduire les causes. Par exemple, les produits de ETL (*Extract Transform & Load*) [DENG2005] résolvent quelques problèmes en commençant par la divergence des schémas de base de données entre des systèmes devant, en principe, partager les dites données. Or la cause apparaît lorsqu'on se demande, dans ce cas précis, pourquoi les schémas de base de données sont-ils différents entre des systèmes devant, en principe, partager ces données? Les quatre causes énumérées en début de paragraphe répondent toutes ensemble à cette question.

1.1.2 Les manifestations

Une première manifestation se situe au niveau des données et de leur sémantique, le problème de base est que les schémas des bases de données sont créés dans le cadre de systèmes donnés, à un temps donné, indépendamment l'une de l'autre.

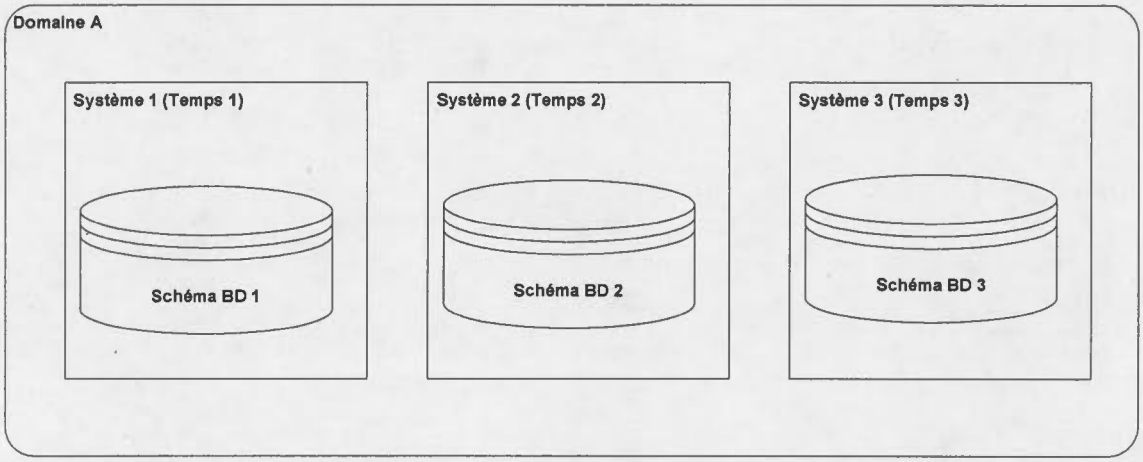


Figure 1 - Schémas BD différents à un temps donné pour un même domaine d'application

Dans la figure ci-haut, trois systèmes créés à trois temps différents ont un schéma BD qui diffère d'une façon ou d'une autre pour un même domaine d'application.

On peut alors remarquer que différents systèmes d'un même domaine d'application [ABRA2004] traitent parfois des données semblables, similaires, parentes ou complémentaires en utilisant des schémas de données précis et donc différents les uns des autres.

En prenant un domaine d'affaire quelconque où l'on retrouve plus d'une base de données (associées à différents systèmes les exploitants), on remarquera qu'en pratique ces bases de données n'ont pas été créées en tentant de préserver un schéma de données global, mais plutôt ont été créées dans le but de répondre aux besoins d'un système précis, qui lui a le mandat de répondre aux besoins précis d'utilisateurs du domaine. La résultante est un ensemble de schémas de base de données présentant des caractéristiques hétérogènes à travers un même domaine d'application. Ceci amène par contre l'idée du *"méta-schéma de donnée"* qui permettrait peut-être

idéalement de réduire les différences entre les schémas implémentés pour un même domaine d'application.

Donc, les manifestations de difficultés d'intégration et de réutilisation qu'on peut attribuer aux utilisations différentes que font les applications des données d'un même domaine d'application sont grandement accentuées par le fait qu'il existe des différences entre les schémas. C'est-à-dire qu'en plus des utilisations différentes que font les systèmes des données d'un domaine, **ces mêmes données diffèrent littéralement** d'un système à l'autre (de façon générale), soient par le biais d'anomalies, ou pour toutes autres raisons.

Voici un exemple de manifestation de difficultés d'intégration et de réutilisation que l'on peut attribuer aux utilisations différentes que font les applications des données:

- Soit un domaine d'applications données pour lequel plus de deux besoins distincts d'utilisateurs, qui doivent nécessairement mener à deux systèmes distincts (ex: un système de facturation et un autre d'acquisition pour le domaine d'application ferroviaire); on retrouve normalement deux utilisations différentes des données potentielles du domaine (ex: identifiant unique d'un wagon). La première, pourrait utiliser les données du domaine ferroviaire pour créer et gérer des factures, ajoutant par le fait même des transactions et des données propres à ces activités, et la deuxième pourrait faire de même, c'est-à-dire ajouter des transactions et des données propres à ces activités d'acquisitions pour le domaine ferroviaire. Le tout en créant des disparités au niveau des données. Par exemple, l'identifiant unique d'un wagon peut être basé sur la même idée fonctionnelle, entre deux schémas, mais dont l'implémentation diffère, et donc une disparité évidente apparaît au niveau des valeurs d'un même champ fonctionnel (l'identifiant unique d'un wagon) d'un domaine d'application. Dans cet exemple, l'un pourrait utiliser la

codification américaine et l'autre, celle du fabricant. On pourrait même aller plus loin en supposant qu'au moment de mettre en œuvre ces deux systèmes fictifs, les standards quant à ce champ de ce domaine d'application, ont tout simplement changés. Or lorsque vient le temps d'intégrer ces deux systèmes du même domaine, des problèmes d'intégration surgissent liés à cette réalité.

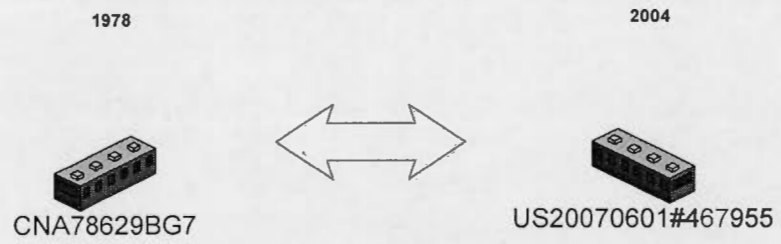


Figure 2 - Disparité dans le temps lié à un domaine d'application qui se reflète dans les schémas BD

L'illustration ci-haut met en relief un exemple fictif où une certaine standardisation d'un domaine d'application change avec le temps et devient une cause du problème de l'intégration entre des systèmes hétérogènes.

Une deuxième manifestation évidente est aussi reliée aux natures technologiques différentes qui peuvent (et le sont très souvent) être utilisées pour répondre aux différents besoins d'un même domaine d'application.

Par exemple, pour un domaine d'application ferroviaire, le système de gestion de facturation est peut-être implémenté sur la centrale IBM et le système d'acquisition peut être implémenté en .Net sur la plate-forme Microsoft. Avec un tel exemple (tout à fait réaliste), on pourrait donc retrouver au moins deux schémas de données distincts, deux utilisations (fonctions) distinctes de données et, finalement, deux natures technologiques totalement différentes.

Les vraies manifestations des problèmes d'intégration et de réutilisation apparaissent lorsque vient le temps d'intégrer ces deux systèmes cités en exemple pour les différences au niveau de leur nature technologique. Diverses méthodes existent bel et bien pour intégrer des environnements de nature technologiques différentes, mais elles demeuraient spécifiques, du cas par cas, avant l'arrivée des notions d'intégration d'application d'entreprise, puis d'architectures orientées-service.
Un autre exemple peut être plus précis. Soit un système A qui est fait en .Net et qui encrypte des données avant de les entreposer dans une BD, à l'aide d'une librairie propriétaire et non standardisé (.Net). Lorsque vient le temps (plus tard) de réutiliser cette donnée à partir d'un autre système existant (Java), il n'y a pas de librairie compatible standardisée et documentée pour décrypter correctement les données. Des coûts supplémentaires de modifications du premier système ne sont alors pas réellement évitables dans le but d'intégrer ces deux systèmes au niveau des données. Sinon, des coûts plus grands encore seraient probablement nécessaires pour arriver à décrypter ces données.

Figure 3 - Autre exemple de manifestation liée aux natures technologiques différentes

1.2 Les perspectives de solutions

1.2.1 Intégration des données

L'intégration au niveau des données donne lieu à diverses tendances telles que les technologies E.T.L. (*extract transform & load*), ou à des solutions de convergence des données de SGBD hétérogènes en un centre logique commun d'outils tels que *DB2 Information Integrator*. Elles ont tentées d'apporter des réponses avec un certain succès.

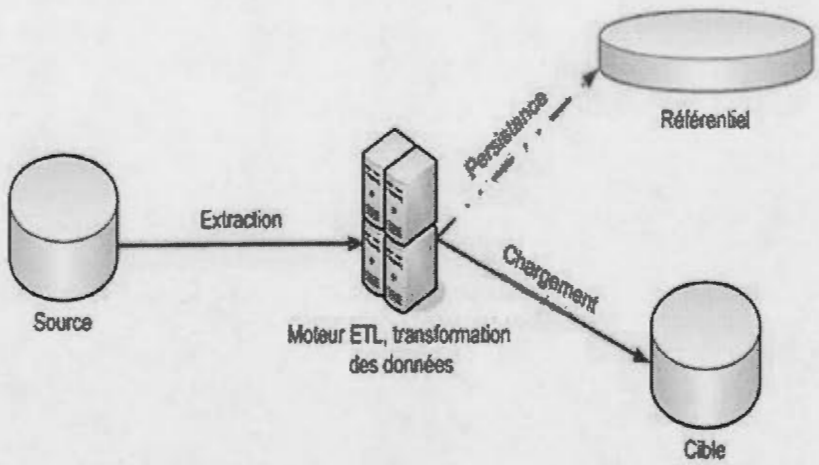


Figure 4 : Exemple d'architecture d'un outil ETL moderne

Dans l'illustration ci-haut, un système A utilisant une certaine base de données sera interprété comme étant la source. Un autre système (existant ou futur) sera quand à lui lié à la base de données que l'on peut interpréter comme étant la cible. Le moteur ETL a pour but d'exécuter l'extraction des données cibles, la transformation voulue et, finalement, le chargement des données cibles. Le référentiel ici n'est qu'un artifice et est propre au moteur ETL seulement. Ont peut aussi comprendre qu'initialement une technique ETL n'est pas vouée à être utilisée en mode transactionnel, ni en ligne ou à la demande, mais bien dans le cadre de travaux de transformation de données afin de fournir et charger des données à un système cible.

1.2.2 Intégration des applications par les données

Il y a aussi des outils d'intégration d'applications d'entreprise tels que *Crossworlds* qui sont très orientés sur les données dans le but d'intégrer des systèmes d'information. Dans ces créneaux, on pourrait aussi situer toutes les intégrations *maisons* qui ont finalement résolu le problème en accédant directement aux données des systèmes visés.

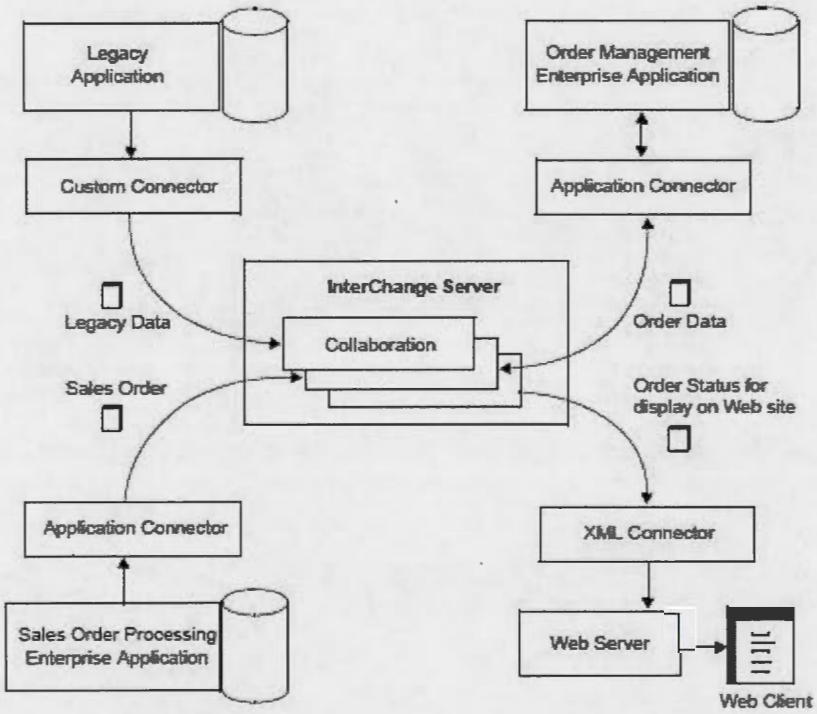


Figure 5: Architecture du produit Crossworlds avant l'achat par IBM (Diagramme de Crossworlds)

Dans l'illustration précédente, un certain ensemble de systèmes d'autant de domaines d'application sont intégrés ensemble au niveau des données par ce système Crossworlds . Contrairement à l'approche ETL, cette approche a pour but d'être fonctionnelle en mode transactionnel, en ligne et à la demande. Pour être plus précis quand à cette implémentation particulière, il s'agit de la mise en place et de l'utilisation de connecteurs qui servent à interagir avec chacun des 4 systèmes à intégrer. Le centre peut être considéré comme étant le moteur du système. Il effectue les transformations des données à l'aide de cartes de traduction d'un format vers un autre (*map*) servant à transformer les données d'un format propriétaire source à un format générique, puis de ce même format générique à un format propriétaire cible. Le tout pouvant être appelé une interface.

1.2.3 Intégration par fonctions de programmation distantes

D'application à application directement, l'intégration est aussi possible. Par exemple, il existe des connecteurs utilisés dans un environnement donné servant à se *connecter* fonctionnellement à des systèmes externes (ex: *connecteur SAP R/3* dans *WebSphere Application Server*).

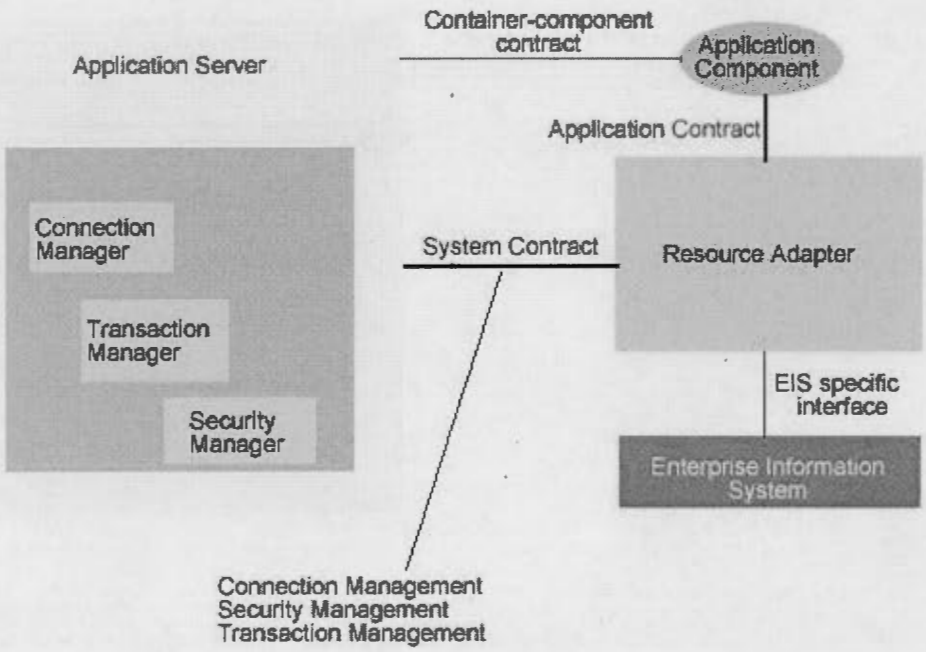


Figure 6: J2EE Connector Architecture (Diagramme de Sun Microsystems)

Dans cet exemple, il s'agit de la définition de la spécification servant au fabriquant de serveurs d'application Java EE afin qu'il puisse, d'une façon standardisée, construire l'architecture pouvant faire fonctionner d'éventuels connecteurs de systèmes d'entreprise. Le but ultime est de créer une infrastructure standardisée permettant

d'intégrer des technologies autres (ex: SAP, Siebel, etc.), avec des composantes applicatives Java EE.

Bien sûr d'autres exemples existent. D'ailleurs cette idée a même été reprise pour arriver à d'autres solutions d'intégration.

1.2.4 Intégration des applications par le biais d'outils middleware

Avec la venue de gestionnaires de transactions sur la centrale (*mainframe*) tels que CICS et le développement des applications client-serveur sur NT et Unix, et encore plus avec la venue des applications web, les systèmes orientés messagerie sont apparus comme des solutions extrêmement utiles. Pour ne donner qu'un seul exemple, *MQSeries* d'IBM est un outil largement utilisé dans les entreprises afin d'améliorer la communication entre les systèmes hétérogènes. La beauté de cette solution ne réside pas dans le rôle de cet outil, ni dans le niveau de qualité de son implémentation, mais plutôt dans la disponibilité d'interfaces hétérogènes allant de .Net à J2EE, en passant par C++ et Cobol. De plus, il faut ajouter que le nombre de plates-formes supportées par cet outil aide grandement à le rendre indispensable dans l'intégration de systèmes hétérogènes. Par contre, due à sa simplicité et à sa portée, l'efficacité de son utilisation demeure très liée à la qualité de la conception et de l'implémentation des couches logicielles *maison* qui auront été déployées afin d'intégrer les systèmes visés. Le niveau de réutilisation des couches logicielles ainsi développées est faible par contre. Il faut mentionner ici, l'apport incontesté et inestimable des gestionnaires de transactions tel que CICS d'IBM ou encore TUXEDO de BEA Systems dans le domaine de l'intégration des systèmes hétérogènes. D'autres outils centraux ne sont en fait que des extensions d'autres, tels que les systèmes orientés messagerie. On peut

donc prendre en exemple *MQSeries Integrator* qui représente un outil beaucoup plus riche se basant sur les qualités d'un système orienté messagerie. D'autres outils centraux sont plutôt basés sur l'orientation *"appel de procédures distantes"* en utilisant les RMO ou RPC par exemple, ou encore les sockets. Ainsi *Microsoft Transaction Server*, les serveurs CORBA, les serveurs de services web, ou encore, les serveurs de *beans* d'entreprise (J2EE), sont des environnements centraux qui permettent le déploiement et l'exécution d'objets/services dans le but d'intégrer et de réutiliser les objets/services. Dans cette veine, la notion de bus d'entreprise tente sa chance dans le domaine de l'intégration de système et de la réutilisation de fonctions. On voit donc des technologies telles que *Web Method, MQSeries Integrator et WebSphere Business Integration Foundation Server*, ainsi que d'autres serveurs basés sur la notion de BPEL (*Business Process Execution Language*) tenter leur chance.

De tous ces outils centraux, la plupart de ceux qui persistent en 2006 sont basés soit sur les services web (incluant le BPEL) ou sur des systèmes orientés messagerie.

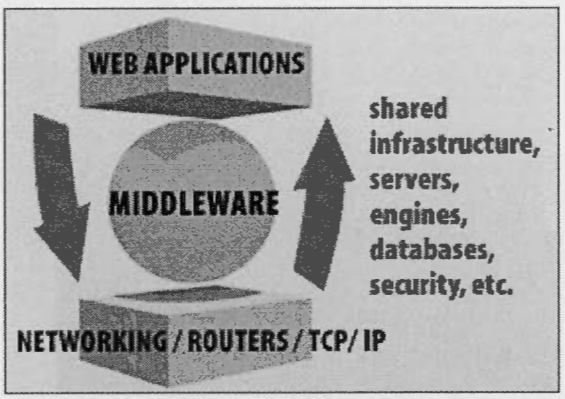


Figure 7: Approche Middleware (Diagramme de doit.wisc.edu)

En conclusion, dans l'illustration ci-haut le *middleware* est une solution qui agit entre les systèmes à intégrer de sorte à supporter les difficultés d'intégration à l'aide de divers outils qui diffèrent d'un type de *middleware* à l'autre.

Dans cet exemple, il faut considérer que la composante du haut (*Web Applications à titre d'exemple*) utilise le middleware afin d'encapsuler les difficultés techniques afin de communiquer avec un éventuel deuxième système (qui n'est pas illustré).

CHAPITRE II

ASPECTS THÉORIQUES

L'architecture orientée-service tente d'apporter des notions théoriques à l'intégration de systèmes dans le but de régir ce domaine pour en améliorer les activités, et surtout la qualité des résultats. Voici tout d'abord les aspects distinctifs de l'AOS, puis certaines de ses limites. Il existe maintes définitions fort intéressantes mais différentes de l'AOS. Ici, une définition proprement dite, n'est donc pas proposée. Plutôt, voici les aspects distinctifs de l'AOS retenues à titre de description théorique.

2.1 Aspects distinctifs de l'AOS

2.1.1 Forte cohérence interne

L'architecture orientée-service se distingue, du modèle RPC par exemple [NATI2003-A], par une plus grande emphase sur la cohésion interne de chacun des services. Un service basé sur l'architecture orientée-service, doit être implémenté pour un client immédiat avec l'optique que d'autres clients, inconnus au départ, devront aussi utiliser ce service. La cohésion interne doit être très forte pour maintenir la "*structure*" [NATI2003-A] fonctionnelle du service à travers le temps. Il est donc très important d'analyser le but et le besoin fonctionnel exact auquel le service doit répondre. On comprend donc que l'architecture orientée-service est à un niveau d'abstraction plus élevé que des techniques d'intégration de bas niveau comme le modèle RPC par exemple.

2.1.2 Couplage externe faible

Le couplage entre le consommateur (le composant qui consomme le service) et le producteur (le service) doit être limité à un interface *de consommation,* le plus général et le plus paramétrable possible dans le but de réduire au maximum le couplage entre le service et son consommateur. Si l'interface est fortement couplée, le consommateur qui l'utilisera, d'une façon ou d'une autre, sera tout aussi fortement couplé au service par le fait même, d'où le lien entre couplage/découplage et les notions de séparation d'interfaces et d'implantation. La notion de surcharge (principe de reprendre la signature d'une fonction pour différents paramètres d'entrées) peut être utilisée pour modifier un service lorsque de nouveaux paramètres d'entrée et de sortie sont ajoutés. Si l'analyse de départ est assez précise et élaborée, une seule interface devrait suffire à couvrir tous les besoins des consommateurs au cours du cycle de vie du service. Il est à noter que certaines technologies AOS limitent, à prime à bord, la surcharge des services [BOOT2004], [NATI2003-A].

2.1.3 Modularité

La notion de modularité dépasse les bornes de l'architecture orientée service. Par contre on peut noter que les distinctions propres à cette notion au niveau de l'architecture orientée service s'expriment par une tendance à *modulariser* les fonctionnalités en un ou des service(s).

2.1.4 Service abstrait versus agent concret

L'architecture orientée-service tend à séparer les fonctionnalités abstraites d'un service quelconque de l'implantation concrète par la description de l'interface. Or, une technologie basée sur l'architecture orientée service offre normalement une solution afin d'identifier, de façon abstraite, l'interface d'un service autre que par l'implantation

elle-même des services [MURP2004].

2.1.5 Notion de centres dans un processus plus large

Le service encapsule des fonctionnalités qui représentent des centres de valeurs pour une organisation. Dans un processus plus large, ces centres de valeurs peuvent devenir des nœuds dans un processus, dans un flux. C'est-à-dire qu'il est possible de chorégraphier les services afin d'utiliser les divers centres de valeurs d'une entreprise afin d'élaborer des applications [ALEX1996], [MURP2004].

2.1.6 Réutilisation du service

La réutilisation des fonctionnalités est un des buts principaux de l'AOS. Par contre, ce n'est pas l'AOS qui a amené ce besoin, cette idée. C'est donc une caractéristique héritée d'autres courants antérieurs du génie logiciel [NATI2003-A].

2.1.7 Notion de description d'un service (ou contrat)

La description de l'interface d'un service est à la base de l'AOS. Le contrat est ainsi défini entre le consommateur et le producteur de service. Cette description comprend les notions d'adressage, de nom de fonction, de structures de données en entrée et en sortie, et d'autres détails servant à découpler les détails d'implantation [CHAV2004].

2.1.8 Généralisation et ré-utilisation

Le but principal de la mise en place d'un service (composante logicielle implémentant une fonction d'affaire cohérente de façon à maximiser sa réutilisation et à diminuer au maximum le couplage entre le client et le service) est de permettre d'atteindre une

certaine généralisation suffisante à la réutilisation du service par plus d'un client potentiel, d'où le bas couplage externe, la forte cohérence interne et finalement, la modularité [NATI2003-A].

2.1.9 Découvertabilité (...qui est découvrable...)

Il s'agit d'un aspect distinctif apporté par la technologie UDDI (*Universal Description Discovery and Integration*) et principalement utilisé dans le cadre des services web. Il s'agit d'une base et d'un service de découvertabilité qui répertorie l'ensemble des services disponible sur un réseau. Cet aspect distinctif est discutable, mais à cause de la portée du UDDI dans les services web, il est noté ici.

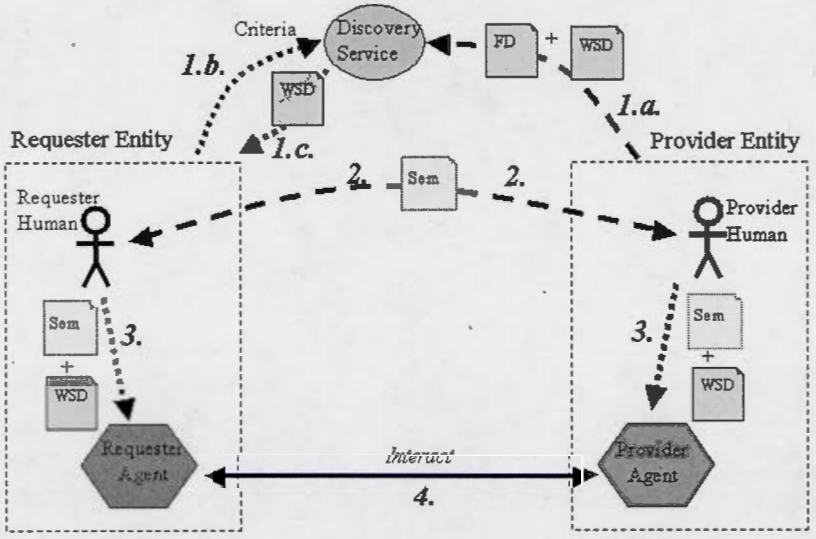


Figure 8 : Notion de découvertabilité d'un service (diagramme du w3c, www.w3.org)

2.2 *Limites*

2.2.1 Impact d'une panne d'un service

La notion de fournisseur de service dans n'importe quel domaine vient inévitablement avec un certain niveau de responsabilité contractuel à livrer aux *consommateurs*. C'est le même phénomène dans un environnement orientée-service. C'est donc dire qu'un service (producteur), qui livre un *produit informatique* à des consommateurs, a une responsabilité plus grande et plus visible dans l'entreprise qu'une série de fonctions quasi-identiques retrouvées à l'intérieur de différents systèmes [NATI2003-A]. Cette responsabilité est donc exprimée par un besoin plus important au niveau de la stabilité de l'environnement d'exécution d'un service par rapport à un système moyen (niveau d'importance moyen dans l'entreprise) ne fournissant pas de service à d'autres systèmes.

Il faut donc prévoir, même pour un environnement uniquement intranet, un plus grand niveau de criticalité lors de la mise en place d'un environnement d'exécution pour un système orientée-service.

2.2.2 Notion de sécurité

La dimension sécurité des fonctions implémentées en service selon l'architecture orientée-service, est amplifiée par les multiples accès possibles et, surtout, par la conception d'ouverture aux requêtes dans un réseau qui est propre aux systèmes distribués. Par "*conception d'ouverture aux requêtes dans un réseau*", il est entendu que si un service peut être accédé par un consommateur, c'est par le réseau dans la grande majorité des cas. De plus, les méthodes d'accès au réseau sont standardisées ce

qui rend donc la reproduction d'un appel d'autant plus facile. La notion de sécurité devient donc un enjeu des plus importants lorsque l'on parle d'architecture orientée-service. C'est peut-être selon l'orientation technologique que la dimension de sécurité de l'AOS s'ajuste ([COTR2004]).

Prenons par exemple, certains hôpitaux québécois qui utilisent des technologies orientées- service afin de maintenir et distribuer les données ayant trait aux dossiers patients. Il est de toute évidence légalement interdit de permettre la divulgation au grand public de ce genre de données par un hôpital. Il faut donc s'assurer que malgré le fait que les méthodes d'accès aux services soient standardisées et donc facilement reproductibles, les couches de sécurité, elles aussi standardisées, parviennent à neutraliser les accès indésirables. Comme telles, les notions de base de l'architecture orientée-service ne dictent pas de méthodes de sécurité des services. Plutôt, il est remarqué que certains standards naissent des besoins exprimés par l'architecture orientée-service tels les services web du W3C. Par contre l'AOS, lui-même, a été utilisé pour définir des stratégies de sécurité comme il est mentionné et expliqué dans la référence [IMAM2005]. On y voit qu'IBM a utilisé des notions de l'AOS afin d'élaborer sur le sujet. Dans le même ordre d'idées, d'autres besoins plus spécifiques à de tels standards donnent naissance à l'adaptation ou carrément à la mise sur pied de nouveaux standards ou protocoles. D'ailleurs, au sujet des services web, on pourrait noter la réutilisation/adaptation des protocoles de sécurité de la couche de transport (ex: HTTPS ou encore SSL) pour ce qui est de l'encryption des échanges réseaux. Au niveau de l'autorisation et l'authentification, d'autres standards/protocoles tels LDAP sont utilisés pour gérer les accès.

2.2.3 Niveau ou degré d'ouverture technologique

L'architecture orientée-service ne dicte pas le degré d'ouverture technologique en ne dictant pas comment définir des standards d'implémentation de technologies orientées-service tels les services web.

Il en revient donc aux divers groupes de mise en place de technologies orientées-service de déterminer le degré d'ouverture technologique des environnements de développement et d'exécution (plate-forme) orientés-service. Par exemple, une technologique telle que *IBM MQSeries Integrator* [PUTT2000] qui s'appellera plus tard *WebSphere Business Integrator* tendra, avec son propre cycle de vie, à améliorer son degré d'ouverture technologique. En 1999-2000, *IBM MQSeries Integrator* (MQSI) n'est que très peu ouverte (accès aux bases données relationnelles en C, accès aux bases données relationnelles en utilisant un script de développement propriétaire, etc.), malgré une bonne fondation (échange de données interne supportant le XML). Plus tard (2003-2004), cette technologie tendra plutôt vers l'ouverture technologique (Services Web, JDBC, JMS, XML, etc.) afin de permettre une plus grande facilité d'intégration entre les consommateurs et producteurs de services.

La plateforme des services web du W3C [BOOT2004], lorsque implémentée dans un environnement d'exécution tel un serveur J2EE [WHAL2006], a pour but l'ouverture technologique en propageant l'idée que le standard proposé par le W3C est une technologie d'ouverture entre des environnements hétérogènes, ce qui s'avérera vrai. Ce qui est certain c'est que plus le degré d'ouverture technologique est grand, plus le principe de généralisation et de réutilisation, propre à l'architecture orientée-service, est grand.

2.2.4 Limites de l'infrastructure technologique choisie

Toutes technologies a ses limites. Or, dans le domaine de l'architecture orientée-service, les implémentations d'environnement de développement et d'exécution sont bornées par différentes limites telles la performance, le niveau de robustesse, la fiabilité, les outils de sécurité, le degré d'ouverture technologique, etc.

CHAPITRE III

SITUATION HISTORIQUE

Il s'agit ici d'élaborer sur les techniques d'intégration ayant précédé l'AOS avant de situer celles ayant vu le jour dans les dernières années. Elles sont abordées à des fins de situation historique. Ensuite, il s'agit d'une discussion des difficultés rencontrées par les diverses techniques d'intégration, et plus particulièrement celles plus récentes auxquelles ont peut associer l'AOS. Finalement, on y retrouve quelques perspectives de solutions à ces difficultés.

3.1 L'intégration avant l'architecture orientée-service

Avant l'utilisation du terme 'Architecture Orientée-Service', des techniques existaient évidemment afin de permettre à des systèmes d'être intégrés les uns aux autres selon les cas.

Les technologies ou techniques qui suivent ne sont pas nécessairement souvent associées à l'AOS, mais il s'agit de techniques qui ont bel et bien servi, et servent encore pour la plupart, dans le domaine de l'intégration de systèmes. Il est surtout important de comprendre que la plupart des technologies qui servent à intégrer les systèmes hétérogènes d'aujourd'hui ont été amenées, voir inventées, avant la popularisation de l'architecture orientée-service. Voici certaines techniques majeures ayant vu le jour avant l'AOS et qui continueront sans doute à être largement utilisées dans le domaine de l'intégration de systèmes.

3.1.1 Les sockets (TCP/UDP)

L'avènement des sockets a permis à deux processus d'échanger des informations sur deux ordinateurs séparés. Cette technique est donc à la base de tout échange de données entre processus distants. De plus, beaucoup de technologies, voir de produits, utilisent les sockets (en particulier les sockets TCP) [POST2003].

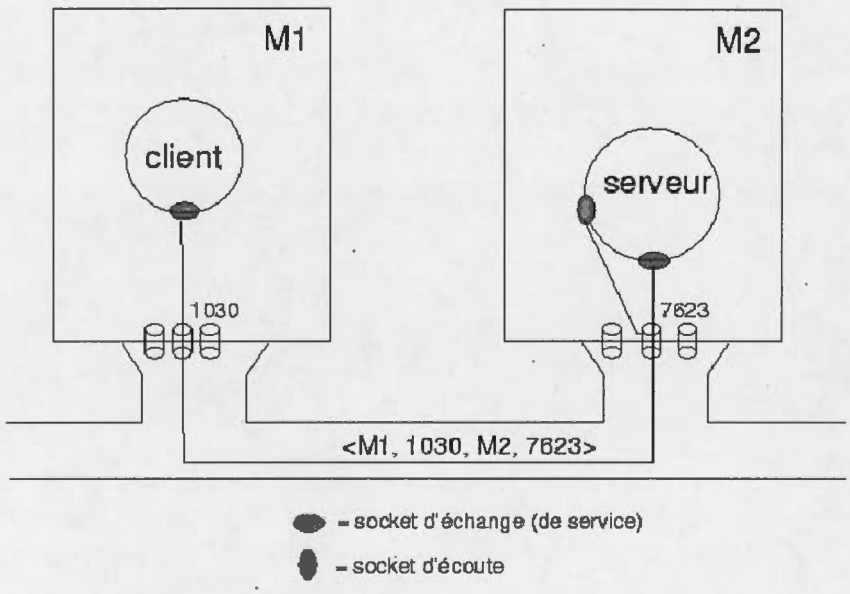


Figure 9 : La communication par socket en Java (diagramme de l'école www.enseeiht.fr)

À partir de la figure 9, le programme client a un socket d'échange qui sera utilisé pour échanger des données avec le programme serveur qui lui a deux sockets. Le premier est utilisé en mode écoute afin de recevoir des demandes de programme client. Le deuxième est utilisé pour échanger des données avec le ou les programmes clients.

La communication par socket est à la base des échanges de données entre des processus sur un réseau TCP ou UDP. Il s'agit donc d'une technique fortement utilisée dans le domaine de l'intégration de systèmes, et donc dans le domaine de l'AOS.

3.1.2 Le FTP (File Transfer Protocol)

Il s'agit d'une solution permettant d'échanger des données entre des machines distinctes sur le réseau. Il s'agit là d'une première technique qui fut utilisée afin de faire communiquer des processus distants ensemble. Cette technique est d'ailleurs encore largement utilisée aujourd'hui [JPOS1985]. Voir la spécification http://www.ietf.org/rfc/rfc959.txt qui date de 1985.

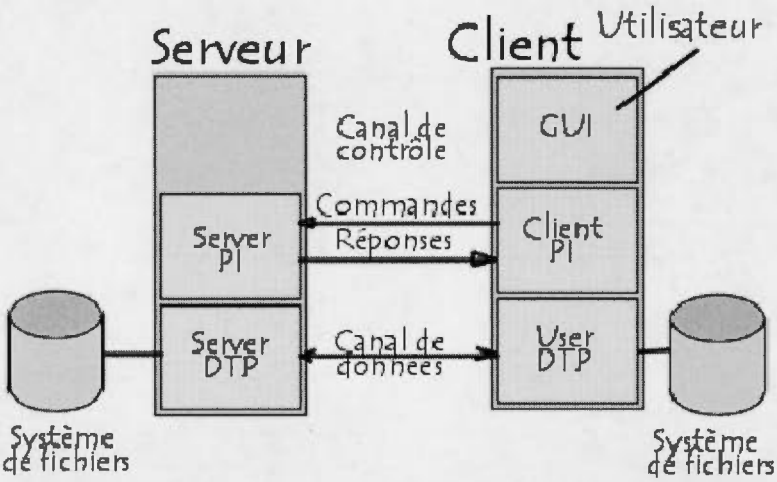


Figure 10: Le modèle FTP (diagramme www.commentcamarche.net)

Dans le diagramme ci-haut, deux canaux sont utilisés. Le premier est utilisé afin de permettre l'échange des commandes et des réponses aux commandes. Le second est utilisé dans le but d'échanger les données. L'utilisateur est normalement au niveau du

client. Le but du FTP est de permettre l'échange de données entre deux systèmes sous forme de fichiers (texte ou binaire), incluant les répertoires.
Bien sûr le FTP est utilisé par des humains, mais il est aussi couramment utilisé par des scripts afin d'élaborer des systèmes automatisés dont les échanges de données se font par l'entremise de FTP. En ce sens, FTP est définitivement utilisé afin d'intégrer des systèmes hétérogènes par le biais d'échanges de données sous formes de fichiers.

3.1.3 Le RPC (Remote Procedure Call)

Le RPC est une technique qui permet à un système client d'interagir avec un système dit serveur. Cette mécanique est à la base de plusieurs techniques/technologies tels le RMI (*Java Remote Method Invocation*) [SUNM1997], ou encore le CORBA (*Common Object Request Broker Architecture*) [COXP2000], etc. [OPEN1997]

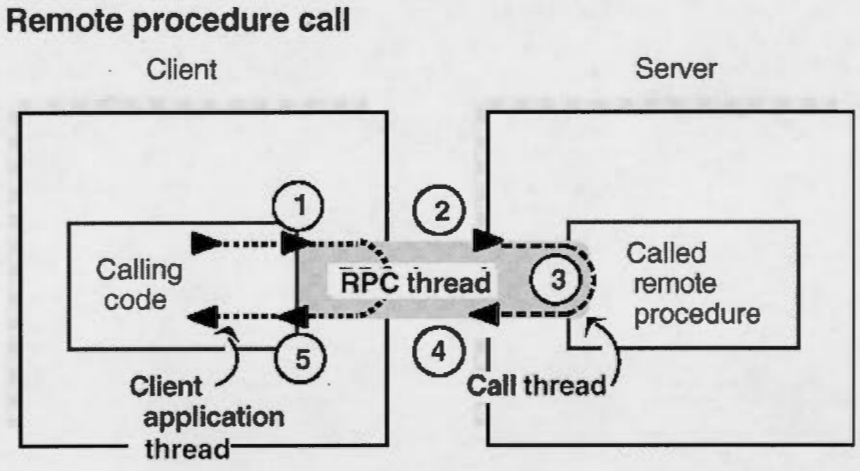


Figure 6-1 Execution Phases of an RPC Thread

Figure 11 - Diagramme du modèle d'exécution du RPC [OPEN1997]

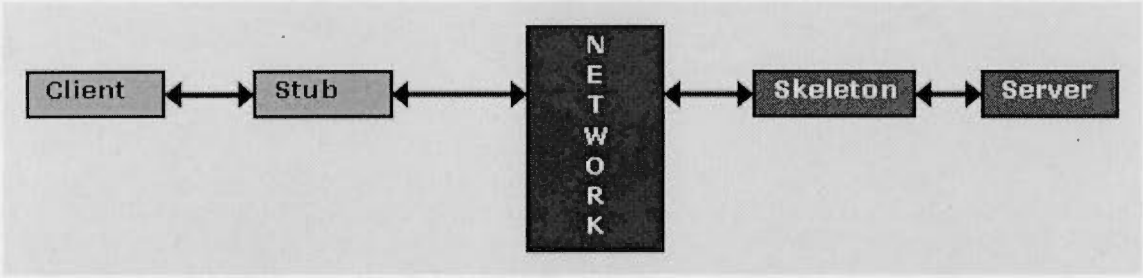


Figure 12 - Diagramme du modèle RMI [SUNM1997]

Dans les deux illustrations ci-haut, un client et un serveur sont en communication dans le but de permettre au client d'appeler une méthode du serveur. Il est aussi à noter que la technique de communication par socket prend toute son importance étant utilisée afin de permettre les communications.

3.1.4 Le CORBA (Common Object Request Broker Architecture)

CORBA sera ensuite introduit afin de définir une méthode orientée-objet permettant de procéder à des *fonctions* distantes (liées au modèle RPC). On peut encore voir ici une solution permettant de faire communiquer ensemble des processus distants. [OBJE2006], [COXP2000]

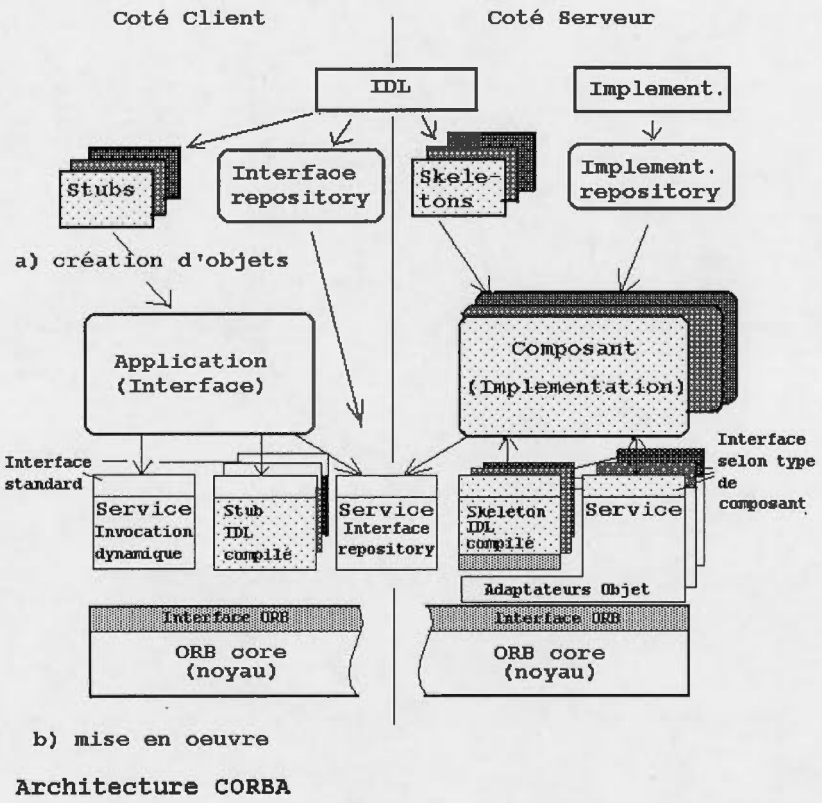


Figure 13 : L'architecture CORBA (diagramme de http://www.pace.ch/cours/cours3/)

Dans le diagramme d'architecture CORBA, ont peut retenir qu'il s'agit d'une technique beaucoup plus complexe que les autres vues jusqu'ici. Il s'agit aussi d'une technique qui respecte beaucoup plus quelques principes théoriques de l'AOS tels que décrit dans cet ouvrage. Que ce soit le couplage externe faible, la modularité, l'idée du service abstrait versus agent concret, la notion de centres dans un processus plus large, la réutilisation, la notion de description d'un service, l'architecture CORBA est un excellent premier effort en ce sens. Cette architecture est d'ailleurs encore fortement utilisée. Par contre, cette technique est plus complexe que d'autres qui ont été proposées par après telles les services web.

3.1.5 MOM (Message-Oriented Middleware)

L'ouverture du mainframe par MOM et le moniteur de transactions permet de mettre sur pied une infrastructure d'entreprise permettant la communication des machines Unix et Windows avec la centrale sous forme de transactions. Par la suite, la technique de files d'attentes de messages sera utilisée pour permettre la communication entre processus sur des plates-formes hétérogènes. Bien que la notion de files d'attentes soit beaucoup plus ancienne que la technologie MOM, celle-ci prend son essor dans les années 90. Mentionnons ici que les moniteurs de transactions (i.e. : IBM CICS ou TUXEDO de BEA Systems) ont joués un rôle important dans l'émancipation des outils MOM. [BEAM2000], [WACK1999], [KEEN2006]

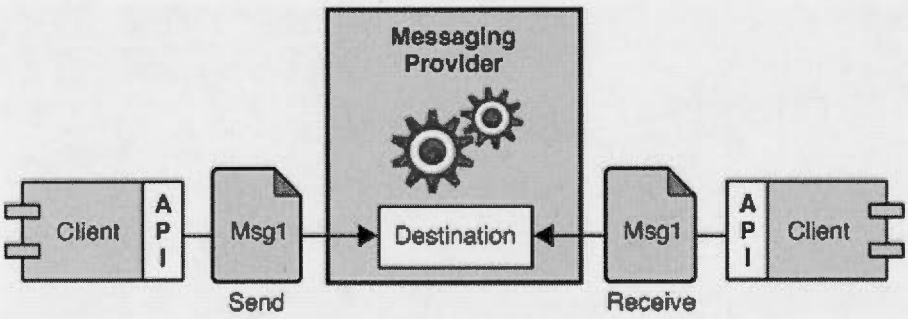


Figure 14: Architecture de systèmes orientés messagerie (Diagramme de Sun Microsystems)

Dans l'illustration suivante, le client gauche communique avec le client droit à l'aide de l'envoi et la réception de messages, le tout géré par le fournisseur de messagerie.

3.2 Pendant la vague de l'AOS

Pendant les premiers balbutiements de l'AOS, d'autres techniques visant l'intégration interprocessus et la réutilisation voient le jour. Celles-ci ne sont pas forcément directement associées à l'AOS. En fait, toutes ces techniques sont valables dans le cadre de l'AOS. En premier lieu, on parlera de EAI en anglais (*Enterprise Application Integration*) [PUTT2000], [GAVI2003], [KEEN2006], puis de services web [ANDE2007], [NORT2007], [BOOT2004], [ROSS2006], [KEEN2006], [KOPE2007], [WHAL2006]. On peut aussi parler, ETL (*Extract, Transform & Load*) qui dérivent de l'EAI mais directement associés aux bases de données. Il est à noter que les techniques ETL existaient bien avant l'AOS, mais ont pris beaucoup d'importance en même temps que l'EAI. De plus des techniques ad hoc on vu le jour telles les EJB MDB (*Enterprise Java Bean Message Driven Bean*) MDB [WHAL2003].

3.2.1 Intégration d'Application d'Entreprise

L'intégration d'applications d'entreprise est implémentée sous forme de *middleware* de deux types reconnus, soit le *hub-and-spoke* et le *enterprise service bus*. [WACK1999], [NORT2007], [GAVI2003]

Hub-and-spoke

Il s'agit de centres (nœuds) d'exécution, soit les *hub*, puis des liens entre ces centres, soit les *spoke* en anglais.

Enterprise Service Bus

Fournit une couche d'abstraction sur l'implémentation d'un système de messagerie d'entreprise qui permet d'exploiter la valeur de la messagerie sans trop de programmation.

3.2.2 ETL (Extract, Transform & Load)

La technique ETL provient du domaine des entrepôts de données, mais a tout de même pris son essor vers le début des années 2000. Le principe est d'extraire des données d'une source quelconque de données, d'appliquer les transformations désirées, puis de recharger ces nouvelles données dans une format quelconque, normalement un système de gestion de bases de données. Plusieurs type de technologies ETL existent et peuvent être utilisées dans le cadre d'architecture orientée-service. [DENG2005]

3.2.3 MDB (message-driven bean)

Les MDB sont une technique mis en route par l'initiative J2EE (Java 2 Enterprise Edition). Ils s'agit d'une composantes technologiques. Elles sont la conjonction entre les EJB (Enterprise Java Bean) et les techniques de messageries, implémentées principalement en Java à travers le JMS (Java Messaging Service. Ces composants

sont particulièrement utiles afin de créer des tâches asynchrones dans le cadre d'un serveur J2EE. [WHAL2003]

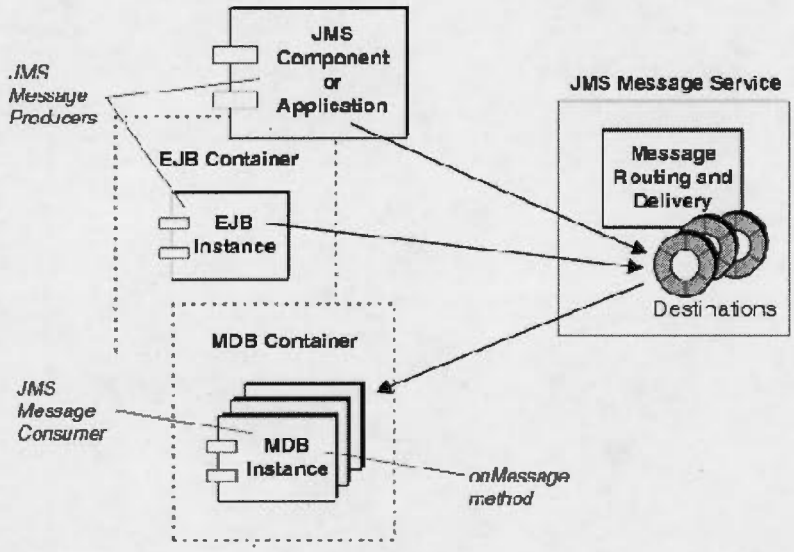


Figure 15 : Architecture du conteneur de Message-Driven Bean (diagramme de Sun Microsystems - http://docs.sun.com/source/819-0069/mq_and_j2ee.html)

Dans le diagramme ci-haut, le service de messagerie (JMS Message Service) reçoit des messages et les envoie aux instances de MDB par l'entremise du conteneur MDB (MDB Container). Les principes de consommateurs et de producteurs de messages font aussi partie de cette architecture.

Cette technique est fortement utile dans les environnements Java pour entreprise (JEE).

3.2.4 Technologie WS (*Web Services*)

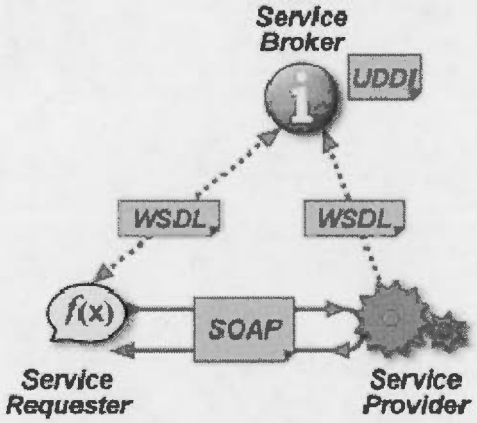


Figure 16 - Diagramme du modèle Service Web

Les services web font leur apparition au début des années 2000 afin de standardiser les techniques de communications entre des processus évoluant sur des environnements hétérogènes distants. Les services Web reposent sur le protocole http (*HyperText Transfer Protocol*) et SOAP (*Simple Object Access Protocol*). [BOOT2004]

Les services web sont sans l'ombre d'un doute le fer de lance de l'architecture orientée-service. Aucune autre technologie n'est aussi fréquemment référencée lorsqu'il est question d'architecture orientée-service. De plus, des fournisseurs, qui normalement ne permettent pas l'interopérabilité de leur plateforme avec d'autres, ont embarqué dans ce mouvement (i.e.: Microsoft).

Diverses technologies tantôt standardisées, tantôt propriétaires ont rapidement

héritées, voir adoptées les services web dans le but de permettre l'échange de données qui a beaucoup de valeur commerciale. Ainsi, les BPEL (*Business Process Execution Language* – technologie standardisée ayant pour but de composer des processus d'exécution utilisant directement d'autres services), les WebMethods avec utilisation des WS, les IBM Business Integration Solutions, etc., ont vu rapidement le jour afin d'attaquer le marché au bon moment. Jamais l'intégration des systèmes n'avait été si omniprésente dans les offres de nouvelles technologies des grands et plus petits fournisseurs de solutions. Ceci nous amène directement à la vague Web 2.0.

3.2.5 Web 2.0 et la promotion du SOA

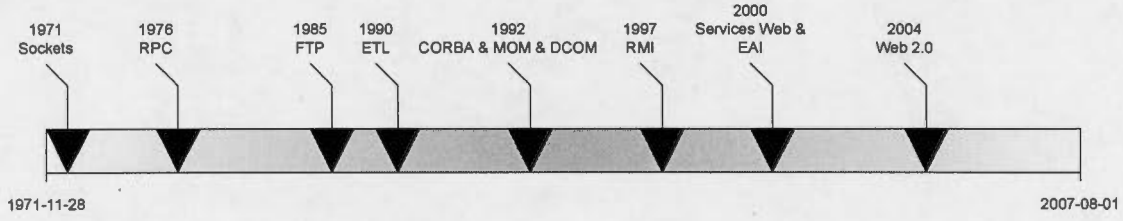
Le Web 2.0, véritable plate-forme informatique à part entière et non seulement une collection de sites web, comprend le SOA (*Service-Oriented Architecture*). Elle donne des ailes à l'AOS pour ainsi éclipser les "anciens noms" tels que EAI (*Enterprise Application Integration*) et simplifier le discours. Le Web 2.0 débute vers 2004. [HINC2006], [OREI2005]

Figure 17 : Représentation de la vague commerciale (illustration de www.journaldunet.com)

Dans l'illustration précédente, on peut par contre comprendre qu'il sera toujours difficile de bien préciser ce qu'est le Web 2.0 tellement la question est large. En fait, toutes les nouvelles technologies susceptibles de faire avancer la cause technologique de l'Internet, font partie du Web 2.0 (ou presque). Dans l'illustration ci-haut, seule une petite partie des technologies faisant partie du Web 2.0 sont en fait illustrées. Il faut comprendre que l'AOS et les WS ne touche beaucoup moins les internautes que les technologies avec lesquelles ils pourront directement interagir (ex: AJAX, les blogs, les wiki, etc.). Mais qu'importe l'AOS fait partie intégrante de cette grande vague commerciale qui déferle sur le marché au moment de la rédaction de ce mémoire. Par son approche *back-end,* elle est moins près du grand public voilà tout. Il faut comprendre que le lien entre l'AOS et le Web 2.0 n'est que commercial dans le but de mousser la mise en marché de quantités d'autres nouvelles technologies impliquées par le Web 2.0.

3.3 *Datation approximative des différentes technologies d'intégration*

Il s'agit ici d'une ligne temporelle exprimant dans quel ordre approximatif les différentes technologies, dites d'intégration, sont apparues. Les références données ici ne comportent pas toutes la date exacte de début d'existence des technologies qu'elles impliquent.



*Références:

Socket	[WINET1971]
RPC	[OPEN1997]
FTP	[JPOS1985]
ETL	[DENG2005]
CORBA	[OBJE2006]
MOM	[BEAM2000], [WACK1999]
DCOM	[GERA1999]
RMI	[SUNM1997]
Services Web	[KEEN2006]
EAI	[PUTT2000], [GAVI2003]

Web 2.0 [HINC2006]

*Il est à noter que le gestionnaires de transactions CICS d'IBM a été mis sur le marché en 1969 et que son ancêtre, le système IMS a été développé dans le cadre du programme spatial Apollo en 1966. Ces techniques, bien qu'anciennes sont encore couramment utilisées par de grandes institutions et banques, par exemple, dans le cadre de systèmes *legacy*.

Voir le dictionnaire en ligne wikipedia pour de plus amples informations à ce sujet (http://en.wikipedia.org/wiki/Information_Management_System).

3.4 Les difficultés rencontrés

Les techniques de l'AOS permettent de résoudre un certain nombre de problèmes de conception, mais on dénote aussi un certain nombre de difficultés lors de leur mise en œuvre.

3.4.1 Responsabilité des livrables et produits

La notion de propriétaire des livrables fonctionnels est difficilement attribuée dans un environnement éclaté où plusieurs groupes de travail se côtoient dans la réalisation de systèmes où l'approche AOS est utilisée.

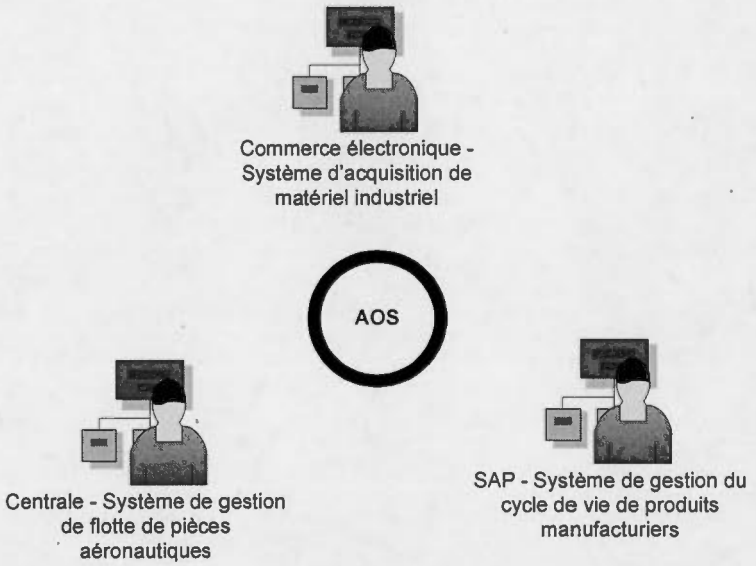


Figure 18: Difficulté d'attribution de la propriété des livrables fonctionnels

Dans le domaine de l'intégration des systèmes, dont l'AOS fait partie intégrante, il est souvent difficile de bien balancer l'attribution de la notion de propriété des livrables puisque l'environnement est typiquement éclaté et couvre souvent plus d'une organisation interne déjà existante et en fonction, en plus, parfois, de nouvelles organisations externes. Ce problème peut s'accentuer au niveau contractuel et légal, voir encore plus, lorsque le système dépasse les bornes de plus d'une entité légale (plus d'une entreprise).

3.4.2 L'approche par cas d'utilisation difficile?

L'approche par cas d'utilisation et ou par unité de tâche ([BOOC1999], [ABRA2004]) est difficilement utilisable dans le domaine de l'intégration de système (i.e.: l'AOS) à cause des mœurs ancrées dans les entreprises. Les cas d'utilisation commencent à peine à vraiment être utilisés globalement qu'il faut s'adapter.

Il ne s'agit pas de cas d'utilisation tel qu'on le connaît, mais bien de composants logiciels inter-système [ALEX1996]. Aussi, l'analyse attribuée à la couche logicielle du type AOS est mise de côté par rapport à ce qui est beaucoup plus évident comme, par exemple, les écrans. De plus, la plupart du temps, les livrables fonctionnels sont liés aux cas d'utilisation, et encore plus précisément aux écrans d'un système. Par exemple, le livrable *P490* de la méthodologie P+ est presque toujours utilisé dans le cadre d'écrans. Or qu'en est-il des services? Il est beaucoup plus facile de ne pas tenir compte de la valeur d'un livrable de la phase d'analyse dans le cadre de l'AOS que dans le cadre d'écrans (interfaces personne-machine).

Bien que l'approche des cas d'utilisation se prête bien à l'AOS, il est ancré dans nos mœurs que les acteurs d'un cas d'utilisation sont plus souvent qu'autrement les utilisateurs et non des consommateurs étant possiblement d'autres services, d'autres systèmes, des gens, etc.

Voici une citation de la définition de 'Cas d'Utilisation' sur wikipedia (en français en date du 2007-04-04):
"Chaque cas d'utilisation contient un ou plusieurs scénarios qui définissent comment le système devrait interagir avec les utilisateurs (appelés acteurs) pour atteindre un but ou une fonction spécifique d'un travail. Un acteur d'un cas d'utilisation peut être un humain ou un autre système externe à celui que l'on tente de définir." - http://fr.wikipedia.org/wiki/Cas_d%27utilisation
On y comprend que l'approche par cas d'utilisation est tout de même tout à fait possible. Ceci étant dit, les mœurs changent toujours.

3.4.3 Le choix d'une technologie d'entreprise est difficile

3.4.3.1 La pression des fournisseurs

Les fournisseurs, voir vendeurs, ont tendance à éblouir les décideurs dans le but évidemment de gagner les ventes. Les bons côtés sont typiquement exagérés et les mauvais presqu'éliminés du discours de vente.

3.4.3.2 Mauvaise compréhension des décideurs

Les décideurs ne sont pas toujours, voir rarement, des informaticiens au fait des derniers avancements du domaine, mais plutôt des gestionnaires de haut niveau. Ajouter à cela que l'AOS et les technologies qui y sont connexes sont plutôt pointues.

3.4.3.3 Peur de la direction à se commettre

La peur face à une certaine technologie qui peut s'avérer complexe et, parfois, obscure pour la direction, peut faire en sorte qu'elle ne se commet pas ou seulement en partie [PATR2005]. Le manque d'appui de la direction peut nuire à l'implantation d'une architecture orientée-service.

3.4.3.4 Les coûts jouent pour beaucoup

Les difficultés au niveau des pourvoyeurs financiers (les payeurs) à s'ajuster au coût initial de l'AOS est une difficulté souvent rencontrée. Les notions AOS sont plutôt d'ordre techniques que fonctionnelles ce qui tend à rendre obscur les besoins réels de telles dépenses aux yeux des responsables financiers.

3.4.3.5 La formation

La formation et l'expérience des analystes et concepteurs peuvent être inadaptées. Afin qu'ils s'améliorent dans un contexte AOS, il peut s'avérer difficile de trouver de bonnes formations efficaces pour l'entreprise.

3.4.4 La modélisation

Les difficultés reliées à la modélisation des services et échanges de données entre des processus distants sont dues au fait d'un non consensus (pas de standards) de la part de la communauté informatique. [BOOC1999], [ROSS2006], [MARC2003]

C'est-à-dire que la modélisation des services étant purement au niveau *back-end* n'a pas de méthode de facto déjà apprivoisée par les informaticiens d'aujourd'hui. Il faudra encore un peu de temps pour que la communauté informatique fasse un choix quand à la méthode de modélisation d'une AOS.

3.5 Les perspectives qui se dégagent actuellement

3.5.1 Le vent dans les voiles des services Web et du Web 2.0

Les services Web ont pris beaucoup d'ampleur depuis 2003-2004 environ et c'est en cours d'amélioration. Des plates-formes technologiques propriétaires, telles que Siebel, ont ajouté la notion de services web à leur environnement après avoir saisi l'importance de l'interopérabilité entre les systèmes hétérogènes d'une entreprise. Principalement, il s'agit d'une pression commerciale qui s'exerce sur tous les acteurs

de l'industrie dans le but de mousser l'aspect commercial bien sûr, mais aussi dans le but d'améliorer la qualité des systèmes de façon globale. Le Web 2.0 est donc une raison que tous se donnent pour suivre une même direction à travers tous les choix qui pourraient s'offrir afin de standardiser une solution pour un certain type de problème (ex: les services web pour l'AOS) [HINC2006], [BOOT2004].

3.5.2 L'arrimage des méthodologies

Au niveau des méthodes de développement il y a une forte tendance à l'arrimage avec l'AOS vu son importance dans le domaine des technologies de l'information [AMIR2004], [MARC2003].

Ainsi, il ne faut pas être surpris si les méthodes Agile, RUP, P+, IBM Method, etc., de ce monde offrent maintenant des saveurs AOS de leurs méthodes. C'est bon signe!

3.5.3 L'arrimage de la modélisation

Au niveau de la modélisation, les outils UML tendent, eux aussi, à s'arrimer avec l'AOS (ex: Rational Modeler) afin de permettre de formaliser et de préciser les livrables d'analyse. [BOOC1999]

3.5.4 Les offres de formation sont compétitives

Plusieurs nouvelles conférences importantes sur l'AOS ont fait surface ces dernières années afin de mieux répondre aux besoins de formation des analystes et concepteurs. D'ailleurs les grandes conférences de l'industrie (ex: Java One) se sont parfaitement adaptées à ce nouveau marché et contribuent, jusqu'à une certaine mesure à l'avancement de la formation dans le domaine de l'AOS.

3.5.5 Émergence technologique

Plusieurs nouvelles technologies AOS ont émergé depuis 5 ans et une certaine épuration est en cours. [NORT2007], [WHAL2006], [PUTT2000], [WHAL2003], [GAVI2003], [WEBM2006]

C'est signe que les technologies se stabilisent et que les choix sont plus sûrs.

3.5.6 La difficulté de cerner l'AOS perdure

L'utilisation à outrance de termes de l'AOS dans le but de mousser les ventes de tel ou tel produit est encore très répandue et nuit à la compréhension générale de l'AOS [WEBM2006].

C'est d'ailleurs une forte tendance dans le domaine de l'informatique que d'utiliser à outrance un terme en vogue, de façon à mousser la commercialisation de tel ou tel autres produits et/ou solutions moins connus du grand public. Cette tendance est aussi remarquable avec l'AOS (SOA en anglais) ce qui nuit aux décideurs pour cerner ce qu'est en fait l'AOS au juste.

CHAPITRE IV

L'AOS EN CORRÉLATION AVEC LE SWEBOK[2]

4.1 Comment est organisé le chapitre 4

Tout d'abord, une problématique est établie appuyée par quelques exemples et références. En deuxième lieu la justification du choix du document de base (SWEBOK) est donnée, suivie par l'établissement des objectifs. Ces objectifs sont donnés sous une forme générique pour être référencés à chaque contribution au SWEBOK. Ces contributions découlent de la juxtaposition de SWEBOK et de l'AOS, en se basant sur les références, le discours des chapitres précédents, ainsi que sur l'expérience de l'auteur.

Les objectifs sont décrits sommairement et, lorsque repris en référence dans une sous-section de contribution (section 5 de ce présent chapitre), la façon dont un objectif en question est rencontrée est expliquée plus en détail dans son contexte.

Finalement les sous-sections de contribution correspondent à des sections définies dans le SWEBOK lui-même. Il est d'ailleurs implicite que la section correspondante est la référence principale du contenu. Une synthèse des liens entre objectifs et contribution est aussi dressée.

---

[2] L'AOS, dans le cadre du chaptire 5, peut aussi être vu comme étant un paradigme de développement, l'orientée-service.

4.2 La problématique abordée par ce mémoire

L'introduction, l'historique, l'aspect théorique et d'autres facettes liées à l'architecture orientée-service sont abordées dans ce mémoire. Par contre, voici que ce chapitre 5 couvre la partie qui peut être qualifiée d'apport au domaine. Or, avant d'attaquer le contenu de cet apport, il est d'usage d'exprimer la problématique à laquelle fait face le domaine étudié. Voici des exemples de problèmes illustrant le besoin de références formelles afin de guider les projets d'implémentation dans l'atteinte de leurs objectifs de succès.

Un premier problème actuel relié à l'implantation d'une AOS est qu'un investissement insuffisant au niveau analyse et conception ne permettra pas la conception de services ayant un bon niveau d'atomicité fonctionnelle. Les possibilités de réutilisation s'en trouveront directement réduites et donc les services ainsi construit seront moins rentables, de façon générale. Voici une citation d'une référence [BECK2007] illustrant ce problème : "*... you need to design business services at the right atomic level...an upfront investment in design is crucial...*".

Un deuxième problème actuel relié à l'implantation d'une AOS est l'approche itérative qui est nécessaire afin de produire des résultats satisfaisants. Si la méthode en cascade est utilisée pour des projets très importants il pourrait en résulter une dichotomie entre les besoins de l'entreprise au moment du déploiement et les besoins de l'entreprise au moment de l'analyse. Voici une citation à ce sujet [GRUM2007]: "*The trick is to not deploy SOA all at once, because by the time most "big-bang" efforts are completed, the business needs may have changed and much of the implementation will be outdated...*". Ce problème est lié aux phases d'analyse, de conception et de construction du cycle de vie du logiciel.

Un troisième problème est relié à la capacité de gouvernance d'une entreprise suite à la mise en place d'une architecture orientée-service. En d'autres termes, arrivée à la phase de maintenance, l'entreprise sera-t-elle capable de justement maintenir son architecture? L'entreprise est-elle capable de prévoir et planifier la phase de maintenance de son AOS? [NADH2004].

Bien d'autres problèmes sont relevés par l'industrie des TI. Ils sont souvent abordés positivement dans la littérature et on parle souvent en termes de défis plutôt qu'en termes de problèmes, bien que certaines références utilisent le terme problèmes comme tel. Finalement, le manque de formalisme nuit au domaine. [SCOT2004], [JACKS2004], [SEEL2006], [ARSA2007], [LYNC2006]

La problématique proposée est donc qu'il y a un manque de références formelles spécialisées sur le sujet sur lesquelles la communauté peut s'appuyer dans l'objectif de faire avancer les pistes d'améliorations et de solutions face aux problèmes liés à l'AOS, dont certains sont identifiés dans la présente section et en partie au début de ce document.

Tel qu'exprimé au début de ce mémoire, l'auteur n'a pas la prétention de créer un guide sur l'architecture orientée-service basé sur le SWEBOK, mais bien de jeter les bases d'un éventuel guide sur l'architecture orientée-service. Tous peuvent convenir que le guide SWEBOK est un ouvrage de référence formel. Or, ce guide se veut généraliste et n'implique pas directement telle ou telle tangente de l'informatique. Il s'agit d'un guide sur le génie logiciel de façon générale. L'architecture orientée-service n'y figure pas et n'y est pas mentionnée non plus. Il existe beaucoup de documents de littérature sur le sujet, mais les références formelles proviennent surtout

des différentes implémentations et sont donc liées au domaine de solutions techniques et technologiques plutôt qu'au domaine de solutions d'architectures logicielles.

Pour créer un ouvrage formel qui pourra être utile à l'ensemble de la communauté informatique, il ne s'agit pas de créer un mémoire de maîtrise, vous en conviendrez. Nous n'avons qu'à considérer l'ampleur du travail et les implications qui ont été investis dans un ouvrage comme le SWEBOK par exemple. Or, pour répondre à cette problématique, cet ouvrage propose le premier jalon soit la création des bases d'un éventuel guide sur l'architecture orientée-service.

4.3 La justification d'un guide sur l'AOS

4.3.1 Pourquoi le SWEBOK?

Le guide du SWEBOK [ABRA2004] est utilisé, par cet ouvrage, comme un gabarit afin de permettre certaines altérations, des précisions ou simplement des commentaires contextuels aux sections initiales dans le but de le préciser selon la saveur de l'AOS. En ce sens, le guide sur l'AOS hérite du guide sur le SWEBOK et y prend d'ailleurs la plus grande partie de son formalisme. Le contenu non-hérité du SWEBOK, donc ajouté dans ce présent document, est non validé par, et non associé au SWEBOK comme tel.

Le SWEBOK [ABRA2004] est la référence principale parce que c'est le guide sur le génie logiciel le plus connu, parce qu'il est facile d'utilisation, parce qu'il est formel et parce qu'il est le fruit d'innombrables efforts d'édition et de révision de gens de partout dans le monde, incluant de grandes entreprises. Il est à noter que d'autres références auraient pu servir comme gabarit principal de ce mémoire.

4.3.2 Justification générale

Un guide sur l'AOS permettrait d'uniformiser et de concentrer les efforts d'amélioration du domaine dans un document dans lequel la communauté informatique aurait confiance. Bien sûr le guide proposé dans ce présent travail n'est qu'une esquisse de ce que serait un véritable guide sur l'AOS, mais il s'agit tout de même d'un premier effort en ce sens. Trop de documents portant sur l'AOS sont axés sur un/des produit(s) offert(s) par un fournisseur, et non sur l'AOS directement et précisément. Or un guide sur l'AOS n'aurait pas de but commercial, mais bien un but scientifique dans le domaine du génie logiciel.

4.4 Objectifs d'un guide sur l'AOS

Les objectifs suivants sont ceux proposés par l'auteur. Il s'agit en fait d'une esquisse de ce qui serait proposé, à titre d'objectifs d'un guide sur l'AOS, par une équipe de chercheurs. Dans un tel cadre, les objectifs proposés découleraient d'une évaluation et d'une analyse systématique des pratiques dans ce domaine, ce que l'auteur n'avait pas la possibilité ni les moyens de faire. En lieu et place, la démarche choisie a été d'identifier un ensemble important de contributions préliminaires réalisées par l'auteur, puis de tenter de les regrouper par type d'objectifs auxquels ils se rattacheraient le mieux. Toujours dans cette démarche, l'intention est de définir une liste de types d'objectifs auxquels pourraient être rattachés tous les objectifs spécifiques associés à chacune des contributions. C'est donc ainsi que la liste suivante des objectifs a été produite :

[SUPPORTER]

- Supporter les analystes, architectes, testeurs, chefs de projet, et développeurs à chaque étape du développement.

[SPECIFIER]

- Spécifier des informations propres au SWEBOK en relation avec l'AOS.

[MODELISER]

- Donner des indications plus précises quand à la réalisation des travaux de modélisation d'une AOS.

[EXCEPTION]

- Améliorer la dimension de gestion des exceptions et des erreurs dans le cadre de l'AOS.

[COMPRENDRE]

- Améliorer la compréhension des différences entre les étapes de génie logiciel dans le cadre de l'AOS.

[CONCEVOIR]

- Supporter la conception de services d'entreprise.

[ENCADRER]

- Fournir un cadre pour mettre sur pied des architectures orientée-service basé sur les règles de l'art.

[GÉRER]

- Améliorer l'aspect gestion dans un contexte d'architecture orientée-service.

[ANALYSER]

- Améliorer les méthodes d'analyse dans un contexte d'architecture orientée-service.

4.5 Contributions associés au SWEBOK

Les contributions proposées à cette section sont associées à des étapes de développement du logiciel identifiées par le SWEBOK. Les étapes traitées et pour lesquelles des contributions sont proposées sont:

- Exigences du Logiciel Orienté Service
- Conception du Logiciel Orienté Service
- Construction du Logiciel Orienté Service
- Essai du Logiciel Orienté Service
- Maintenance du Logiciel Orienté Service
- La Gestion de Projet
- Gestion de la Configuration du Logiciel Orienté Service

Pour chacune de ces étapes de développement du logiciel, le découpage des thèmes du SWEBOK est donnée sous la forme d'un diagramme, suivi des contributions proprement dites.

4.5.1 Exigences du Logiciel Orienté Service

Voici tout d'abord le découpage des thèmes pour le domaine des connaissances des exigences du logiciel, tel que défini par le SWEBOK, auquel le lecteur peut faire référence.

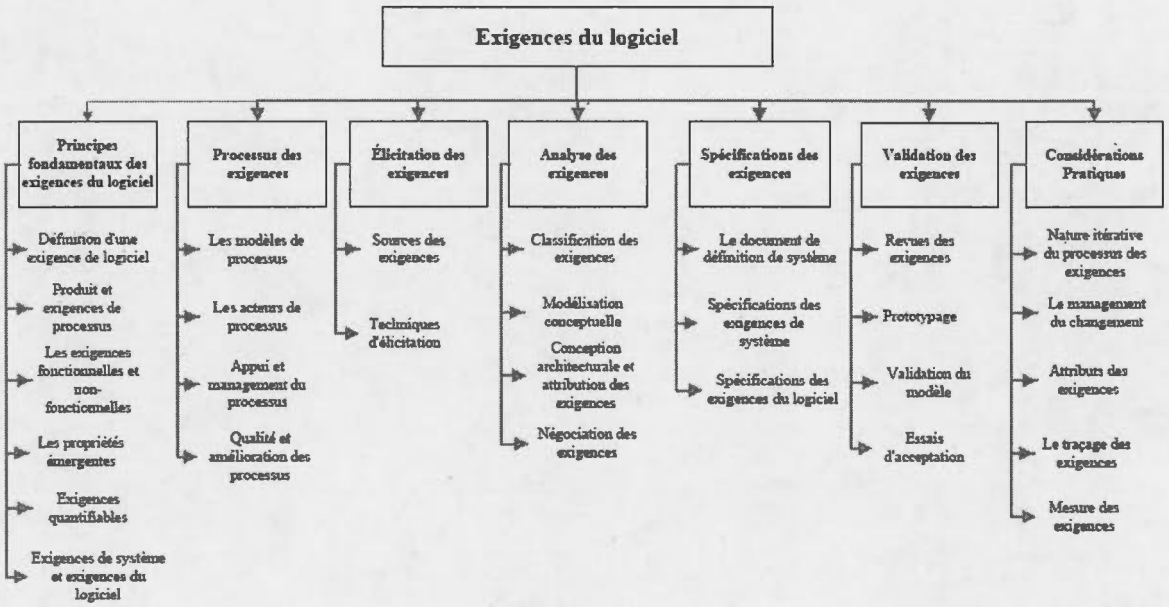


Figure 19 - le découpage des thèmes pour le domaine des connaissances des exigences du logiciel

Même si l'AOS tient plus du domaine de la conception que du domaine de l'analyse, certaines difficultés particulières surviennent dans des projets d'AOS au niveau de l'élaboration des exigences logicielles précisément.

A – Définir un intégrateur

Premièrement, l'élicitation des exigences, tel que défini dans le guide SWEBOK, est contraint, dans un contexte d'AOS, au fait que plus d'un participant (sous-système) doit évaluer des exigences logicielles. Or il faut définir un intégrateur qui est propriétaire du processus d'élicitation des exigences système et logicielle, tel que défini dans le guide SWEBOK.

Pourquoi?

- Éviter le dédoublement des exigences connexes et/ou similaires permet d'éliminer les anomalies dans les exigences.
- Intégrer les exigences connexes et/ou similaires dans les livrables d'exigences logicielles de façon à ce que la conception AOS soit appropriée.
- Augmenter les chances que toutes les exigences importantes soient effectivement élicitées.
- Améliorer la qualité des exigences de façon générale.

Complément d'information:

Les réunions facilitées se prêtent très bien à ces situations (voir guide SWEBOK), où les spécialistes des domaines pour lesquels on doit *intégrer* les exigences, sont présents.

Contribution aux objectifs:

[GERER]: Améliorer la gestion de l'étape d'élicitation des exigences logicielles.

B- Les modèles de flux de données

Au niveau de la modélisation conceptuelle, telle que défini dans le SWEBOK, il faut modéliser les échanges de données (messages) au niveau conceptuel, comme on le fait, par exemple, pour les modèles d'objets. Pour ce faire, les modèles de flux de données sont tout à fait appropriés. De plus, certaines structures de données au niveau des formats d'échanges peuvent être problématique. Pour ce faire, il faut s'adresser à la littérature technologique appropriée (selon le cas).

Pourquoi?

"...Quelques types de logiciel demandent que certains aspects soient analysés particulièrement rigoureusement." – SWEBOK

Évidemment, dans le cadre des AOS, l'aspect des flux de données est très important au niveau des échanges de messages entre les systèmes hétérogènes et/ou distants. Ici, on n'entend pas l'analyse des flux de données entre le client (interface personne machine) et le serveur d'application par exemple. Plutôt, on entend modéliser les échanges entre les processus. C'est-à-dire tous les aspects à saveur AOS, où donc les échanges de messages sont inévitables sous toutes sortes de forme (ex: requête/réponse, *send-and-forget*, etc.).

Contribution aux objectifs:

[MODELISER]: Aider dans les choix de type de modèle à l'étape d'analyse.
[SPECIFIER]: Spécifier dans quel contexte type un diagramme peut être utile dans une AOS.

C- Attribuer des exigences à des composantes orientées-service

"... La conception architecturale est un point avec lequel le processus des exigences chevauche celui de la conception..." – SWEBOK
Cette réalité amenée par le SWEBOK identifie donc, dans un contexte d'AOS, le besoin d'identifier les exigences logicielles qui correspondront à des composantes de conception (ex: service) déjà à la phase d'analyse.

Pourquoi?

Afin de bien comprendre quoi concevoir au niveau de la conception AOS. Cette étape aidera grandement l'ingénieur logiciel. Pour un ensemble d'exigences logicielles données, seules certaines parties ont des liens avec la nature AOS du système visé. Or l'identification du ou des sous-ensembles des exigences logicielles ayant un lien clair avec l'AOS, aidera à spécifier la totalité du "quoi" impliquée dans la conception AOS.

Contribution aux objectifs:

[ANALYSER] : Permettre dès le départ d'associer certaines exigences à certains composants logiciels à concevoir par la suite. Amélioration du lien entre l'analyse et la conception dans un cadre d'une AOS.

D- Pas de BPEL à l'analyse

Au moment de la "*Validation du Modèle*" (voir le SWEBOK), il est important de vérifier que le modèle de flux de données n'est pas en fait un ensemble de scénarios d'exécution tel que spécifié par un modèle BPEL (*Business Process Execution Language*).

Pourquoi?

Même s'il s'agit d'un effort de standardisation, le BPEL est plutôt à cheval entre la phase de conception et de construction. Il s'agit d'une méthode trop près de la réalisation pour être utile afin de spécifier le ou les modèle(s) de flux de données appliqué(s) aux exigences logicielles à saveur d'AOS.

Contribution aux objectifs:

[COMPRENDRE] : Améliorer la compréhension de l'utilisation du BPEL dans le cadre des étapes du génie logiciel.

E- Corrélation entre gestion du changement et le principe d'intégrateur

Il faut maintenir une corrélation entre le management du changement (tel que défini dans le SWEBOK) et le principe d'intégrateur des exigences logicielles (tel que défini dans cette présente section).

Pourquoi?

Le but est de maintenir le principe de propriété des exigences logicielles qui sont partagés entre deux ou plusieurs systèmes intégrés par une technique d'AOS, et ce à travers le cycle de vie du système global. Finalement, il est donc suggéré que l'intégrateur soit ou nomme un gestionnaire du changement.

Contribution aux objectifs:

[SUPPORTER] : Supporter les chefs de projet dans la gestion du changement à la phase d'élicitation des exigences.

[SPECIFIER] : Spécifier comment, dans le cadre d'une AOS, il peut s'avérer important de bien gérer le changement en responsabilisant l'intégrateur.

[GERER] : Aider la gestion du changement dans un contexte d'AOS.

F- Ne pas dédoubler les exigences à cause de leur saveur AOS

Tel que défini dans le SWEBOK, il est en général utile d'avoir un certain concept de "volume" des exigences. Or dans un contexte d'AOS, il peut être beaucoup plus facile de multiplier une ou des exigences à saveur d'AOS à travers les différents sous-systèmes impliqués.

Pourquoi?

Parce que telle ou telle exigence logicielle peut être reprise plus d'une fois dans de multiples contextes dû à la nature *distribuée* et la notion de producteur/consommateur d'une architecture orientée-service ce qui est, a priori correct. C'est pourquoi il ne faut pas dédoubler le volume des exigences à saveurs AOS. Il est donc important d'être plus vigilant à ce niveau comparativement à un contexte non AOS.

Contribution aux objectifs:

[ANALYSER] : Aider à être productif lors de l'élicitation des exigences dans le cadre d'une AOS.

[ENCADRER] : Aider à encadrer l'analyste dans une AOS.

4.5.2 Conception du Logiciel Orienté Service

Voici tout d'abord le découpage des thèmes pour le domaine des connaissances de la conception du logiciel, tel que défini par le SWEBOK, auquel le lecteur peut faire référence.

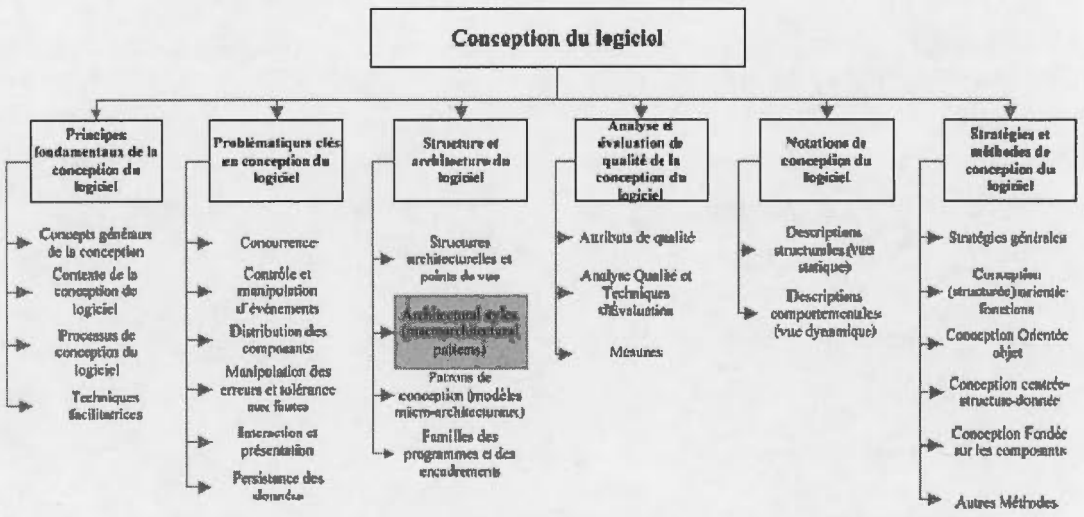


Figure 20 - le découpage des thèmes pour le domaine des connaissances de la conception du logiciel

C'est à la phase de conception que l'AOS prend le plus de place dans le cycle de vie du logiciel. C'est bien pendant cette phase que l'informaticien (disons l'architecte, ou encore l'ingénieur logiciel) décide de prendre une tendance, ou une saveur d'architecture orientée service. Dans les termes du SWEBOK, ont parle de deux activités de conception importantes soit la *conception architecturale du logiciel* et, dans un deuxième temps, la *conception détaillée du logiciel.*

A- Déterminer les services candidats à partir des exigences

Il faut arriver à *élire*, à partir des exigences logicielles, les services candidats afin de déterminer quel sont les services qui feront partie de l'orientation service d'une architecture logicielle donnée. Il est à noter qu'une architecture logicielle peut comporter une certaine *quantité* de fonctionnalités qui sont orientées-service, en plus d'une certaine quantité de fonctionnalités qui ne le sont pas. Afin de déterminer les services candidats, on peut faire référence aux techniques facilitatrices (*enabling techniques*) tel que défini dans le SWEBOK.

Contribution aux objectifs:

[COMPRENDRE] : Aider à la compréhension du rôle de la conception par rapport à l'analyse.

[CONCEVOIR] : Aider à la conception de services, dans le cadre d'une AOS, en fonction des extrants de l'analyse.

[SPECIFIER] : Donner un exemple au *enabling techniques* du SWEBOK, dans le cadre d'une AOS.

B- Décomposition et modularisation

"... a habituellement pour but de placer différentes fonctionnalités ou responsabilités dans différents composants." – SWEBOK

Il faut alors établir quelles fonctionnalités définies par les exigences logicielles ont assez de cohésion afin de les *placer* dans différents services (au sens ou le SWEBOK l'entend).

On peut donc procéder à une association de différentes exigences logicielles que l'on sait devoir être traduites en fonctionnalités/responsabilités afin d'établir des sous-

ensemble candidats (là où l'échange d'information sera nécessaire) qui pourront devenir un/des service(s).

Contribution aux objectifs:

[SUPPORTER] : Supporter le concepteur dans le cadre d'une AOS afin de l'aider à décomposer et modulariser en fonction des exigences.

[CONCEVOIR] : Aider à la conception de service.

C- Le couplage et la cohésion

"... couplage est défini comme la force des relations entre les modules ... cohésion est définie par la façon dont les éléments...sont reliés." – SWEBOK
Le couplage entre des modules (i.e.: sous-système) devant inévitablement échanger des données, selon les exigences logicielles, devra être traduit par un/des service(s) où la force du lien qui unit consommateur et producteur est au plus bas possible, sans affecter une bonne cohésion du service lui-même. C'est-à-dire que le service, émergeant du besoin d'échange entre deux modules (i.e.: sous-systèmes), doit trouver un équilibre entre la tendance à diminuer le couplage et la tendance à augmenter la cohésion au même titre qu'une procédure dans un paradigme procédural doit le faire.

Pourquoi?
Parce que les services candidats doivent simplement être amenés par les besoins du logiciel visé, c'est-à-dire par les exigences logicielles.

Contribution aux objectifs:

[CONCEVOIR] : Aider le concepteur à tenir compte, et à se servir, du couplage et de la cohésion lors de la conception dans le cadre d'une AOS, et lors de la désignation des services.

D- Abstraction et encapsulation

Tel que défini dans le SWEBOK, la paramétrisation et la spécification sont les outils d'abstraction dans le domaine du génie logiciel. Il faut donc contrôler ces deux aspects afin de maximiser le niveau d'abstraction des services. L'encapsulation des services sert à grouper et assembler les détails internes au service de l'abstraction.

Pourquoi?

Au niveau de l'architecture orientée-service, il est primordial que les services soient le plus abstraits possible, grâce à une bonne encapsulation logicielle, afin de tendre vers la réutilisation, voir vers la centralisation des services à développer et à maintenir.

Contribution aux objectifs:

[CONCEVOIR] : Aider le concepteur à tenir compte de l'abstraction et de l'encapsulation lors de la conception dans le cadre d'une AOS.

E- Séparation d'interface et d'implantation

La séparation d'interface et son implantation dans l'architecture orientée-service permet l'abstraction et l'encapsulation des détails internes aux services. Sans être une nécessité du domaine de l'architecture orientée-service, la standardisation de la définition de l'interface est un atout de mise en œuvre extrêmement important car il facilite l'interopérabilité des environnements technologiques hétérogènes.

Pourquoi?

Afin de réduire le couplage entre le consommateur et le producteur par l'abstraction et l'encapsulation. De plus, cette technique tend aussi vers la réutilisation des services à développer et à maintenir.

Contribution aux objectifs:

[ENCADRER] : Faire en sorte que le concepteur tienne compte d'une notion AOS fondamentale.

F- Suffisance, complétude et primarité

Ces notions permettent la cohérence par la simplicité de mise en œuvre en capturant toutes les caractéristiques importantes d'une abstraction et rien d'autres.

Pourquoi?

- Tel qu'on le dit si bien en langue anglaise "*Keep it simple*" est un dicton ayant porté fruit à maintes reprises dans le domaine du génie logiciel. De plus, la complétude et la suffisance dénote une forte cohérence.
- Dans le domaine de l'architecture orientée-service, ces notions prennent toutes leur importance au niveau de la conception des services dans l'optique de leur utilisation éventuelle à titre de *service centraux*, avec toutes les spécificités que cela comprend, c'est-à-dire les notions de réutilisation, de concurrence, de performance et de robustesse, pour ne nommer que celles-ci. Donc la suffisance, la complétude et la primarité permet aux services d'une architecture de mieux répondre aux besoins de leurs consommateurs, du point de vue de leurs exigences fonctionnelles et non-fonctionnelles communes.

Contribution aux objectifs:

[ENCADRER] : Faire en sorte que le concepteur tienne compte d'une notion AOS fondamentale.

G- Gestion de la concurrence d'accès

La notion de concurrence prend toute son ampleur dans le domaine de l'architecture orientée-service, voir plus que le domaine de client/serveur conventionnel. L'architecture orientée-service hérite du modèle client/serveur en se concentrant sur l'architecture de bas niveau, sur les composants logiciels ayant trait au domaine à informatiser. Dans un modèle client/serveur la concurrence est un facteur non négligeable.

Pourquoi?

- Dans une architecture orientée-service, les exigences de concurrence doivent être rencontrées pour chaque service mis en place, en gardant en tête que le nombre de consommateurs n'est pas déterminé au moment de la mise en œuvre, puisque les services doivent être conçus afin de répondre à un nombre évolutif de requêtes concurrentes du point de vue logiciel. Du point de vue infrastructure, le facteur concurrence doit être perçu de la même manière que dans une architecture client/serveur.
- Il faut comprendre que la notion de concurrence d'accès est un aspect de conception à ne pas omettre lors de la conception d'un service ou d'une AOS. La concurrence n'est donc pas exprimée ici comme un service en soi, mais

bien une exigence non fonctionnelle fondamentale et réelle à un service ou à une AOS.

Contribution aux objectifs:

[CONCEVOIR] : Supporter le concepteur afin qu'il voit certains problèmes typiques dans le cadre d'une AOS.

H- Distribution de composants (services)

L'intergiciel, qu'il soit développé, ou simplement réutilisé, est étroitement lié à la distribution des composants (services), puisqu'il aura comme principal but d'exécuter le *logiciel hétérogène*. Par *logiciel hétérogène*, il est ici sous-entendu "l'ensemble des composants orientés-service, l'ensemble des consommateurs, l'ensemble des producteurs, ainsi que tout autres acteurs logiciels et/ou humains faisant partie de la solution finale". Il est convenu qu'une intervention humaine peut faire partie d'une solution orientée-service selon les exigences identifiées au préalable.

Pourquoi?

- Le *middleware* (intergiciel) est étroitement lié à la distribution des composants puisqu'il guide généralement cette distribution par sa nature technologique, ou encore par sa conception même. Par exemple, un intergiciel Java EE utilisé afin de déployer et d'exécuter des services web agit directement sur la manière dont les services seront distribués. Il est alors convenu, dans ce cas précis, que c'est l'architecture globale (incluant la détermination du lieu d'exécution de l'intergiciel) qui agit directement sur la distribution des services.

- Dans d'autres cas, cette réalité sera moins frappante grâce à la nature plus souple de l'intergiciel utilisé. Par exemple, la simple utilisation d'un outil de files d'attentes (*Message Queuing*) n'a pas beaucoup d'effet sur la distribution des composants (services) puisque ceux-ci peuvent accéder à distance à l'intergiciel et que celui-ci ne contient pas les services qui l'utilisent.

Contribution aux objectifs:

[SPECIFIER] : Aider à la compréhension de la distribution des composants, tel qu'expliqué dans le SWEBOK, dans le cadre d'une AOS.

I- Manipulation des erreurs et tolérance aux pannes

En plus, d'appliquer les règles conventionnelles de manipulation des erreurs et de tolérance aux pannes, les intergiciels orientés-service ont avantage à fournir un soutien supplémentaire dans ce domaine. De tels intergiciels peuvent fournir un support au principe de compensation. C'est-à-dire une fonction dédiée à la gestion des erreurs lorsqu'elles font surface pendant le cours normal d'exécution.

Pourquoi?

- Il est avantageux lors du développement et plus facile à maintenir lorsqu'un intergiciel offre le support à la gestion des exceptions par le principe de code de compensation. Normalement, c'est à l'équipe de développement de concevoir et de créer les codes de compensation qui seront exécutés lorsque les exceptions surviendront.
- Ainsi, la standardisation de la gestion des exceptions peut être atteinte dans le cadre d'une architecture orientée-service.

Contribution aux objectifs:

[EXCEPTION] : Améliorer la gestion des exceptions systèmes dans le cadre d'une AOS.

J- Style architectural réparti

Le style architectural réparti est celui qui devrait être préconisé dans le cadre d'une solution orientée-service.

Pourquoi?

- La notion client-serveur est reprise de façon intégrale entre le consommateur et le producteur dans le cadre orienté-service.
- L'approche trois-tiers est aussi parfaitement valide (voir n-tiers plutôt).
- De plus, l'approche courtier est valable lorsqu'un intergiciel entre en scène d'une architecture orientée-service.

Contribution aux objectifs:

[SPECIFIER] : Aider au choix du style architectural, tel qu'avancé dans le SWEBOK, dans le cadre d'une AOS.

K- Langage de description de l'interface standardisé (LDI)

Il est fortement préconisé d'utiliser un langage de description de l'interface standardisé (LDI). Certaines technologies orientées-service ont d'ailleurs adopté, voir inventé leur propre langage de description de l'interface. Par exemple, les services web ont défini le standard WSDL (Web Service Description Language), CORBA a défini son propre IDL (Interface Definition Language), etc.

Pourquoi?

- Permet une plus facile réutilisation et une meilleure séparation des responsabilités entre le producteur et le consommateur de service.
- D'autres avantages en découle (ex: diminution des coûts de développement).

Contribution aux objectifs:

[SPECIFIER] : Spécifier l'importance et le rôle du langage de description de l'interface standardisé dans le cadre d'une AOS.

L- Diagramme d'activités pour la description comportementale des services

Le diagramme d'activités se prête bien à la description comportementale des services et facilite la compréhension au niveau conceptuel. Il peut servir de livrable fonctionnel afin de spécifier plus précisément les modalités de tel ou tel service. Un fait important à noter ici est qu'un service peut nécessiter, afin de remplir son mandat, l'accès à plus d'une source, voir même à d'autres services.

Pourquoi?

Parce que les activités d'un tel diagramme expriment assez facilement le découpage interne d'un service de façon à mieux cerner ce qu'il doit et ne doit pas faire.

M- Diagramme de flux de données pour la description des échanges de données entre les sous-systèmes par l'entremise des services candidats

Le diagramme de flux de données se prête bien à la description comportementale des échanges de données entre les sous-systèmes par l'entremise des services candidats.

Plus souvent qu'autrement, les échanges de données entre les consommateurs et producteurs de services sont fondamentales quant à la conception d'une architecture orientée-service.

Pourquoi?

- Ce type de diagramme permet d'illustrer comment le ou les modèles de données d'une architecture orientée-service sont utilisés par les divers consommateurs et producteurs de services.

- En sachant comment ils sont utilisés, le ou les modèles de données peuvent même être adapté(s) selon le cas, au même titre que dans l'approche client-serveur, où les interfaces personne-machines peuvent guider la conception du ou des modèle(s) de données.

Contribution aux objectifs:

[MODELISER] : Donner des indications plus précises quand à la réalisation des travaux de modélisation d'une AOS.

N- Diagramme de collaboration UML, excellent pour décrire la conception interne d'un service d'entreprise

À l'inverse du diagramme de flux de données, le diagramme de collaboration UML est excellent pour décrire la conception interne d'un service d'entreprise.

Pourquoi?

Parce qu'il permet de découper en petite unité conceptuelle chacune des étapes internes propres à l'exécution du service d'entreprise.

Contribution aux objectifs:

[MODELISER] : Donner des indications plus précises quand à la réalisation des travaux de modélisation d'une AOS.

O- Diagramme de déploiement, très important pour déterminer l'infrastructure sous-jacente à un ou plusieurs service(s) d'entreprise

Au niveau de la conception physique, le diagramme de déploiement, utilisé pour représenter un ensemble de nœud et leurs interrelations, est très important pour déterminer l'infrastructure sous-jacente à un ou plusieurs service(s) d'entreprise.

Pourquoi?

Il permettra de définir où, dans l'infrastructure et le réseau d'une entreprise, un ou des service(s) seront déployés. Il s'agira d'un livrable très important pour l'exploitation du ou des services, entre autres, afin de mieux définir l'architecture des systèmes consommateurs du ou des services.

Contribution aux objectifs:

[ENCADRER] : Aider à encadrer la gestion des déploiements dans le cadre d'une AOS.

P- Les langages de description d'interface et des données sont fondamentaux

Peu importe par quel formalisme ce sera fait, le langage de description d'interface est fondamental dans le cadre de la conception d'un ou de service(s). Dans le même ordre d'idée, le langage de description des données l'est tout autant.

Pourquoi?

Parce qu'il permet de formaliser l'interface d'un service afin de guider la conception d'un consommateur de service, que ce soit un service aussi ou simplement un composant client. Dans le cas du langage de description des données, il sert à définir les structures de données d'une façon standardisé et portable entre les différents environnements technologiques hétérogènes. Par exemple le XML (XSD) peut être une solution parfaitement applicable.

Contribution aux objectifs:

[SPECIFIER] : Appuyer l'importance et le rôle du langage de description de l'interface standardisé dans le cadre d'une AOS.

4.5.3 Construction du Logiciel Orienté Service

Voici tout d'abord le découpage des thèmes pour le domaine des connaissances de la conception du logiciel, tel que défini par le SWEBOK, auquel le lecteur peut faire référence.

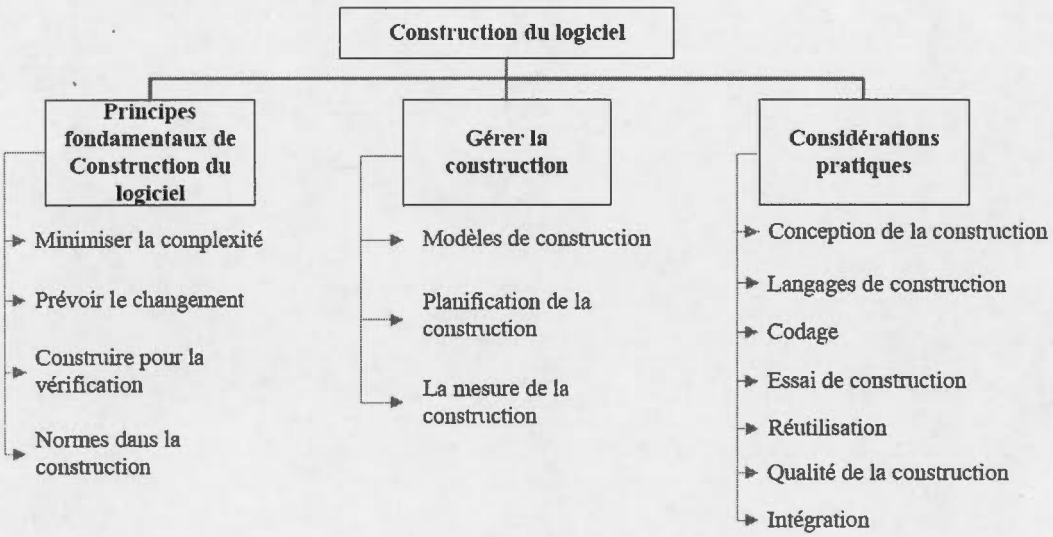


Figure 21 - le découpage des thèmes pour le domaine des connaissances de la construction du logiciel

A- Choix du langage en fonction de la plate-forme choisie

Le choix du langage de construction doit être motivé par la plate-forme technologique choisie. De toute évidence, ce dernier choix doit s'orienter vers les capacités de la plate-forme pour satisfaire les caractéristiques d'une architecture orientée-service.

Pourquoi?

- Parce que le langage lui-même est moins important que la plate-forme technologique sur laquelle il s'exécutera.
- Par plate-forme technologique, il est sous-entendu, un ensemble d'outils technologiques répondant aux caractéristiques d'une architecture orientée-service (ex: les services web dans le cadre de JEE).

Contribution aux objectifs:

[ENCADRER] : Aider l'architecte dans le cadre de décision importante dans la phase de construction, suite à la phase de conception.

B- Coffres à outils orientée-service

Les coffres à outils orientées-service sont souvent responsables de fournir le support à la mise en place d'une architecture orientée-service.

Pourquoi?

Parce que ce sont des ensembles intégrés de pièces réutilisables propres au domaine de l'architecture orientée-service, et fournissent donc une couche logicielle fondamentale à son bon fonctionnement.

Contribution aux objectifs:

[ENCADRER] : Aider l'architecte ou le programmeur à améliorer le processus de construction logicielle dans le cadre d'une AOS.

C- Notation visuelle et chorégraphie

La notation visuelle est fort utile pour représenter la chorégraphie d'un ensemble de services afin d'associer différents services pour en créer un nouveau.

Pourquoi?

Dans le cadre du BPEL (Business Process Execution Language), la notation visuelle est normalement utilisée pour définir les processus (chorégraphies).

Contribution aux objectifs:

[MODELISER] : Aider l'architecte ou le programmeur à améliorer son travail de construction d'une AOS.

D- Notation visuelle et configuration

La notation visuelle est fortement utile pour configurer les différents composants d'un service souvent exprimés sous forme XML.

Pourquoi?

Il s'agit d'une méthode qui permet aux développeurs de se retrouver facilement dans une multitude de métadonnées de configuration souvent persistée sous la forme de fichiers XML.

Contribution aux objectifs:

[SPECIFIER] : Spécifier comment les notations visuelles peuvent s'appliquer dans le cadre d'une AOS.

E- Gestion de compensation

Afin de supporter la gestion des exceptions et des erreurs, il faudra prévoir du code de gestion de compensation.

Pourquoi?

- Dans le cas où un service rencontre une exception, une erreur, il doit être capable de la gérer. Pour ce faire, il faut prévoir du code spécifique à ces éventualités.
- Dans un cadre transactionnel, il peut être primordial de créer du code de compensation qui permet d'effectuer un rollback complet des activités de la transaction en erreur.

Contribution aux objectifs:

[EXCEPTION] : Améliorer la dimension de gestion des exceptions et des erreurs dans le cadre de l'AOS.

F- Essais d'intégration

Les essais d'intégration sont importants et consistent à créer des cas de tests qui émulent un système consommateur éventuel.

Pourquoi?

Les essais d'intégration sont importants parce que les tests unitaires ne sont pas suffisants. Ils ne s'attardent pas à représenter le système client éventuel. Par exemple, la concurrence et la charge sont des dimensions qui échappent normalement aux tests unitaires.

Contribution aux objectifs:

[SUPPORTER] : Supporter le testeur dans le cadre de la construction d'une AOS.

G- Réutilisation et intégration sont la base

Évidemment la réutilisation, de même que l'intégration, sont à la base d'une architecture orientée-service.

Pourquoi?

- Parce qu'un service en soi ne peut être utile que dans le cadre d'un appel d'un consommateur.
- Or le service doit être réutilisable par différents consommateurs.
- De plus, l'architecture orientée service doit servir l'intégration entre les consommateurs et les producteurs.

Contribution aux objectifs:

[SPECIFIER] : Spécifier comment la réutilisation et l'intégration sont des notions importantes à l'AOS.

4.5.4 Essai du Logiciel Orienté Service

Voici tout d'abord le découpage des thèmes pour le domaine des connaissances des essais du logiciel, tel que défini par le SWEBOK, auquel le lecteur peut faire référence.

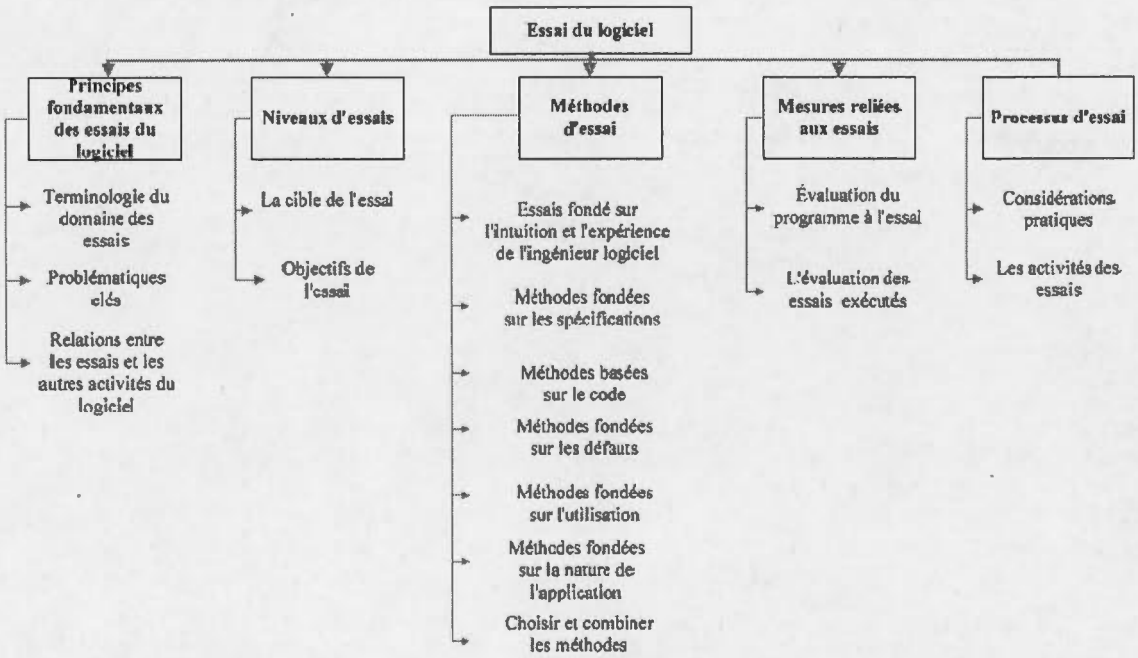


Figure 22 - le découpage des thèmes pour le domaine des connaissances des essais du logiciel

A- Les essais d'intégration sont nécessaires

Le niveau d'essai d'intégration est nécessaire dans le cadre d'une architecture orientée-service. Comme un jeux d'ensembles, lorsqu'arrive le temps de tester le super-ensemble, il s'agira de tests d'intégration.

Pourquoi?

Ne pas tester le super-ensemble, c'est-à-dire formé de *systèmes consommateurs* et de *services*, revient à déléguer à la maintenance le lot de travail associé à la stabilisation de la solution.

Contribution aux objectifs:

[SUPPORTER] : Supporter le testeur dans le cadre d'une AOS.

[GERER] : Aider au gestionnaire d'une solution d'AOS à gérer les essais intégrés.

B- Cible d'essai et objectifs de l'essai

Les objectifs de l'essai, dans le cadre d'une architecture orientée-service, sont fixés en fonction de la cible. L'architecture orientée-service implique qu'il y a un ensemble dit *services* et un ensemble dit *systèmes consommateurs*. Nous pourrions facilement, pour le bien de la discussion, établir qu'un autre ensemble pourrait être formé d'une ou plusieurs instances d'ensemble dit *systèmes consommateurs* et d'un certain ensemble dit *services.* Or la cible d'essai doit varier en fonction de cet ensemble, en premier lieu. C'est-à-dire que l'ensemble choisi déterminera les objectifs, et donc les types d'essais à effectuer.

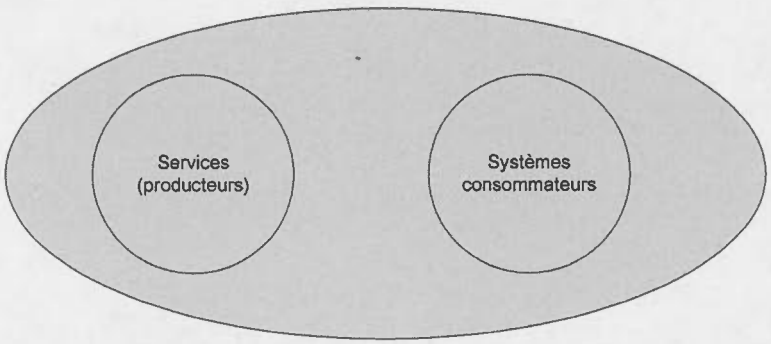


Figure 23 - Ensembles (cibles) en exemple pouvant être identifiés dans le cadre d'une AOS

Pourquoi?

Parce que certains niveaux d'essais s'appliquent mieux à certains ensembles (cibles) qu'à d'autres, dans le cadre de l'architecture orientée-service.

Par exemple, supposons que la cible d'essai vise l'ensemble dit *services*. Il est évident que des essais d'intégration ne sont pas appropriés puisque ce type d'essai serait beaucoup plus approprié pour une cible visant la conjonction entre un ensemble donné *services* et un ensemble donné *systèmes consommateurs*, afin, par déduction, de tester l'intégration entre ces ensembles (cibles). Plutôt, dans ce même exemple, un niveau d'essai de performance, de stress, ou encore de fiabilité serait beaucoup plus approprié.

Contribution aux objectifs:

[ENCADRER] : Fournir un cadre de compréhension au gestionnaire, face aux ensembles à tester dans le cadre d'une AOS.

C- Essais de performance, de fiabilité versus essais de concurrence

Bien que le SWEBOK n'aborde pas les essais de concurrence, il est essentiel, dans le cadre d'une architecture orientée-service de procéder à des tests de concurrence. Malheureusement, souvent on confond les essais de performance, ou encore de fiabilité, avec les essais de concurrence.

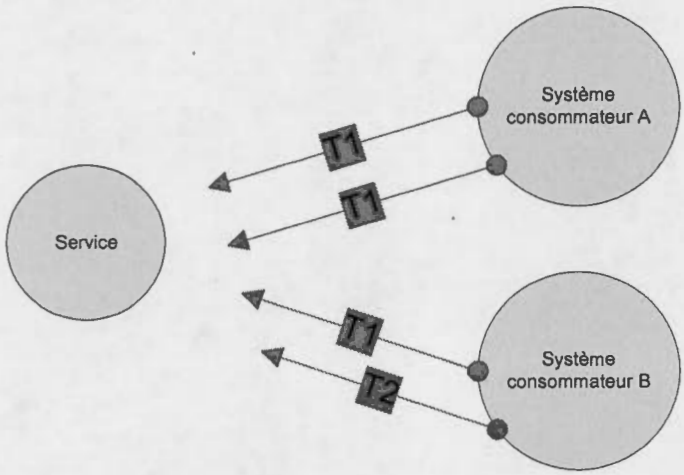


Figure 24 - La notion de concurrence d'accès dans le cadre d'une AOS

Pourquoi ?

Les essais de concurrence sont aussi importants que les essais de performance et de fiabilité, en ce sens que la problématique de concurrence d'accès est inhérente à tout système transactionnel. Or une composante dite *service* aura, du moins à terme, à faire face à plus d'un système consommateur. De plus, dans le cadre d'un seul système consommateur, la concurrence peut être requise. Or il devient évident que des tests de concurrence d'accès doivent avoir lieu avant d'établir une cible d'essai plus grande que l'ensemble dit *services*.

Contribution aux objectifs:

[SPECIFIER] : Spécifier certaines notions détaillées d'essais dans le cadre d'une AOS.

D- Réutilisation de cas d'essais et essais unitaires dans le cadre d'essais d'intégration

À la base, lorsqu'un service est modifié d'une quelconque façon, il faut procéder à des tests unitaires. Par contre, il faudra aussi procéder à un/des cas d'essais unitaires équivalent(s) (ayant trait aux mêmes fonctionnalités) au niveau du/des consommateur dans le cadre des cas d'essais d'intégration. Or il peut être intéressant de partager le ou les essais unitaires du service aux équipes d'essais d'intégration afin qu'il oriente leur cible vers la/les fonctionnalités visées.

Contribution aux objectifs:

[GERER] : Aider à gérer les essais dans le cadre d'une AOS.

4.5.5 Maintenance du Logiciel Orienté Service

Voici tout d'abord le découpage des thèmes pour le domaine des connaissances de la maintenance du logiciel, tel que défini par le SWEBOK, auquel le lecteur peut faire référence.

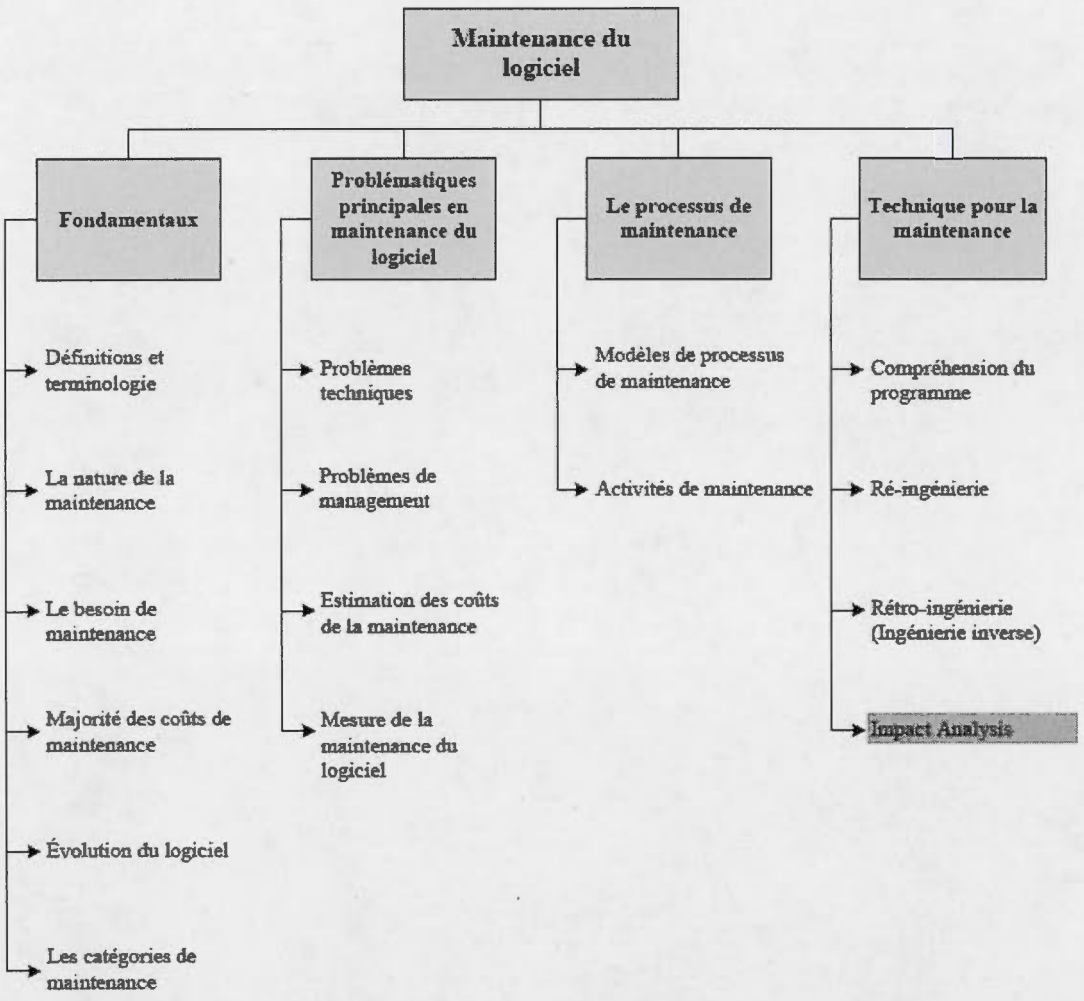


Figure 25 - le découpage des thèmes pour le domaine des connaissances de la maintenance

A- Importance de la planification de la maintenance

Autant la planification d'affaire, la planification des mises en production/version que la planification des demandes de changements sont d'une haute importance dans le cadre d'une architecture orientée-service puisque des *cellules* de travail indépendantes et inter-reliées travaillent en parallèle tout en ayant un couplage technologique les unes envers les autres.

En fait, il s'agit de s'assurer d'une forte cohésion entre les différents responsables de la gestion de la maintenance dans le but de maintenir l'intégrité d'un système basé sur l'AOS.

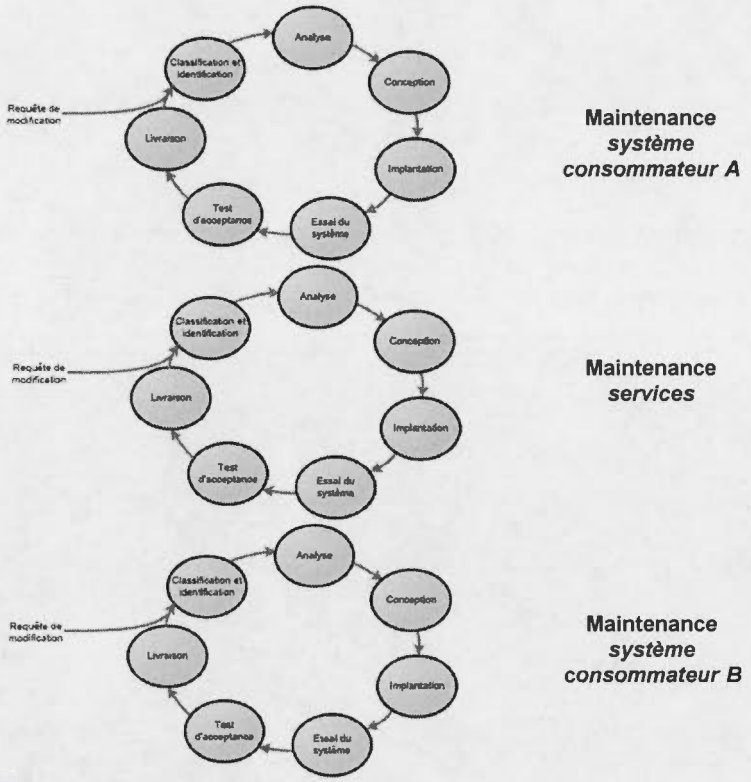


Figure 26 - Illustration de la dynamique des groupes de maintenance dans une AOS

Contribution aux objectifs:

[GERER] : Améliorer la gestion de la maintenance dans le cadre d'une AOS.

4.5.6 La Gestion de Projet

Voici tout d'abord le découpage des thèmes pour le domaine des connaissances de la gestion du génie logiciel, tel que défini par le SWEBOK, auquel le lecteur peut faire référence.

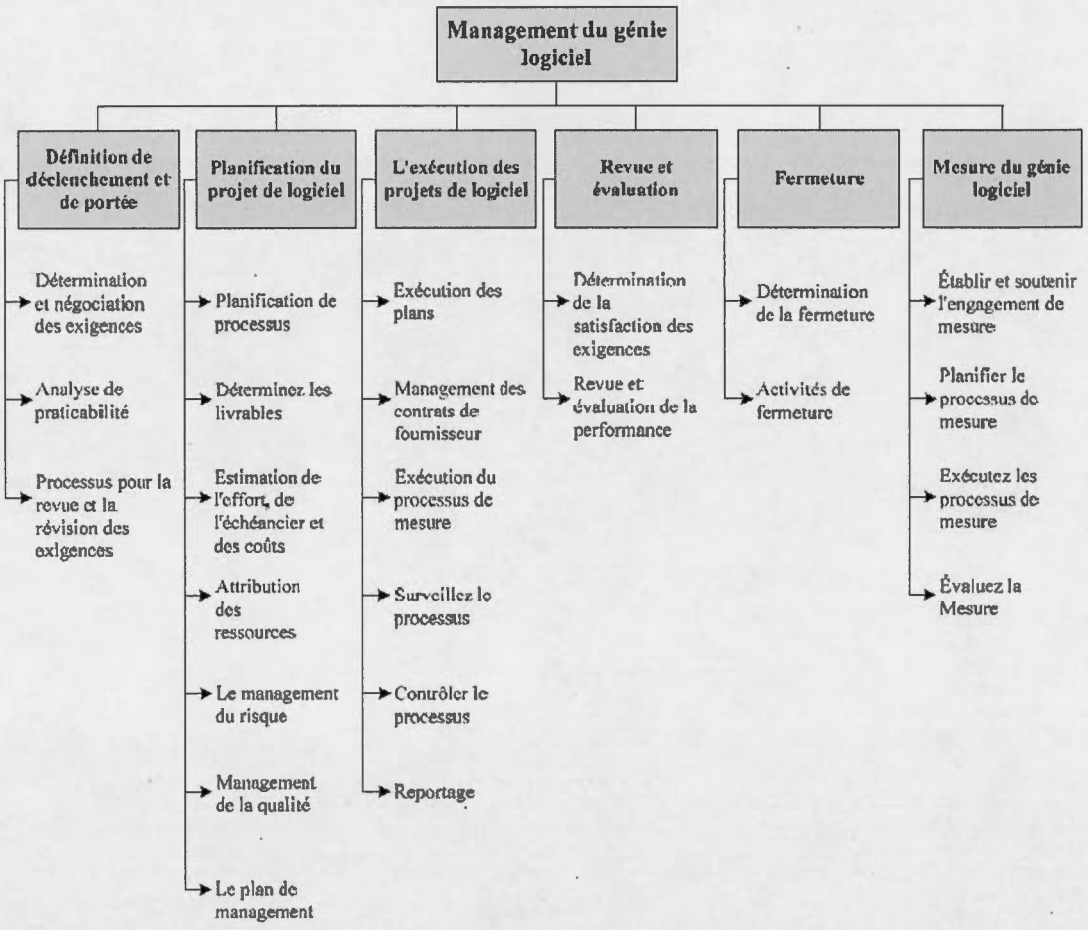


Figure 27 - le découpage des thèmes pour le domaine des connaissances de la gestion

A- Gestion des contrats des services

À la section 3.2 (voir le SWEBOK, chapitre sur la gestion de projet) dans cette section du SWEBOK, on discute brièvement de *management des contrats de fournisseurs*. Or, dans une architecture orientée-service, la notion de contrat prend une place fondamentale au niveau de la gestion en génie logiciel. Pour tout service, son interface constitue une grande partie de son *contrat*. Reste finalement l'implantation du service lui-même qui peut être influencé par le contrat. Typiquement, dans l'entreprise, un premier client payeur (système consommateur) permettra de déterminer un contrat d'entreprise ou inter-entreprise, dans le but de régir les spécificités d'un futur service. Ainsi, un prochain client (système consommateur) pourra prendre compte de ce même contrat afin de s'adapter lui-même, voir de faire modifier le dit contrat au besoin dans le cadre d'une entente entre tous les partenaires technologiques. Finalement, on peut voir que la notion de contrat avec les fournisseurs, s'étend entre les acteurs d'une architecture orientée-service.

Contribution aux objectifs:

[SPECIFIER] : Il s'agit de spécifier comment la gestion des sous-contractants se transforme dans le cadre d'une architecture orientée-service.

B- L'importance du plan de gestion intégré

À la section 2.7 (voir le SWEBOK, chapitre sur la gestion de projet) dans cette section du SWEBOK, on discute de l'importance d'un plan de gestion. Dans le cadre d'une architecture orientée-service, il faudra tenir compte de la notion de *sous plans* pour arrimer la réalité de ce type de projet qui implique, typiquement, plus d'une équipe de développement. Or, il peut être utile d'établir un plan de gestion intégré. De cette façon il sera possible de gérer chacun des *sous-projets* (correspondant à un/des sous-système(s)) dans le cadre d'un projet d'ensemble qui comprend à la fois les *systèmes consommateurs* et les *services*.

Contribution aux objectifs:

[GERER] : Aider à saisir l'importance du plan de gestion intégré dans le cadre d'une AOS.

[SPECIFIER] : Il s'agit de spécifier comment le plan de gestion peut être important dans le cadre de l'AOS.

4.5.7 Gestion de la Configuration du Logiciel Orienté Service

Voici tout d'abord le découpage des thèmes pour le domaine des connaissances de la gestion de la configuration du logiciel, tel que défini par le SWEBOK, auquel le lecteur peut faire référence.

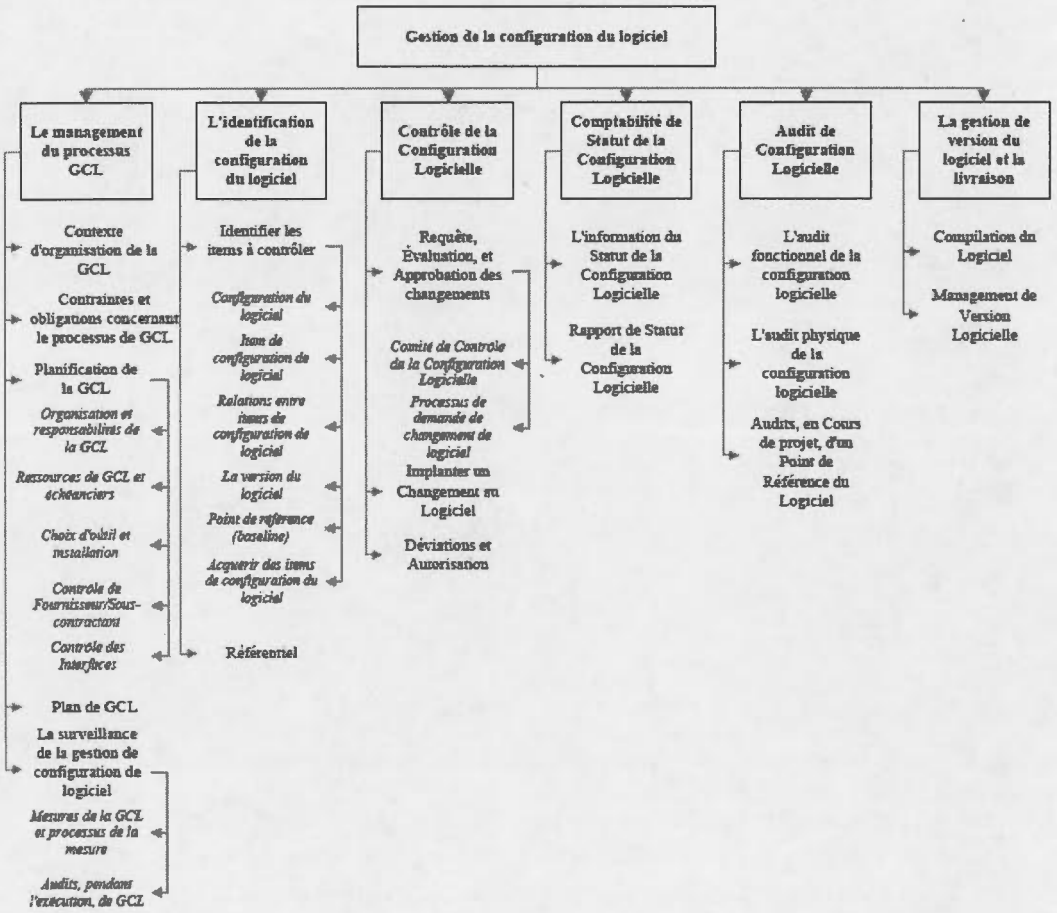


Figure 28 - le découpage des thèmes pour le domaine des connaissances de la gestion de configuration

A- Contrôle de fournisseurs/sous-contractant

Dans le cadre de la gestion de configuration, il serait intéressant de tenir sous cette gestion les livrables contractuels. Puisque plus d'un client (système consommateur) peut, à terme, utiliser tel ou tel service, il est fort possible que ce contrat soit mis à jour avec le temps. Or l'historique, ou traçabilité a une importance en génie logiciel. De plus, lorsqu'un contrat est modifié, typiquement, le ou les services associés seront aussi modifiés. Il faut, le plus possible, tenir la gestion de configuration pour ces artéfacts inter reliés.

Contribution aux objectifs:

[SUPPORTER] : Supporter le responsable de la gestion de la configuration dans un cadre d'AOS afin qu'il tienne compte du rôle de fournisseurs et sous-contractant.

B- Contrôle des interfaces

La notion d'interface, dans une architecture orientée-service, est fondamentale. Ces interfaces sont, en quelques sortes, les contrats technologiques des services. Or, ces interfaces doivent être gérées en parallèle avec le/les contrat(s) et service(s) correspondant(s).

Contribution aux objectifs:

[SPECIFIER] : Spécifier l'importance du contrôle des interfaces dans le cadre d'une AOS.

C- Notion de version du logiciel

La gestion de version du logiciel est fondamentale en génie logiciel mais peut s'avérer lourde dans une architecture orientée-service puisque le nombre de composantes versionnées peut être assez grand. De plus, certaines versions d'un certain service peuvent être d'une grande importance puisque l'interface ou le comportement change en vertu d'un nouveau contrat. Dans ces cas, il faudra établir un lien de communication entre les groupes de développement impliqués dans une architecture orientée-service. Pour les autres versions de service (celles qui n'ont pas d'impact réel sur les systèmes consommateurs), elles n'ont pas à être communiquées à prime abord. Certaines technologies fondamentales à l'AOS ne sont pas orientées vers la notion de gestion de configuration, comme par exemple les services web. Or il faut mettre sur pied un mécanisme au niveau de l'équipe de gestion de configuration pour supporter cette notion de versionnement de service.

Contribution aux objectifs:

[SPECIFIER] : Spécifier l'importance du principe de versionnement dans le cadre d'une AOS.

D- Référentiel

La notion de référentiel de services est déjà fortement populaire avec l'arrivée du UDDI (Universal Description, Discovery and Integration). Par contre, ce référentiel permettant d'améliorer, ou de supporter, la *découvertabilité* n'a pas du tout le même sens que le référentiel de gestion de configuration. Bien qu'il peut y exister un lien, le but est totalement différent. Le premier a pour but de rendre disponible les interfaces de divers services sur le réseau prêtes à être découvertes (on découvre l'interface pour accéder au service) par d'éventuel systèmes consommateurs, l'autre à pour but de

gérer la configuration (incluant les principes de versionnement de composants logiciels).

Contribution aux objectifs:

[SPECIFIER] : Spécifier la différence entre les différentes notions de référentiels dans le cadre d'une AOS.

4.6 Synthèse des liens entre objectifs et contributions

Le tableau suivant a été construit à partir de statistiques établies en effectuant un dénombrement des contributions de l'auteur dans ce chapitre, en fonction des objectifs associés et des champs d'expertise du génie logiciel (qui correspondent directement à des sections du SWEBOK, par ailleurs). Les champs d'expertise du génie logiciel peuvent aussi être référencés à titre d'étapes du génie logiciel.

Tableau 1 - Synthèse des liens entre objectifs et contributions

	Exigences	Conception	Construction	Essai	Maintenance	Gestion	Configuration	Total
SUPPORTER	1	1	1	1			1	5
SPECIFIER	2	5	2	1		2	3	15
MODELISER	1	2	1					4
EXCEPTION		1	1					2
COMPRENDRE	1	1						2
CONCEVOIR		5						5
ENCADRER	1	3	2	1				7
GÉRER	2			2	1	1		6
ANALYSER	2							2
Total	10	18	7	5	1	3	4	48

Dans le diagramme suivant on illustre le nombre de contributions aux objectifs en fonction des étapes du génie logiciel. En ce sens, on remarque que les étapes « exigences », « conception » et « construction » ont relativement beaucoup d'importance.

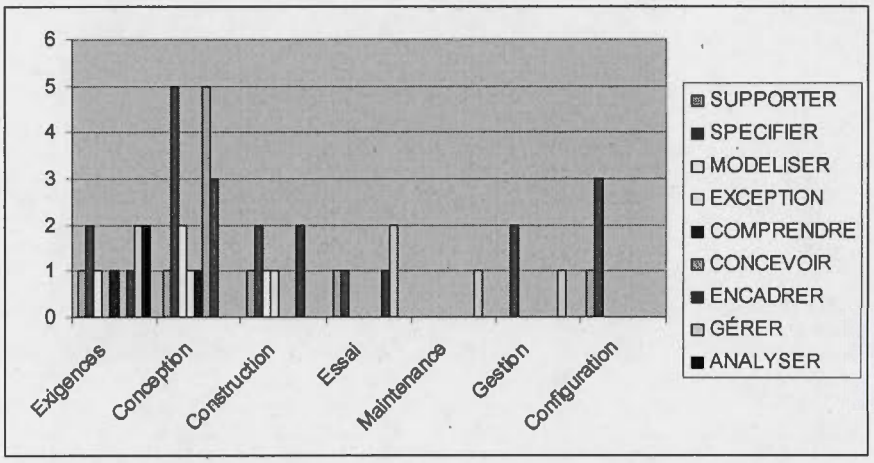


Figure 29 - Corrélation entre les étapes du cycle de vie du logiciel et les contributions du mémoire

Ainsi, les contributions présentées dans ce mémoire sont très fortement axées sur les étapes liées au pré-développement. C'est-à-dire qu'une importante part des contributions concerne les travaux d'analyse et de conception. En effet, ces deux premières étapes comptent 28 contributions sur un total de 48, donc 58%. Si on ajoute l'étape de construction, on atteint 73%.

Posons comme hypothèse que chacune des étapes du génie logiciel sont égales entre elles. Dans le diagramme suivant, la différence entre le nombre de contributions par étapes du génie logiciel et le nombre hypothétiquement moyen est illustrée.

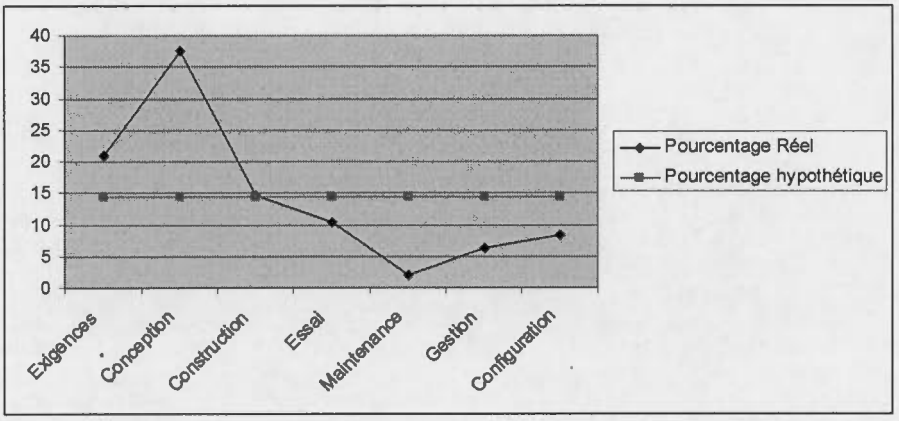


Figure 30 - Différences avec l'hypothèse d'égalité en importance de toutes les étapes du cycle de vie

En considérant donc hypothétiquement que chacune des étapes abordées sont égales entre elles, ont atteindrait 14% chacune. C'est donc dire que seules les étapes « exigences », « conception », et même « construction », semblent avoir été abordées avec plus d'attention (que la moyenne hypothétique) par l'auteur. Cette observation appuie celle effectuée précédemment. Les étapes liées au pré-développement ont fait l'objet de plus de contributions que les autres étapes.

Le tableau suivant illustre le dénombrement des contributions effectué en fonction des objectifs.

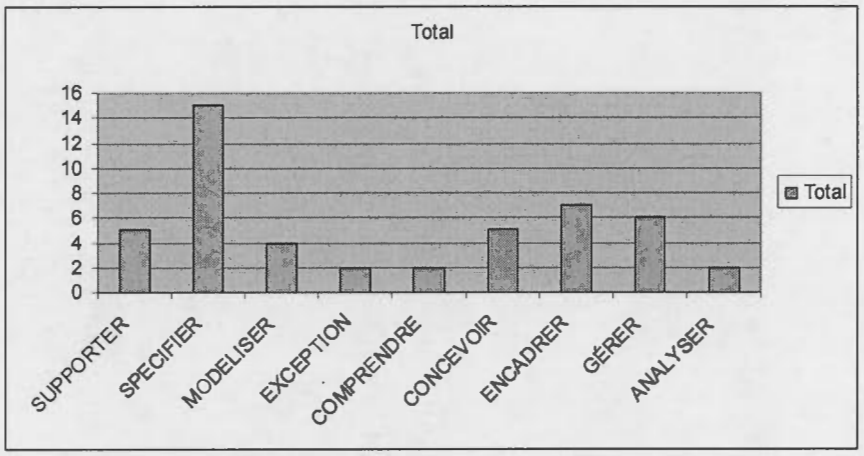


Figure 31 - Nombre de contributions par objectifs

En considérant chaque objectif, on se rend compte que c'est l'objectif « spécifier » qui reçoit le plus d'attention, suivi des objectifs « encadrer » et « gérer ».

Ces observations semblent consistantes avec les pratiques que l'on situe au niveau II du CMMI (voir le livre *CMMI for Outsourcing: Guidelines for Software, Systems, and IT Acquisition*) [HOFM2008]. L'auteur croit, quant à lui, que les étapes « exigences », « conception » et « construction » sont les étapes les plus fondamentales du cycle de vie du logiciel puisqu'elles déterminent le quoi (analyse), le comment (conception), ainsi que la réalisation (construction) proprement dite. On peut donc déduire que les résultats observés sont en corrélation avec cette dernière croyance de l'auteur.

CONCLUSION

Avec le temps, l'informatique d'intégration d'applications d'entreprise a pris différentes tangentes. Tantôt orientées vers la centrale IBM, tantôt orientées vers des systèmes Unix, et plus tard Windows NT, les solutions sont très diversifiées. Il est d'ailleurs très intéressant de noter que la plupart de ces solutions, telles que vues dans ce mémoire, sont d'ailleurs demeurées actuelles au fil du temps, jusqu'à aujourd'hui même. Ceci est un bon présage pour les nouvelles techniques et technologies liées aux architectures orientées-service.

Après avoir élaboré sur l'état de l'art et les aspects théoriques, les contributions de ce mémoire ont été avancées par l'auteur à titre de suggestions à un éventuel guide sur l'architecture orientée-service. Malgré tout, elles proviennent du meilleur des connaissances de l'auteur et ont pour but de servir le développement éventuel de ce futur guide, donc de servir, d'une façon plus large, le développement des connaissances de ce champs de recherche qu'est l'architecture orientée-service.

Les contributions de l'auteur, proposées dans ce mémoire, sont limitées par les connaissances, les références et la seule expérience de l'auteur. En revanche, ces contributions proposent clairement des avenues pour poursuivre ce travail de recherche.

Le commentaire de validation proposant que des pistes de solutions concrètes soient plus présentes au niveau des contributions est une invitation à les définir à un niveau plus technique. Par contre, comme on peut remarquer dans la synthèse des contributions, des solutions peuvent aussi être au niveau de la gestion (i.e.: objectif GÉRER) et au niveau de l'encadrement (i.e.: objectif ENCADREMENT), donc à des

niveaux moins technique. La question est ouverte à savoir si des contributions à des niveaux différents que le niveau technique sont aussi des pistes de contributions concrètes à l'architecture orientée-service.
Il est intéressant de constater que les évaluations ont donné suite à des commentaires forts différents dépendamment du bagage propre aux évaluateurs en question. Nous les remercions d'ailleurs beaucoup pour le temps qu'ils ont mis, gratuitement, à réviser parfois très en détail le contenu de ce mémoire.

L'architecture orientée-service est un champ d'expertise très intéressant et l'auteur se considère chanceux d'avoir pu le découvrir plus en détail dans le cadre de l'élaboration de ce mémoire. À l'heure du dépôt de ce mémoire, l'AOS est encore très en vogue. Il est prévisible qu'il prendra ensuite d'autres formes, mais il sera probablement toujours question de définir des architectures qui servent l'intégration entre les systèmes.

Finalement, du point de vue de l'auteur, il serait très souhaitable qu'un jour, un groupe d'individus mandatés approche la question encore plus formellement pour donner lieu à un véritable guide sur l'architecture orientée-service neutre technologiquement.

6.1 Perspectives d'utilisation d'un tel guide

Tel que suggéré par un évaluateur, certaines entreprises pourraient tirer avantage de ce guide afin de mettre en place des parcours méthodologiques liées à l'AOS. Même si ce guide n'est pas une méthodologie, il peut servir de référence, jusqu'à un certain point, afin d'orienter la configuration d'une méthodologie pour le domaine étudiée.
D'autres circonstances peuvent s'avérer positives pour l'utilisation de ce guide. Il s'agira principalement de projets portant sur l'AOS. Il est évident que des équipes de développement d'AOS pourraient être intéressés à lire ce guide.

Dans un contexte de formation, d'autres trouveront un survol d'information assez complet sur l'intégration de système du début de l'informatique à l'AOS.

Finalement, dans le perspective d'une thèse de doctorat en génie logiciel, une personne pourrait poursuivre le travail en procédant à une évaluation approfondie des pratiques pour consolider la méthodologie proposée.

6.2 Les suites à donner

Il serait important de considérer la possibilité de faire une version anglaise du guide et de la publier dans les deux langues sur Internet. Une synthèse pourrait également en être faite sous la forme d'un article à publier dans une conférence du domaine du génie logiciel.

BIBLIOGRAPHIE

[ABRA2004]

- Alain Abran, James W. Moore, Pierre Bourque, Robert Dupuis, 2004, «Guide to the Software Engineering Body of Knowledge», IEEE Computer Society, p. 1-204

[AGRA2001]

- Rakesh Agrawal, Roberto J. Bayardo Jr., Daniel Gruhl, Spiros Papadimitriou, 2001, « Vinci: A Service-Oriented Architecture for Rapid Development of Web Applications », ACM, ACM 1-58113-348-0/01/0005, p. 357-365

[ALEX1996]

- Christopher Alexander, 1996, « The Origins of Pattern Theory », ACM, 1996 ACM Conference on Object-Oriented Programs, Systems, Languages and Applications (OOPSLA), http://www.patternlanguage.com/archive/ieee/ieeetext.htm

[AMIR2004]

- Rafik Amir, Dr. Amir Zeid, 2004, « A UML Profil For Service Oriented Architectures », ACM 1-58113-833-4/04/0010, p. 192, 193

[ANDE2007]

- Tyler Anderson, 2007, « Understanding Web Services specifications, Part 5: WS-Policy », International Business Machines, http://www-128.ibm.com/developerworks/webservices/edu/ws-dw-ws-understand-web-services5.html?S_TACT=105AGX04&S_CMP=HP

[ARSA2002]

- Ali Arsanjani, David Ng, 2002, « Business Compilers: Towards Supporting a Highly Re-configurable Architectural Style for Service-oriented Architecture », ACM, ACM 1-58113-626-9/02/0011, p. 26-27

[ARSA2007]

- Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, Kishore Channabasavaiah, 2007, «Design an SOA solution using a reference architecture», IBM, http://www-128.ibm.com/developerworks/architecture/library/ar-archtemp/

"the current Web services specifications generally lack the facility to define comprehensive relationships among business entities"

[BARE2003]

- Luciano Baresi, Reiko Heckel, Sebastian Thone, Daniel Varro, 2003, « Modeling and Validation of Service-Oriented Architecture: Application vs. Style », ACM, ACM 1-58113-743-5/03/0009, p. 68-77

[BEAM2000]

- 2000, « BEA MessageQ, Programming Guide », BEA, p. 1-474, http://edocs.bea.com/tuxedo/msgq/pdf/mqprog.pdf

[BECK2007]

- Helen Beckett, 2007, «Case studies in implementing service oriented architecture», Computer Weekly, http://www.computerweekly.com/Articles/2007/04/17/222985/case-studies-in-implementing-service-oriented-architecture-soa.htm

[BETIN2005]

- Aysu Betin-Can, Tevfik Bultan, Xiang Fu, 2005, « Design for Verification for Asynchronously Communicating Web Services », International World Wide Web Conference Committee (IW3C2), ACM 1-59593-046-9/05/0005, p. 750-759

[BOOC1999]

- Booch, G., I. Jacobson and J. Rumbaugh, 1999, «The Unified Modeling Language User Guide», Addison-Wesley, pp. 219-241

[BOOT2004]

- David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard, 2004, « Web Services Architecture », W3C, http://dev.w3.org/cvsweb/~checkout~/2002/ws/arch/wsa/wd-wsa-arch-review2.html

[CASA2003]

- Fabio Casati, Eric Shan, Umeshwar Dayal, Ming-Chien Shan, 2003, « Business-Oriented Management of Web Services », Communications of the ACM, no 10, Vol. 46, Octobre, p. 55-60

[CHAV2004]

- Kamalsinh F Chavda, 2004, « Anatomy of a Web Service », Kennesaw State University, JCSC 19, 3

[COTR2004]

- Domenico Cotroneo, Almerindo Graziano, Stefano Russo, 2004, « Security Requirements in Service Oriented Architectures for Ubiquitous Computing », ACM, ACM 1-58113-951-9, p. 172-177

[COXP2000]

- Ryan Cox, Joaquin Picon, Gianni Scenini, Andrea Conzett, 2000, « Component Broker 3.0: First Steps », International Business Machines, p.1-441, ISBN 0738419184

[HOFM2008]

- Hubert F. Hofmann, Deborah K. Yedlin, John W. Mishler, Susan Kushner, 2008 « CMMI for Outsourcing: Guidelines for Software, Systems, and IT Acquisition », Carnegie Mellon University, p.1-464, ISBN-10: 0-321-47717-0, ISBN-13: 978-0-321-47717-0

[CURBE2003]

- Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai, Sanjiva Weerawarana, 2003, « The next step in web services », ACM, Vol. 46, no. 10, p. 29-34

[DENG2005]

- Roger Deng, 2005, «Service-Oriented ETL Architecture», www.datawarehouse.com, http://www.datawarehouse.com/article/?articleid=5560

[GAVI2003]

- Lee Gavin, Gerd Diederichs, Piotr Golec, Hendrik Greyvenstein, Ken Palmer, Skreekumar Rajagopalan, Arvind Viswanathan, 2003, « An EAI Solution using WebSphere Business Integration (V4.1) », International Business Machines, p. 1-563, ISBN 0738426547

[GERA1999]

- Ronan Geraghty, Sam Joyce, Tom Moriarty, Gary Noone, 1999, «COM-CORBA Interoperability», Prentice Hall, p. 1-304

[GRUM2007]

- Galen Gruman, 2007, «The real challenge of SOA», Techworld, http://www.techworld.com/itevolution/features/index.cfm?FeatureID=2574

[GUDG2006]

- Martin Gudgin, Marc Hadley, Tony Rogers, Ümit Yalçinalp, 2006, « Web Services Addressing 1.0 – WSDL Binding », W3C, http://www.w3.org/TR/2006/WD-ws-addr-wsdl-20060216/

[HINC2006]

- Dion Hinchcliffe, 2006, «Web 2.0 and SOA: Contrived or Converging?», web2.wsj2.com, http://web2.wsj2.com/web_20_and_soa_contrived_or_converging.htm

[HOGG2004]

- K. Hogg, P. Chilcott, M. Nolan, B. Srinivasan, 2004, « An Evaluation of Web Services in the Design of a B2B Application », Australian Computer Society, Inc. - Conferences in Research and Practice in Information Technology, Vol. 26, 331-340

[IMAM2005]

- Takeshi Imamura, Michiaki Tatsubori, Yuichi Nakamura, Christopher Giblin, 2005, « Web Services Security Configuration in a Service-Oriented Architecture », IBM/ACM, ACM 1-59593-051-5/05/0005, p. 1120-1121

[JACKS2004]

- Joab Jackson, 2004, «An insider's thoughts on SOA», GCN, http://www.gcn.com/print/24_28/36976-1.html

[JPOS1985]

- J. Postel, J. Reynolds, 1985, « FILE TRANSFER PROTOCOL (FTP) », Network Working Group, Request for Comments: 959, p. 1-69

[KEEN2006]

- Martin Keen, Chris Backhouse, Jim Hollingsworth, Stephen Hurst, Mark Pocock, 2006, « Architecting Access to CICS within an SAO », International Business Machines, p. 1-428, ISBN 0738496952

[KOPE2007]

- Jacek Kopecky, Carine Bournez, Eric Prud'hommeaux, 2007, « Semantic Annotations for Web Services Description Language Working Group », W3C, http://www.w3.org/2002/ws/sawsdl/

[LITT2003]

- Mark Little, 2003, « Transactions and Web Services », Communications of the ACM, no 10, Vol. 46, p. 49-54

[LYNC2006]

- Regina Lynch, 2006, «Wanted: A Service-Oriented Architecture Methodology», TheServerSide.com, http://www.theserverside.com/news/thread.tss?thread_id=40256

"Web services appears to be lacking a methodology that addresses the unique attributes of an SOA project"

[MARC2003]

- Esperanza Marcos, Valeria de Castro, Belén Vela, 2003, « Representing Web Services with UML: A Case Study », IC-SOC, LNCS 2910, p. 17-23

[MERE2003]

- L.G. Meredith, Steve Bjorg, 2003, « Contracts and Types », Communications of the ACM, no 10, Vol. 46, Octobre, p. 45-47

[MUKH2004]

- Nirmal K Mukhi, Ravi Konuru, Francisco Curbera, 2004, « Cooperative Middleware Specialization for Service Oriented Architectures », ACM, ACM 1-58113-912-8/04/0005, p. 206-215

[MURP2004]

- Richard C. Murphy, 2004, « The Architecture of Services and the Phenomenon of Life », Fawcette Technical Publications, 30 mars 2004, http://ftponline.com

[NADH2004]

- Easwaran G. Nadhan, 2004, «Service-Oriented Architecture: Implementation Challenges», Microsoft, http://msdn2.microsoft.com/en-us/library/aa480029.aspx

[NAKA2004]

- Masahide Nakamura, Hiroshi Igaki, Haruaki Tamada and Kenichi Matsumoto, 2004, « Implementing Integrated Services of Networked Home Appliances Using Service Oriented Architecture », ACM, ACM 1581138717/04/0011, p. 269-278

[NATI2003-A]

- Yefim V. Natis, Roy W. Schulte, 2003, « Introduction to Service-Oriented Architecture », Gartner, SPA-19-5971, p. 1-6

[NATI2003-B]

- Yefim Natis, 2003, « Service Oriented Architecture Scenario », Gartner, AV-19-6751, p. 1-4

[NORT2007]

- Philip Norton, 2007, « Invoke Web services with WebSphere MQ and WebSphere Enterprise Service Bus », International Business Machines, http://www-128.ibm.com/developerworks/webservices/edu/ws-dw-ws-mq-esb.html?S_TACT=105AGX04&S_CMP=HP

[OBJE2006]

- Object Management Group, 2006, «CORBA Component Model Specification», OMG, Version 4.0 , p. 1-335, http://www.omg.org/docs/formal/06-04-01.pdf

[OPEN1997]

- Open Group, 1997, « Introduction to the RPC Specification », The Open Group, http://www.opengroup.org/onlinepubs/9629399/chap1.htm

[OREI2005]

- Tim O'Reilly, 2005, « What Is Web 2.0 », O'Reilly, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[PAPA2003]

- M. P. Papazoglou, D. Georgakopoulos, 2003, « Service-oriented computing: Introduction », Communications of the ACM, no 10, Vol. 46, p. 24-38

[PATR2005]

- Paul Patrick, 2005, « Impact of SOA on Enterprise Information Architectures », BEA/ACM, ACM 1-59593-060-4/05/06, p. 844-848

[POST2003]

- Jon Postel, Traduction par Valéry G. FREMAUX, 2003, « RFC 793 : Transmission Control Protocol (TCP) - Specification », Department of Defense – US Gov. p.1-52, http://abcdrfc.free.fr/rfc-vf/rfc793.html

[PUTT2000]

- Geert Van de Putte, Colin Brett, Paul Sehorne, Sharon Stubblebine, 2000, « Business Integration Solutions with MQSeries Integrator », International Business Machines, p. 1-267, ISBN 0738417254

[ROSS2006]

- Steve Ross-Talbot, Tony Fletcher, 2006, « Web Services Choreography Description Language: Primer », W3C, http://www.w3.org/TR/2006/WD-ws-cdl-10-primer-20060619/ .

[SEEL2006]

- Rich Seeley, 2006, «OASIS panel ponders: What is SOA», SearchWebServices.com, http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci1187553,00.html,"Lack of clarity in defining SOA makes it difficult for software architects to sell it to the business side of their organizations"

[SIMMO2006]

- Scott Simmons, 2006, «Scott Simmons: SOA governance and the prevention of service-oriented anarchy», IBM, http://www-128.ibm.com/developerworks/websphere/techjournal/0609_col_simmons/0609_col_simmons.html

[SUNM1997]

- Sun Microsystems, 1997, «RMI Architecture and Functional Specification», Sun Microsystems, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html

[TAYLO]

- Kerry Taylor, Tim Austin, Mark Cameron, « Charging for Information Services in Service-Oriented Architectures », ACM

[WACK1999]

- Dieter Wackerow, David Armitage, Tony Skinner, 1999, « MQ Series 5.1 Administration and Programming Examples », International Business Machines, p.1-255, SG24-5849-00

[WEBM2006]

- webMethods.com, 2006, «webMethods SOA Roadmap», webMethods, p. 1-2, http://www1.webmethods.com/PDF/datasheets/SOA_Roadmap_datasheet.pdf

[WHAL2003]

- Ueli Wahli, Wouter Denayer, Lars Schunk, Deborah Shaddon, Martin Weiss, 2003, « EJB 2.0 Development with WebSphere Studio Application Server », International Business Machines, p. 1-725, ISBN 0738426091

[WHAL2006]

- Ueli Wahli, Owen Burroughs, Owen Cline, Alec Go, Larry Tung, 2006, « Web Services Handbook for WebSphere Application Server 6.1 », International Business Machines, p. 1-782, ISBN 0738494909

[WINET1971]

- Joel M. Winett, 1971, «The Definition of a Socket», Lincoln Laboratory, Request for Comment 147, NIC 6750, http://tools.ietf.org/rfc/rfc147.txt

[YANG2003]

- Jian Yang, 2003, « Web Service Componentization », Communications of the ACM, no 10, Vol. 46, Octobre, p. 35-40

[ZHAN2004]

- Jia Zhang, Jen-Yao Chung, Carl K. Chang, 2004, « Migration to Web Services Oriented Architecture », ACM Symposium on Applied Computing, ACM 1-58113-812-1/03/04, p. 1624, 1628