

Cycle de formation des ingénieurs en Télécommunications

Option :

Réseaux et Services Mobiles

Rapport de Projet de fin d'études

Thème :

Conception et développement de la publication des services d'intermédiation documentaire via des services web

Réalisé par :

M. Rached Achouri

Encadrants :

M. Sami Tabbane (SUP'COM)

M. Hichem Maatallah (TUNISIE TRADNET)

Travail proposé et réalisé en collaboration avec



Année universitaire : 2006/2007

Dédicaces

A ma famille,
A mes amis,
A tous ceux qui me sont chers

Je dédie ce travail

Résumé

Les architectures orientées service (SOA) vont s'imposer au cours des prochaines années. Elles trouveront particulièrement leur place dans les contextes d'applications d'entreprise.

En effet, L'architecture orientée services est une méthode de conception basée sur des standards permettant de créer une infrastructure informatique intégrée capable de répondre rapidement aux nouveaux besoins d'une entreprise. Elle fournit les principes et directives permettant de transformer un réseau existant de ressources informatiques hétérogènes, distribuées, complexes et rigides en ressources intégrées, simplifiées et particulièrement souples pouvant être modifiées et combinées afin de mieux satisfaire les objectifs de l'entreprise.

Les acteurs les plus présents dans le domaine des SOA sont ceux qui se sont positionnés parmi les premiers autour des services web, et qui proposent des solutions matures à ce niveau.

Ce projet présente une initiative pour la conception et le développement de la publication des services d'intermédiation documentaire, ceci en développant une application qui se base à la fois sur les avantages et les caractéristiques de l'architecture Orientée Service (SOA) et les Services Web.

Mots clés: Architecture Orientée Services (SOA), Services Web, Middleware, Entreprise Service Bus (ESB), intermédiation documentaire

Avant Propos

Le travail présenté dans ce rapport a été réalisé à T.T.N (Tunisie TradeNet) qui est une société anonyme qui a démarré le 1er novembre 2000 et qui a pour principale mission de gérer le centre serveur de la Liasse Unique, dans le cadre de notre projet de fin d'études du cycle d'Ingénieur en Télécommunications à l'Ecole Supérieure des Communications de Tunis (SUP'COM).

Au terme de ce projet, mes premiers et mes plus vifs remerciements seront pour Mr. Sami Tabbane, professeur à Sup'Com, pour m'avoir soutenu durant la période de mon projet, pour tout son dynamisme et ses compétences scientifiques qui m'ont permis de mener à bien ce travail.

Mon plus grand remerciement s'adresse aussi à Mr. Hichem Maatallah, ingénieur principal à Tunisie TradeNet pour son encouragement, sa disponibilité et pour tout son aide et son savoir faire qu'il a mis à ma disposition.

*J'*adresse ma plus vive reconnaissance à tous mes enseignants de SUP'COM pour la formation d'ingénieur qu'ils m'ont procurée.

Je remercie tous ceux qui n'ont épargné le moindre effort, de près ou de loin, pour me permettre d'accomplir mon projet et j'espère que ça sera le bon départ pour ma carrière d'Ingénieur.

Pour conclure, je remercie profondément les membres de jury qui ont accepté d'évaluer mon projet.

Rached

Table des matières

Introduction Générale	1
Chapitre 1: Concepts de base	3
Introduction	3
1.1. Architecture Orientée Service (SOA)	4
1.1.1. Définition	4
1.1.2. Services	4
1.1.2.1. Couplage faible de service	4
1.1.2.2. Transmission de messages	5
1.1.3. Conclusion	5
1.1.4. Entreprise Service Bus (ESB)	5
1.1.4.1. Qu'est ce que Entreprise Service Bus (ESB)	7
1.1.4.2. Caractéristiques et capacités d'ESB	8
1.1.4.2.1. Transformation de message	8
1.1.4.2.2. Routage de message	8
1.1.4.2.3. Capacité de découvrir des fautes	9
1.1.4.2.4. Fiabilité et sécurité	9
1.1.4.3. Conclusion	9
1.2. Les services Web	9
1.2.1. Définition	10
1.2.2. Principe fondamental du service Web	10
1.2.3. La couche de technologie dans un service Web	11
1.2.3.1. Découverte	12
1.2.3.2. UDDI (Universal Description, Discovery et Integration)	12
1.2.3.3. Description	12
1.2.3.4. WSDL (Web Services Description Language)	12
1.2.3.5. Empaquetage	13
1.2.3.6. SOAP (Simple Object Access Protocol)	13
1.2.3.7. Transport	13
1.2.3.8. Réseau	13
1.2.3.9. Application	14
1.2.4. Architecture des services Web	14
Conclusion	15
Chapitre 2: Etude de l'existant et spécification des besoins	16
Introduction	16
2.1. Etude et critique de l'existant	16
2.2. Spécification des besoins	17
2.2.1. Les besoins fonctionnels	17
2.3. Les Acteurs	18
2.4. Diagramme des cas d'utilisation	18
2.4.1. Diagramme des cas d'utilisation général	18
2.4.2. Diagramme des cas d'utilisation « Administration »	20
2.4.3. Diagramme des cas d'utilisation « Déploiement de nouveaux services »	20
2.4.4. Diagramme des cas d'utilisation pour le cas du service TCE	21
2.4.4.1. Diagramme des cas d'utilisation « Domiciliation »	22
2.5. Scénarios décrivant le fonctionnement global du service TCE	23
Conclusion	24

Chapitre 3 : Conception de la plateforme	25
Introduction	25
3.1. Architecture générale d'une plateforme SOA	25
3.1.1. Services ESB	26
3.1.2. Services d'entreprise connectés	26
3.1.3. Communications et processus ESB	26
3.2. Décomposition en packages	27
3.2.1. Description des liens	28
3.3. Conception détaillée	29
3.3.1. Package « XML Manager ».....	29
3.3.2. Package « Validation Manager ».....	29
3.3.3. Package « Buisness »	30
3.3.4. Package « Administration »	31
3.4. Diagrammes de séquence	32
3.4.1. Diagramme de séquence pour un scénario du package « XML Manager ».....	32
3.4.2. Diagramme de séquence pour un scénario du package « Validation »	33
3.4.3. Diagramme de séquence pour un scénario du package « Business ».....	34
Conclusion.....	35
Chapitre 4 : Réalisation	36
Introduction	36
4.1. Description de l'environnement de travail	36
4.1.1. Environnement technique	36
4.1.1.1. Présentation et architecture générale de Mule.....	37
4.1.2. Environnement logiciel	40
4.1.2.1. Les langages de développement	40
4.1.2.2. Outils CASE	41
4.1.2.2.1. PowerDesigner 9	41
4.1.2.3. Environnement de développement	41
4.1.2.4. Serveur web Axis (Boite à outils Apache Axis)	42
4.1.2.5. Protocole de communication Client/Serveur.....	42
4.1.2.6. Système de Gestion de Base de Donnée MySQL	42
4.2. Test du fonctionnement de la plateforme	42
Conclusion	45
Conclusion générale	46
Annexe A : Architecture des services web	47
A.1. Qu'est-ce que SOAP?	47
A.2. Messages SOAP	48
A.3. Eléments du Système	51
Annexe B : XML (eXtensible Markup Language)	54
B.1. Présentation	54
B.2. Mise en page de XML	54
B.3. Structure des documents XML	55
B.4. Décodage d'un document XML	55
B.5. Avantages de XML	55
Bibliographie.....	57

Table des figures

Figure 1.1. Liaison explicite entre les services	6
Figure 1.2. Liaison des services à travers un middleware	7
Figure 1.3. Architecture générale d'un ESB	7
Figure 1.4. Protocole Switch	8
Figure 1.5: Un Service Web permet l'accès au code de l'application en utilisant les standards de la technologie Internet	10
Figure 1.6: Le service Web offre un couche d'abstraction entre l'application client et le code de l'application.....	11
Figure 1.7: Pile de technologies dans les services Web.	11
Figure 1.8: Rôles et opérations pour les Services Web.....	14
Figure 2.1. Diagramme de cas d'utilisation général.....	19
Figure 2.2. Diagramme des cas d'utilisation « Administration».....	20
Figure 2.3. Diagramme des cas d'utilisation « Déploiement de nouveaux services»	20
Figure 2.4. Diagramme des cas d'utilisation du service TCE.....	21
Figure 2.5. Diagramme des cas d'utilisation « Domiciliation».....	22
Figure 2.6. Diagramme de séquence décrivant le scénario : « dossier de domiciliation accepté ».	23
Figure 2.7. Diagramme de séquence décrivant le scénario : « demande de complément d'informations	24
Figure 3.1. Architecture d'une solution SOA	25
Figure 3.2. Diagramme des packages	28
Figure 3.3. Diagramme de classe pour le package « XML Manager ».....	29
Figure 3.4. Diagramme de classe pour le package « Validation Manager ».....	30
Figure 3.5. Diagramme de classe pour le package « Buisiness ».....	30
Figure 3.6. Diagramme de classe pour le package « Administration ».....	31
Figure 3.7. Diagramme de séquence pour un scénario du package « XML Manager »	32
Figure 3.8. Diagramme de séquence pour un scénario du package « Validation ».....	33
Figure 3.9. Diagramme de séquence pour un scénario du package « Business »	34
Figure 4.1. Architecture générale de Mule	38
Figure 4.2. Contenu du fichier TCE.bat	43
Figure 4.3. Fichier de configuration pour le service TCE.....	43
Figure 4.4. Exécution du service TCE	44
Figure 4.5. Réponse du service TCE après son invocation	45
Figure A.1. Architecture des services web	47
Figure A.2. Message SOAP pour l'exemple d'un service qui double la valeur d'un entier	49
Figure A.3. Processus côté client	52
Figure A.4. Processus côté serveur	53

Liste des Abréviations

ASA	A pplication S erver A pplications
BPEL	B usiness P rocess E xecution L anguage
CML	C hemical M arkup L anguage
CSS	C ascading S ty S heet
DAML	D ARPA A gent M arkup L anguage
DTD	D ocument T ype D efinition
ESB	E ntreprise S ervice B us
HTML	H ypertext M arkup L anguage
HTTP	H ypertext T ransfer P rotocol
IIOB	I nternet I nter- O RB P rotocol
IP	I nternet P rotocol
J2EE	J ava 2 E ntreprise E dition
JCA	J 2EE C onnect O r A rchitecture
JMS	J ava M essage S ervice
MathML	M athematical M arkup L anguage
OFX	O pen F inancial e Xchange
OOM	O bject O riented M iddleware
RDF	R essource D escription F ramework
SGBD	S ystème de G estion de B ase de D onnées
SMIL	S ynchronized M ultimedia I ntegration L anguage
SMTP	S imple M ail T ransfer P rotocol
SOA	S ervice O riented A rchitecture
SOAP	S imple O bject A ccess P rotocol
TCE	T itre de C ommerce E xtérieur
TCP	T ransmission C ontrol P rotocol
UDDI	U niversal D escription D iscovery and I ntegration

UML	Unified Modeling Language
WSDL	Web Service Description Language
XML	eXtensible Markup Language
XML-RPC	XML-Remote Procedure Call
XMPP	eXtensible Messaging and Presence Protocol
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language Transformation

Introduction Générale

Dans une entreprise, le département informatique doit gérer des environnements de plus en plus complexes. À mesure que l'entreprise évolue, il doit s'assurer que les technologies en place répondent toujours aux besoins. Si ce n'est pas le cas, la réactivité de l'entreprise sera compromise.

Le département informatique n'est généralement pas confronté à un problème de fonctionnalités insuffisantes, mais plutôt au fait que des systèmes très importants pour l'entreprise, tels que des logiciels de gestion de la relation client et des progiciels de gestion intégrés fonctionnent séparément d'autres systèmes vitaux, même si les processus métiers se répartissent souvent sur plusieurs applications. L'intégration des informations et des processus est donc nécessaire pour obtenir une vue globale d'un processus métier complexe. Jusqu'à présent, cela nécessitait des interventions manuelles laborieuses ou le développement de solutions spécifiques, rigides et difficiles à gérer.

L'orientation services permet d'organiser des ressources informatiques distribuées dans une solution intégrée, éliminant ainsi les informations isolées et conférant davantage de souplesse à l'entreprise. L'orientation services organise les ressources informatiques en modules, créant des processus métier faiblement couplés qui intègrent des informations provenant de différents systèmes. L'un des facteurs de réussite d'une architecture orientée services est la création de processus métier peu soumis aux contraintes de l'infrastructure informatique sous-jacente, afin de fournir à l'entreprise toute la latitude dont elle a besoin.

L'architecture orientée services (SOA – Service Oriented Architecture) permet de créer toute une nouvelle génération d'applications dynamiques (parfois nommées applications composites). Ces applications offrent aux utilisateurs des informations plus précises et plus complètes ainsi qu'une meilleure connaissance des processus, mais aussi la possibilité d'accéder à ces informations en utilisant la forme et la présentation les mieux adaptées, que ce soit par le biais du Web, d'un client riche ou d'un dispositif mobile. Les applications dynamiques permettent aux entreprises d'améliorer et d'automatiser les tâches manuelles, d'obtenir une vue cohérente des relations avec les clients et partenaires, et d'orchestrer des processus métier conformes aux exigences internes et aux réglementations extérieures. Ces entreprises bénéficient ainsi d'une souplesse leur permettant d'être plus performantes sur le marché.

Dans l'esprit de ce qui précède, ce projet s'inscrit dans le cadre de développement de nouvelles applications tirant profit des nouvelles caractéristiques et avantages qu'offrent l'Architecture Orientée Service (SOA) et les Services Web. L'objectif de ce travail est la conception et le développement d'une application de publication des services d'intermédiation via des Services Web.

Dans ce rapport, nous commençons dans un premier chapitre par présenter une étude théorique aux grands concepts rencontrés lors de la réalisation du projet à savoir l'Architecture Orientée Service (SOA) et les services web. Le deuxième chapitre est consacré à une étude de l'existant et une spécification des besoins. Le troisième chapitre traite la conception détaillée du projet. Enfin, le chapitre réalisation va illustrer par des captures d'écran le travail réalisé ainsi qu'une description de l'environnement du travail. Tout au long de ce travail nous allons adopter la méthodologie orientée objet et le formalisme UML (Unified Modeling Language).

Chapitre 1

Concepts de base

Introduction

Les architectures orientées services (SOA) vont s'imposer au cours des prochaines années. Elles trouveront particulièrement leur place dans les contextes d'applications d'entreprise, imposant peut-être une refonte majeure des architectures déjà en place.

Selon le Gartner Group, plus de 75% des projets d'entreprise des années 2008 reposeront sur les SOA (Service Oriented Architecture) [1]. À elle seule, cette perspective suffit à faire prendre conscience de l'ampleur du phénomène, même s'il convient de mesurer les implications concrètes de cette approche sur les systèmes actuels.

En effet, il est important de pouvoir estimer si cette mouvance se traduira par une évolution de l'existant, ou si elle engendrera plutôt une refonte majeure des architectures déjà en place. Pour tenter d'apporter une réponse à cette question, voyons donc plus en détail ce que comprend l'appellation dans une première partie de ce chapitre.

En termes de technologies, les SOA ne sont pas étroitement liées à un éditeur ou une mouvance particulière. Il est ainsi possible d'envisager l'implémentation de projets respectant les caractéristiques des SOA, aussi bien sous une technologie Microsoft qu'avec des solutions Open source, J2EE (Java 2 Enterprise Edition) ou non.

En fait, les acteurs les plus présents dans le domaine des SOA sont ceux qui se sont positionnés parmi les premiers autour des services web, et qui proposent des solutions matures à ce niveau. En effet, même s'il est possible d'envisager la mise en place d'une architecture orientée services sans utiliser les services web, force est de constater que ces derniers sont particulièrement adaptés à ce type d'architecture, et qu'ils ont fortement contribué à son avènement. Donc une étude des web services paraît nécessaire dans une deuxième partie de ce chapitre.

1.1. Architecture Orientée Service (SOA)

1.1.1. Définition

L'Architecture Orientée Service (SOA) est une approche structurale qui permet d'intégrer les fonctions qui incluent des applications d'entreprise comme des services interopérables. Ces services sont basés sur des standards, ils peuvent simplement être réutilisés pour développer des nouveaux services ou traiter des nouvelles exigences venantes des clients [2].

SOA encourage les entreprises à penser à réutiliser leurs business services actuels, des données associées, des logiciels et des autres ressources, ce modèle libre des entreprises à commencer d'un niveau de définition business plus haut sur des données, des interfaces, des processus, etc. SOA dispose les services sur une nouvelle ou une ancienne infrastructure, elle nous permet d'ignorer le détail de chaque service, on n'a pas besoin de savoir qu'un service est déployé dans quel genre de système, est écrit en quel langage ou est accédé par lesquels protocoles.

1.1.2. Services

Le noyau de la conception de SOA est le service. SOA réalise l'intégration des différents services basés sur des standards ouverts. Un service représente une action de monde réel, il peut résoudre un problème tout seul ou avec des autres services intégrés sur la même infrastructure. L'infrastructure de SOA intègre des différents services, des composants et des processus business. Un composant sur l'infrastructure fonctionne sous la forme de service. Chaque processus business est composé par un ensemble de service, et le processus lui-même est aussi un service dans l'infrastructure.

1.1.2.1. Couplage faible de service

Tous les services concernés dans SOA sont couplage faible, autrement dit, tous les services sont indépendants, ils se connaissent par leurs propres descriptions qui contiennent un ensemble d'informations comme le nom de service, les paramètres demandés par ce service, le résultat retourné d'un service, etc. La description et l'implémentation d'un service sont aussi indépendantes. Il n'y a pas une liaison de communication explicite entre deux services. Le couplage faible de service est réalisé par les communications entre des services, il désigne une communication asynchrone. Par exemple, si un service A connaît la description de service B, A peut envoyer un message à B. Le service B reçoit ce message venant de A, mais peut-être il ne répond pas à A, plus précisément, cette communication n'a aucune liaison directe vers le service B. Chaque communication de service est réalisée par la transmission de message [2].

1.1.2.2. Transmission de messages

La transmission de message est la technique de base dans SOA, elle donne aussi une base à réaliser les communications entre des services, elle n'est pas une technique nouvelle, en fait, elle est déjà utilisée par plusieurs middlewares depuis longtemps. Mais SOA utilise une technique particulière de transmission de message. Après l'émission d'un message, le service n'a plus le droit de le contrôler, il ne sait pas à quel service ce message va être envoyé ou via laquelle façon. Par cette façon, SOA réalise le couplage faible de service, car toutes les communications sont indépendantes.

Nous avons dit que le couplage faible de service est réalisé par le mécanisme de transmission de message dans SOA, différentes façons sont fournies pour transmettre des messages entre des services, soit par une façon synchrone, soit par une façon asynchrone.

1.1.3. Conclusion

SOA ne possède pas une définition rigoureuse, on peut dire qu'elle est un nouveau modèle pour l'intégration de service, elle est une méthode abstraite. Cette nouvelle conception peut aider les grandes entreprises à changer ou améliorer leurs infrastructures simplement et dynamiquement, elle propose une façon qui permet de réutiliser des ressources, ça peut aider les grandes entreprises à réduire leur investissement sur leurs structures d'intégration.

Les services dans SOA sont basés sur les services web. Tous les services intégrés sont couplage faible, leurs propres descriptions sont décrites par WSDL (Web Service Description Language), les transmissions de message entre des services sont normalisées par SOAP (Simple Object Access Protocol).

SOA possède en général des caractéristiques comme ci-dessous :

- SOA souligne la réutilisation et l'encapsulation des services d'application ;
- Couplage faible entre les services ;
- SOA est basé sur des standards ouverts
- SOA propose une infrastructure flexible, le détail d'implémentation d'un service, la localisation d'un service, ou bien les protocoles d'accès d'un service doivent être transparents. Nous pouvons dynamiquement ajouter, supprimer ou changer un service pour traiter des nouvelles exigences.

1.1.4. Entreprise Service Bus (ESB)

Nous avons une idée générale sur SOA. Imaginons que nous avons déjà créé nos propres services métiers d'après la conception de SOA, et il y a des gens comme nous qui ont aussi

créé leurs services, on publie tous les services, et puis on essaye de réaliser l'interopérabilité de nos différents services. Il semble qu'on réalise l'idée de SOA, mais est-ce que ce modèle qu'on a construit peut nous apporter quelques profits ? Peut-être par ce modèle, on peut arriver d'appeler un composant de .NET dans l'environnement J2EE, mais on peut aussi le réaliser sans ce modèle. Si les deux noeuds qui ont envie de se communiquer se connaissent bien et s'admettent la façon de communication entre eux, même s'il n'existe pas une infrastructure de service commune, on peut encore réaliser l'interconnexion point à point. Nous devons donc avouer si nous n'avons que des services, et les services ne peuvent que se communiquer avec des liaisons explicites entre eux, ce n'est pas une infrastructure typique de SOA. Comme l'affichage de figure 1.1, il y a des liaisons explicites entre chaque deux services, évidemment, cette structure n'a pas bien respecté les caractéristiques de SOA, elle ne peut donc pas implémenter la conception de SOA.

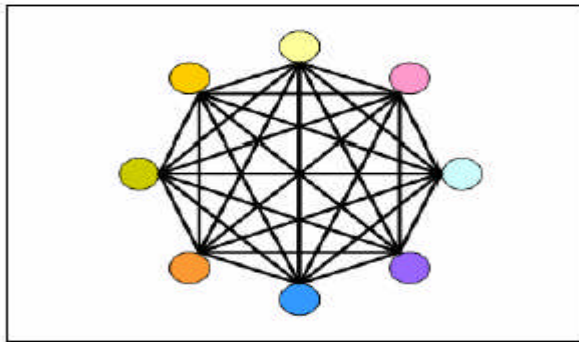


Figure 1.1. Liaison explicite entre les services

Nous regardons une autre architecture affichée dans la figure 1.2. Cette structure utilise un intermédiaire entre les services qui s'occupe de la transmission de message entre tous les services, elle semble une bonne architecture qui peut supporter SOA, car elle n'a plus de problèmes sur la transparence de service, mais cette structure nous apporte encore des problèmes. Toutes les communications de service doivent être transmises par cet unique intermédiaire, la charge de ce middleware va être très lourde ; si cet intermédiaire ne fonctionne plus, tout ce système va être en panne.

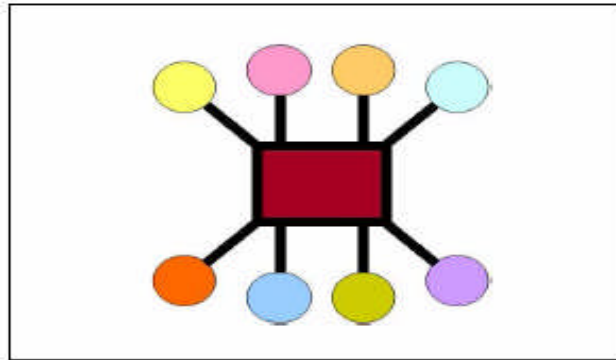


Figure 1.2. Liaison des services à travers un middleware

Heureusement, l'apparition d'une nouvelle technique ESB (Enterprise Service Bus) nous donne une bonne solution pour implémenter SOA.

1.1.4.1 Qu'est ce que Entreprise Service Bus (ESB)

ESB est un nouveau modèle basé sur le message, il respecte les caractéristiques de SOA et propose une infrastructure utilisable pour implémenter la conception de SOA, il permet d'intégrer des services qui peuvent être déployés dans des systèmes distribués ou des environnements hétérogènes [2].

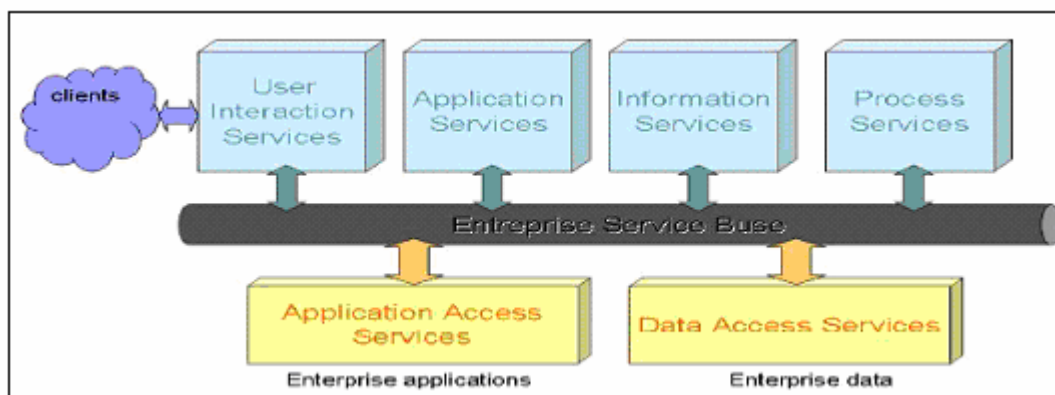


Figure 1.3. Architecture générale d'un ESB

La figure ci-dessus (figure 1.3) nous montre une architecture générale d'ESB, par rapport à l'architecture affichée dans la figure 2, cette nouvelle structure peut nous apporter lesquels avantage ?

D'abord, elle est plus ouverte que l'architecture en dessus qui ne possède qu'un intermédiaire, la structure de bus peut être infiniment étendue. Ensuite, elle n'utilise pas un seul middleware pour transmettre des messages de services, tous les middlewares sont aussi intégrés sous une forme de service sur le bus.

1.1.4.2. Caractéristiques et capacités d'ESB

L'ESB propose une méthode standard et fournit une couche partagée pour réaliser l'interopérabilité, l'interaction et l'intégration des services distribués. Les services (peut être un composant, une application business, un processus business, etc.) sont basés sur Web services, chacun a sa propre description décrite par WSDL, ils se connaissent par leurs description, puis se communiquent via le bus en façon transparente. Le bus s'occupe de la transmission de message entre des services par des différentes façons, soit par message ('call' ou 'one-way'), soit par événement (pub./sub.). Nous pouvons dynamiquement et librement changer, supprimer un service, ou ajouter un nouveau service. Un ESB doit posséder au minimum des capacités comme la transformation de message, le routage de message, la capacité de découvrir des fautes, l'assurance de la fiabilité et la sécurité.

1.1.4.2.1. Transformation de message

Les services intégrés sur ESB sont hétérogènes, si le demandeur d'un service et le fournisseur de ce service sont basés sur des protocoles différents ou décrits en différents langages, le bus doit d'abord transformer des données vers des données du format commun[2]. Ça peut être réalisé par exemple en utilisant un middleware associé, un protocole switch (Figure 1.4).

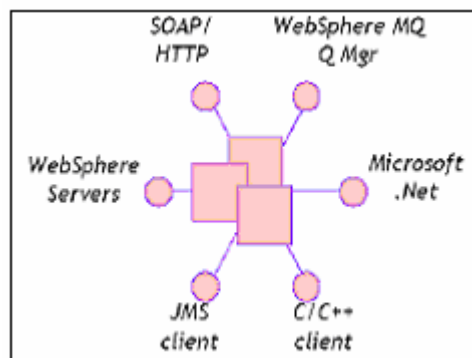


Figure 1.4. Protocole Switch

1.1.4.2.2. Routage de message

Un service communique avec un autre service d'après sa propre description, mais ce service ne connaît pas vraiment le service destinataire, il n'existe pas une liaison directe entre des services, c'est l'ESB qui doit réaliser la communication entre deux services. Quand le bus

reçoit un message, en utilisant quelques mécanismes de routage, il route intelligemment ce message à un bon service destinataire qui peut traiter l'exigence de ce message [2].

Nous donnons ici quelques mécanismes de routage utilisés actuellement dans ESB :

- Routage par le contenu basé sur XML (eXtensible Markup Language) ;
- Routage externe : le routage de message est configuré extérieurement, il est complètement déconnecté avec le contenu de message.

1.1.4.2.3. Capacité de découvrir des fautes

Un ESB doit fournir une capacité de découverte et de recouvrement des fautes sur la transmission de message. Le bus peut être associé avec un middleware qui joue un rôle comme un moniteur, ce moniteur observe tous les transmission de message entre des services, il peut découvrir tout de suite des fautes comme l'échec d'une transmission, l'émission dupliquée d'un message, la faute de réseau, etc.

1.1.4.2.4. Fiabilité et sécurité

Pour chaque message envoyé par des services, l'ESB doit pouvoir assurer la sécurité et la fiabilité en utilisant des mécanismes comme l'authentification, l'autorisation, la confidentialité, etc.

1.1.4.3. Conclusion

Comme nouveau modèle qui peut implémenter la conception de SOA, ESB fournit une couche partagée pour intégrer des services hétérogènes et réaliser l'interaction et l'interopérabilité entre ces services, ESB utilise des interfaces et des formats standards, elle supporte un ensemble de divers protocoles, et tous les services intégrés sont aussi basés sur des standards et sont couplage faible. ESB propose aux entreprises une infrastructure d'intégration assez flexible qui peut les aider à développer la capacité pour traiter des exigences variées des clients et de résoudre des problèmes arrivant soudainement. ESB souligne la réutilisation des services (application, composant ou ressource). En utilisant ESB, les grandes entreprises peuvent réduire leurs investissements sur l'infrastructure d'intégration, et en même temps, élever l'efficacité pour résoudre un problème.

1.2. Les services Web

Il existe beaucoup de méthodes, dans la littérature, pour implémenter un dialogue entre deux applications distantes. La méthode qui nous vient directement à l'esprit c'est les Sockets. Mais, cette méthode exige une implémentation très coûteuse (en temps). En effet, le

débogage devient de plus en plus difficile chaque fois que la complexité du dialogue augmente. Ajoutons à cela les problèmes de portabilité du code, etc. Pour nous faciliter la tâche nous allons implémenter l'architecture distribuée client serveur comme service Web.

Ce paragraphe va introduire la notion de Service Web.

1.2.1. Définition

Un service Web [3] est un module logiciel effectuant une tâche discrète ou un ensemble de tâches, qui peut être localisé et appelé au travers d'un réseau, en particulier le World Wide Web. En effet, une application cliente peut appeler une série de services Web par l'intermédiaire d'appels de procédures distantes ou d'un service de messagerie pour fournir une partie de la logique de l'application (figure 1.5). En d'autres termes, si une application peut être accessible sur le réseau en utilisant une combinaison de protocoles comme HTTP (Hypertext Transfer Protocol), XML, SMTP (Simple Mail Transfer Protocol), ou Jabber, alors c'est un service Web.

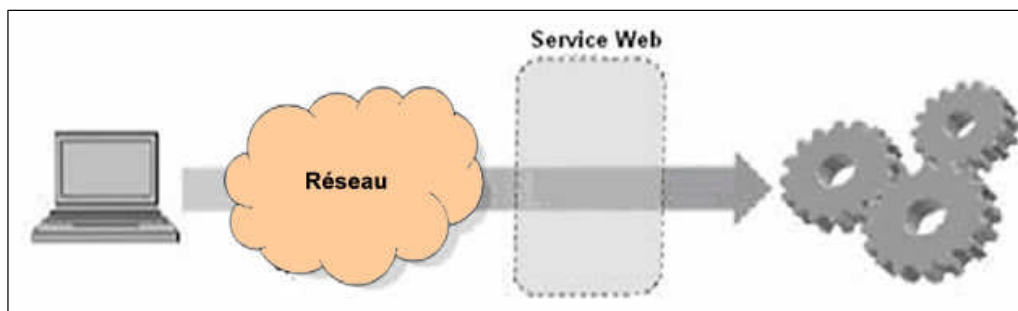


Figure 1.5: Un Service Web permet l'accès au code de l'application en utilisant les standards de la technologie Internet

1.2.2. Principe fondamental du service Web

Un Service Web est une interface positionnée entre le code de l'application et l'utilisateur de ce code. Elle agit comme couche d'abstraction, qui sépare les détails spécifiques à un langage ou plate-forme donnée, de la façon dont le code sera invoqué. Cette couche de standardisation signifie que n'importe quel langage qui supporte le service Web peut accéder aux fonctionnalités de l'application (figure 1.6).

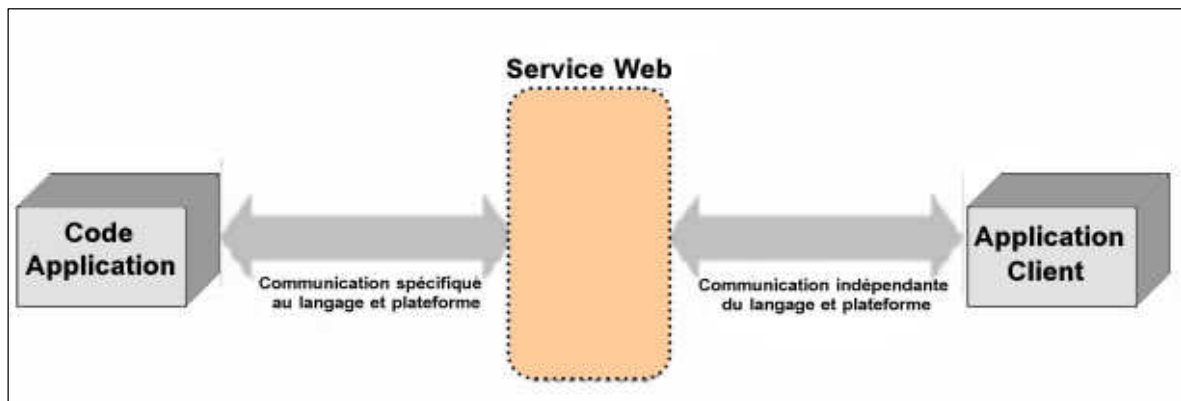


Figure 1.6 : Le service Web offre un couche d'abstraction entre l'application client et le code de l'application

Les services Web que nous voyons aujourd'hui sur Internet sont les sites Web HTML. Dans ces services –les mécanismes de publication, de gestion, de recherche– sont accessible via des protocoles standard comme HTTP et HTML. Les applications client (navigateur Web) qui comprennent ces standards peuvent interagir avec ces services.

À cause de cette couche d'abstraction offerte, il n'est plus problème que les applications clients sont écrites en Java et les clients en C++, ou bien les applications sont déployés sur Unix alors que les clients tournent sur le modeste Windows. Les services Web permettent l'interopérabilité cross plateforme. L'interopérabilité est un gain important lors de l'implémentation des services web.

1.2.3. La couche de technologie dans un service Web

L'architecture d'un service Web est implémentée à travers cinq technologies mises en couches comme le décrit la figure 1.7.

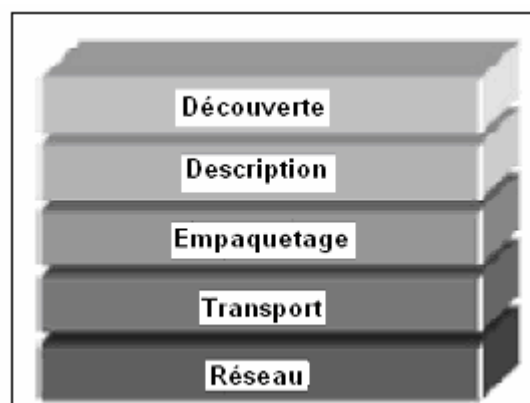


Figure 1.7: Pile de technologies dans les services Web.

1.2.3.1. Découverte

La couche découverte offre aux consommateurs un mécanisme pour dégager les descriptions des fournisseurs. Un des mécanismes de découverte largement reconnus est Universal Description, Discovery and Intergration (UDDI).

1.2.3.2. UDDI (Universal Description, Discovery et Integration)

Il s'agit d'une spécification de registre distribué d'informations sur des services Web [4]. Lorsqu'un service Web a été développé et qu'un document le décrivant a été créé, il doit y avoir un moyen de mettre les informations à la disposition des utilisateurs désirant utiliser le service Web décrit. La publication d'un service Web dans un registre UDDI permet aux utilisateurs de rechercher et d'apprendre l'existence du service Web. Le contenu d'un registre UDDI est similaire à celui d'un répertoire téléphonique.

Dans la section des "pages blanches" du registre, on trouve des informations comme le nom, l'adresse et le numéro de téléphone du business qui propose un ou plusieurs services Web.

La section des "pages jaunes" identifie le type de business et la classe par industrie. La section des "pages vertes" fournit des informations sur les services Web proposés par le business.

1.2.3.3. Description

Lorsqu'un service Web est implémenté, il doit faire des décisions à chaque niveau inférieur qu'il va supporter. Une description représente ces décisions d'une manière à permettre au consommateur de contacter et utiliser le service. Le langage WSDL (Web Service Description Language) est en fait le standard qui offre ces descriptions. Il existe aussi d'autres approches moins connues comme W3C Ressource Description Framework (RDF) et DARPA Agent Markup Language (DAML).

1.2.3.4. WSDL (Web Services Description Language)

Un service Web n'est utilisable que si d'autres personnes peuvent découvrir ce qu'il fait et comment l'appeler. De plus, les développeurs doivent posséder suffisamment d'informations sur un service Web pour pouvoir écrire un programme client qui l'appelle. WSDL [4] est un langage basé sur XML qui permet de définir des services Web et de décrire la manière d'y accéder. Il décrit en particulier les contrats de données et de messages d'un service Web. En examinant le document WSDL d'un service Web, les développeurs connaissent les méthodes disponibles et savent comment les appeler en utilisant les paramètres adéquats.

1.2.3.5. Empaquetage

Pour que les données puissent être transportées par la couche réseau, elles doivent être empaquetées dans un format que toutes les parties peuvent comprendre (le processus s'appelle « Serialisation » et « Marshalling »). HTML est un format d'empaquetage, mais il peut être inconvenable pour travailler avec puisque HTML est plus lié à la présentation de l'information qu'à sa signification. Actuellement, XML est la base de tout format d'empaquetage pour les services web, puisque XML peut être utilisé pour représenter la signification des données transférées, et parce que les parseurs XML sont maintenant omniprésents dans tout framework de développement. SOAP est aussi un format d'empaquetage, basé sur XML, qu'on va détailler l'usage plus tard dans ce chapitre. Désormais, il existe beaucoup de standards d'empaquetage dans la littérature, comme XML-RPC (XML Remote Procedure Call), mais pour des raisons qu'on va expliciter après, on a choisi le standard SOAP [5].

1.2.3.6. SOAP (Simple Object Access Protocol)

Pour échanger des messages entre un service Web et une application appelante, on utilise les messages SOAP. Ces messages sont des documents XML transportés via le protocole http. En réalité, il s'agit d'un important composant de base pour développer des applications distribuées. Les services Web peuvent être écrits dans n'importe quel langage et exécutés sur n'importe quelle plate-forme. Un client d'un service Web peut également être écrit dans n'importe quel langage et exécuté sur n'importe quelle plate-forme (voir Annexe A).

1.2.3.7. Transport

La couche de transport inclue les différentes technologies qui permettent une communication application à application. Parmi ces technologies on peut citer TCP, HTTP, SMTP, et Jabber. Le rôle primaire de la couche transport étant de transporter les données entre deux emplacements du réseau. Les services Web peuvent être basés sur tout protocole de transport. Le choix du protocole de transport est basé largement sur les besoins de communication du service Web implémenté.

1.2.3.8. Réseau

La couche réseau de la pile de technologies d'un service Web est la même que celle dans la pile TCP/IP (Transmission Control Protocol /Internet Protocol). Elle offre l'adressage, le routage, etc.

1.2.3.9. Application

La couche application est le code qui implémente la fonctionnalité du service web, (c'est à dire le Business Logic), qui est accessible via les couches inférieures de la pile.

1.2.4. Architecture des services Web

L'architecture des services Web a trois rôles distincts :

- Le fournisseur crée le service Web et le met à la disposition des clients qui souhaitent l'utiliser.
- Un demandeur est une application cliente qui utilise le service Web. Le service Web demandé peut également être client d'autres services Web.
- L'agent permet au fournisseur et au demandeur d'un service Web de communiquer ensemble. Il peut être par exemple un registre de service.

Les trois rôles de fournisseur, de demandeur et d'agent interagissent les uns avec les autres par l'intermédiaire des opérations de publication, de recherche et de liaison. Un fournisseur informe l'agent de l'existence du service Web en utilisant l'interface de publication de cet agent pour permettre aux clients d'accéder au service. Les informations publiées décrivent le service et spécifient son emplacement. Le demandeur consulte l'agent pour localiser un service Web publié. Grâce aux informations sur le service Web obtenues par l'agent, le demandeur peut lier ou appeler le service Web. Le diagramme suivant résume la manière dont le fournisseur, le demandeur et l'agent interagissent les uns avec les autres (figure 1.8).

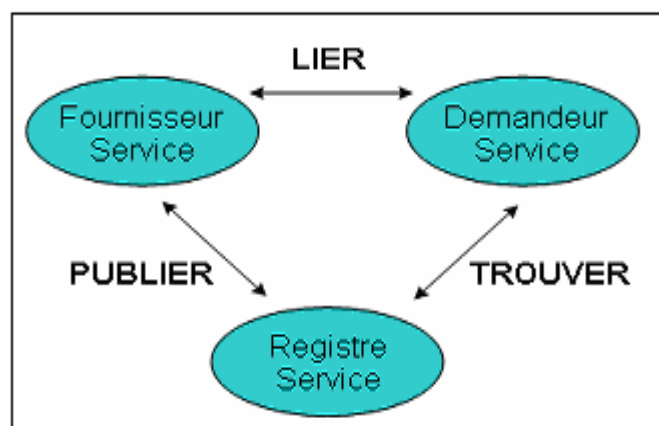


Figure 1.8: Rôles et opérations pour les Services Web

Conclusion

Dans ce chapitre, nous avons présenté deux grands concepts : L'Architecture Orientée Service (SOA) et les Services Web en découvrant leurs caractéristiques, leurs intérêts et leurs modes d'emploi qui seront utiles par la suite pour la conception et le développement de notre application. Ainsi le lecteur ne trouvera pas d'ambiguïté pour comprendre le choix de ces solutions pour répondre à nos besoins décrits dans le chapitre suivant qui est consacré à l'étude de l'existant et la description des besoins fonctionnels et non fonctionnels de notre application.

Chapitre 2

Etude de l'existant et spécification des besoins

Introduction

Après avoir donné une idée sur les concepts utiles pour la réalisation de notre projet, et pour pouvoir proposer une solution compétitive, il faudra explorer les différentes technologies existantes traitant cette problématique. Le chapitre suivant présentera en premier temps une étude de l'existant et par la suite une spécification détaillée des besoins fonctionnels et non fonctionnels de notre application.

2.1. Etude et critique de l'existant

Notre étude pour l'existant s'est basée sur les solutions actuelles qu'offrent les entreprises. Généralement, ces solutions sont monoprocolaires, c'est-à-dire qu'elles supportent un seul type de protocole. En plus, elles ne représentent pas des structures standards basées par exemple sur XML, mais au contraire, la plupart des structures sont propriétaires. Il est à noter aussi que ces applications ne peuvent pas opérer avec des clients hétérogènes, on a besoin d'un développement spécifique pour chaque type de clients. Enfin, ces solutions manquent de fiabilité puisqu'on a un grand risque pour les pertes des messages et ne répondent pas aux exigences des nouvelles technologies.

Donc l'entreprise n'est généralement pas confrontée à un problème de fonctionnalités insuffisantes, mais plutôt au fait que des systèmes très importants pour l'entreprise, tels que des logiciels de gestion de la relation client et des progiciels de gestion intégrés fonctionnent sur des systèmes différents. L'intégration des informations et des processus est donc nécessaire. Jusqu'à présent, cela nécessitait des interventions manuelles laborieuses ou le développement de solutions spécifiques, rigides et difficiles à gérer.

L'architecture orientée service est une méthode de conception basée sur des standards permettant de créer une infrastructure informatique intégrée capable de répondre rapidement aux nouveaux besoins d'une entreprise et résoudre les différents problèmes cités au début de

ce paragraphe. Elle fournit les principes et les directives permettant de transformer un réseau existant de ressources informatiques hétérogènes, distribuées, complexes et rigides en ressources intégrées, simplifiées et particulièrement souples pouvant être modifiées et combinées afin de mieux satisfaire les objectifs de l'entreprise.

2.2. Spécification des besoins

La finalité de ce projet est de mettre en place une solution pour les services d'intermédiation documentaire (pour notre cas le service TCE: Titre de Commerce Extérieur) basée sur l' Architecture Orientée Service (logique SOA), dont l'objectif est de permettre aux utilisateurs ou plus exactement aux opérateurs la domiciliation d'un titre de commerce extérieur.

Pour mettre en place le service TCE on a recouru à une solution basée sur la logique SOA qui répond bien aux problèmes cités dans le paragraphe précédent et donc à nos besoins cités ci-dessus:

- Invoquer des services via des messages standards et en format unique: ce sont des messages XML.
- Gérer la réception des messages.
- Recherche du service correspondant à une invocation bien déterminée.
- Retour du résultat du service à invoquer.
- Le mapping ou le séquençement des messages échangés.
- La gestion des erreurs de duplication des messages échangés.
- La gestion des acquittements.

2.2.1. Les besoins fonctionnels

Pour aboutir à la réalisation de notre plateforme SOA et l'implémentation du service TCE, l'application doit répondre à un ensemble des besoins fonctionnels tels que:

- La généralité: notre plateforme basée sur la logique SOA permettra d'implémenter les différents services de la même manière.
- Le service TCE (Titre de Commerce Extérieur): C'est le service à implémenter pour notre architecture SOA.
- L'authentification: l'utilisateur doit s'authentifier avant d'utiliser les services offerts par l'application.

- L'administration des utilisateurs: l'application permettra d'administrer les utilisateurs tel que:
 - Créer un opérateur.
 - Supprimer un opérateur.
 - Modifier un opérateur.

2.3. Les Acteurs

Pour n'importe quel service d'intermédiation documentaire ce sont les systèmes d'informations. Et pour notre service TCE on a les trois acteurs suivants:

- L'opérateur:c'est un opérateur transitaire qui demande la domiciliation d'un titre de commerce extérieur.
- La banque de domiciliation.
- La banque centrale.

On a aussi l'administrateur de notre plateforme ou notre service et l'agent de paramétrage qui est responsable du déploiement d'un nouveau service d'intermédiation documentaire autre que le service TCE sur notre plateforme SOA.

2.4. Diagramme des cas d'utilisation

2.4.1. Diagramme des cas d'utilisation général

Le diagramme principal des cas d'utilisations permet de traduire les spécifications fonctionnelles d'utilisation du système. C'est-à-dire les différentes possibilités d'utilisation de notre plateforme vis-à-vis des différents sous systèmes d'information. Ce diagramme est représenté par la figure suivante (figure 2.1):

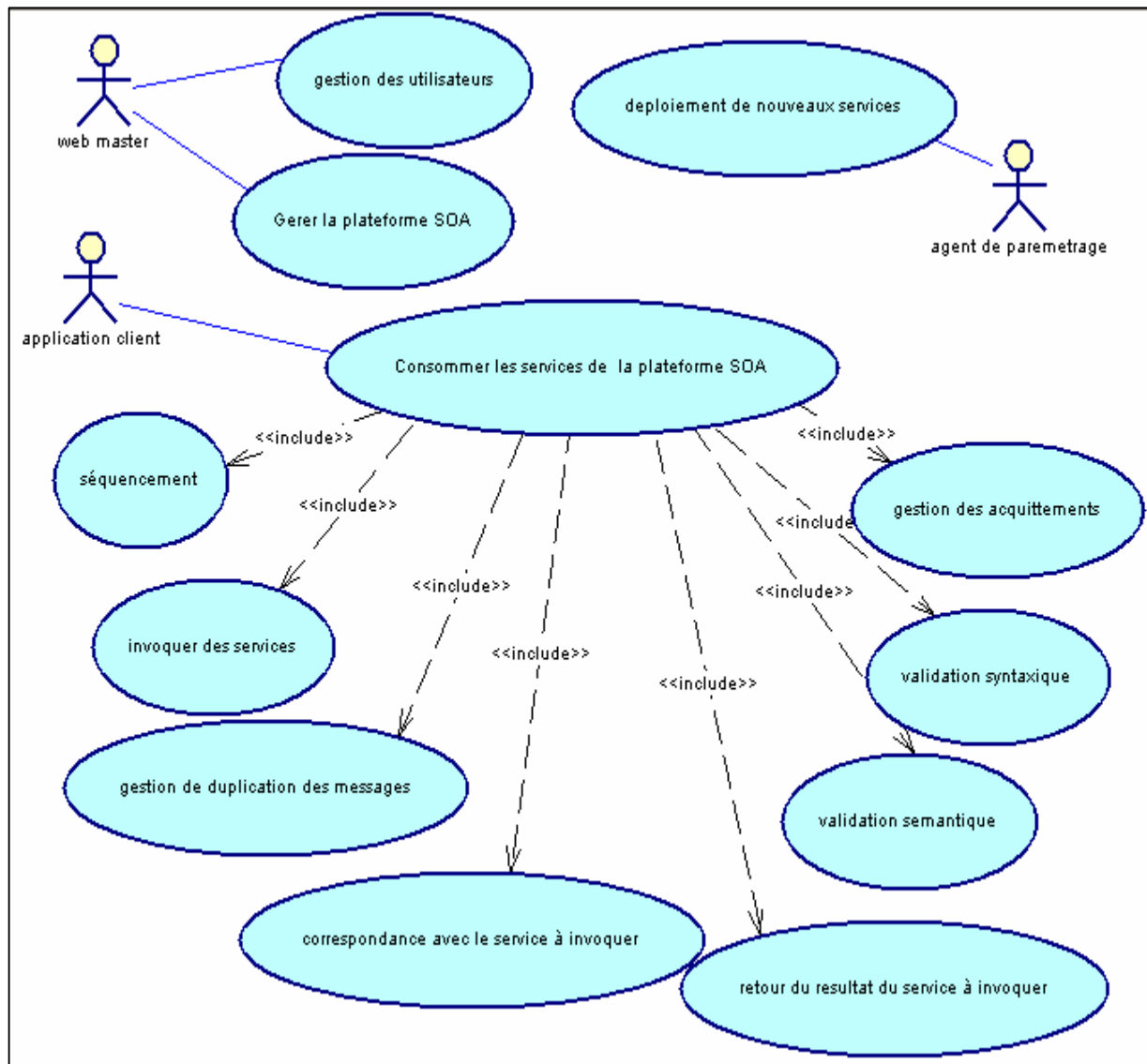


Figure 2.1. Diagramme de cas d'utilisation général

Ce diagramme de cas d'utilisation représente les différentes spécifications qui vont servir au bon fonctionnement de notre plateforme, cités ci-dessous :

- ✓ Déploiement de nouveaux services
- ✓ Gestion des utilisateurs
- ✓ Validation syntaxique des messages XML reçus
- ✓ Validation sémantique des messages XML reçus
- ✓ Assurer le bon séquencement des messages XML
- ✓ Gestion des duplications de messages
- ✓ Assurer la correspondance avec le service à invoquer
- ✓ Invoquer les services

- ✓ Retour du résultat du service invoqué
- ✓ Gestion des acquittements.

2.4.2. Diagramme des cas d'utilisation « Administration »

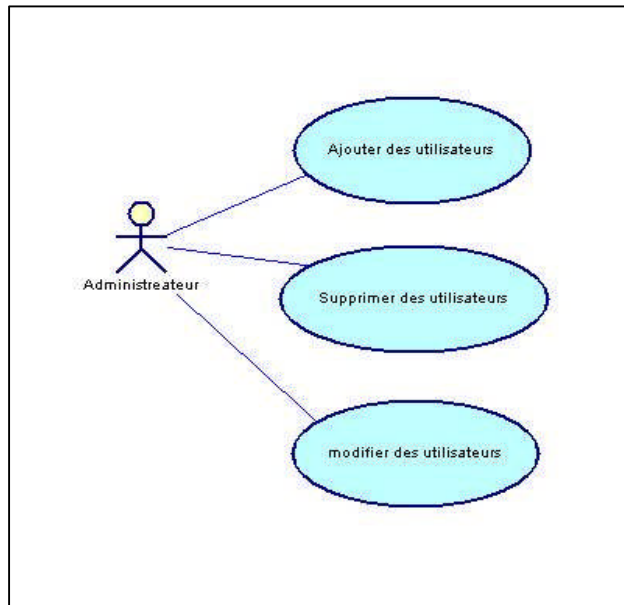


Figure 2.2. Diagramme des cas d'utilisation « Administration »

Cette spécification permet à l'administrateur d'ajouter, supprimer ou modifier des utilisateurs au niveau de notre plateforme en intervenant sur les champs non d'utilisateur et mot de passe.

2.4.3. Diagramme des cas d'utilisation « Déploiement de nouveaux services »

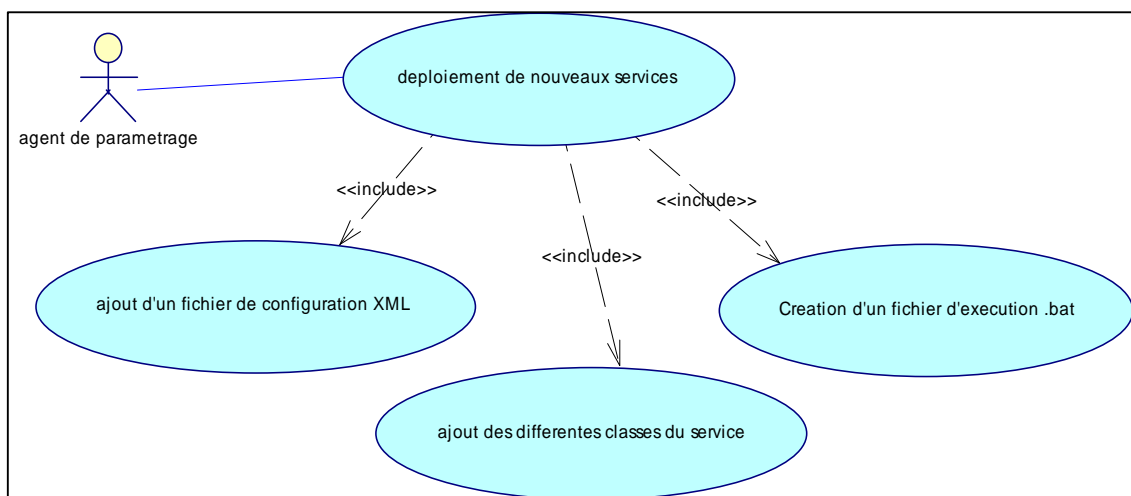


Figure 2.3. Diagramme des cas d'utilisation « Déploiement de nouveaux services »

Ce diagramme de cas d'utilisation bien détaillé explique comment un agent de paramétrage peut ajouter un nouveau service au niveau de notre plateforme SOA. Notre plateforme est générique donc la méthode de déploiement est unique pour nos différents services d'intermédiation documentaire (on prendra un exemple de service à implémenter dans le paragraphe suivant, le service TCE : Titre de Commerce Extérieur). L'agent de paramétrage doit ajouter les différentes classes du service à déployer, ensuite ajouter le fichier de configuration XML et générer le fichier d'exécution (.bat).

2.4.4. Diagramme des cas d'utilisation pour le cas du service TCE

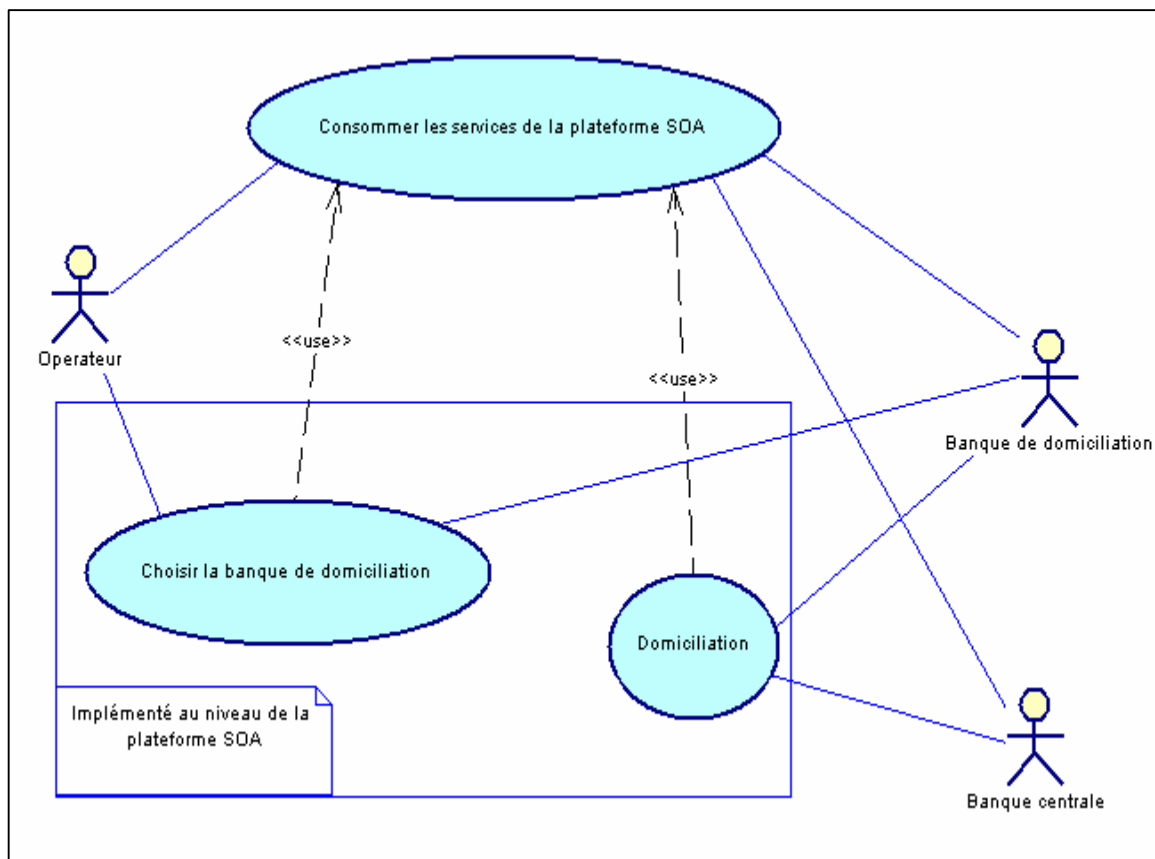


Figure 2.4. Diagramme des cas d'utilisation du service TCE

Ce diagramme de cas d'utilisation montre l'exemple d'un service d'intermédiation documentaire (service TCE) à ajouter au niveau de notre plateforme SOA. Ce service bénéficie des fonctionnalités offertes par notre plateforme SOA citées dans le paragraphe 4.1. Pour ce cas, la correspondance avec le service à invoquer correspond au choix de la banque de

la domiciliation et le retour du résultat correspond à la domiciliation du titre du commerce extérieur qui seront bien détaillés au niveau du paragraphe suivant.

2.4.4.1. Diagramme des cas d'utilisation « Domiciliation »

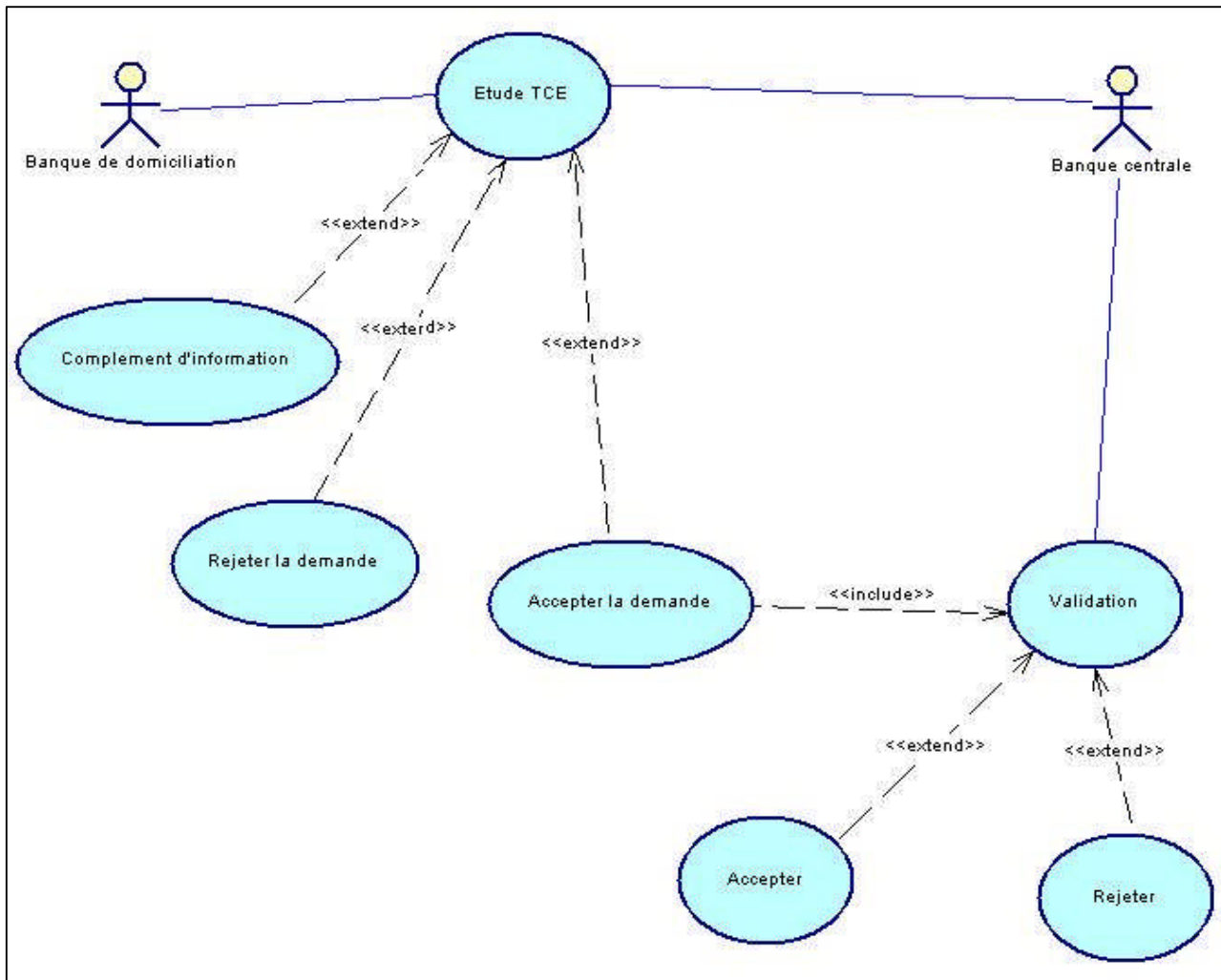


Figure 2.5. Diagramme des cas d'utilisation « Domiciliation »

La banque de domiciliation choisie par l'utilisateur reçoit la demande, elle traite alors la requête et décide soit de rejeter la demande, soit de demander un complément d'information, soit d'accepter la demande et dans ce cas elle est obligée de l'envoyer à la banque centrale pour vérifier la validation .

2.5. Scénarios décrivant le fonctionnement global du service TCE

Un scénario qui décrit une exécution particulière d'un cas d'utilisation général du début à la fin peut expliquer le fonctionnement global de notre service ou n'importe quel autre service d'intermédiation documentaire implémenté suivant la logique SOA.

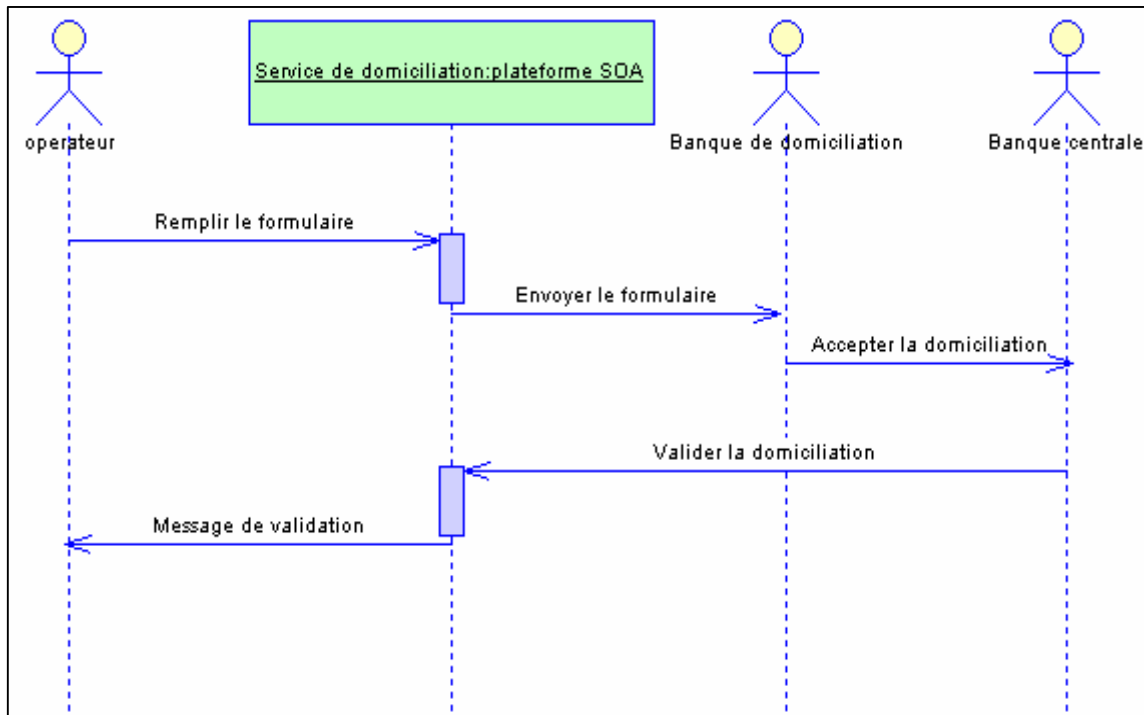


Figure 2.6. Diagramme de séquence décrivant le scénario : « dossier de domiciliation accepté ».

L'échange des différents messages entre les acteurs et les classes de notre service se fait en XML. C'est l'avantage primordial de notre architecture SOA qui permet ainsi d'intégrer différents systèmes d'information hétérogènes.

D'autres scénarios peuvent exister pour ce service, on cite par exemple le scénario dont la banque de domiciliation demande un complément d'information (figure 2.7)

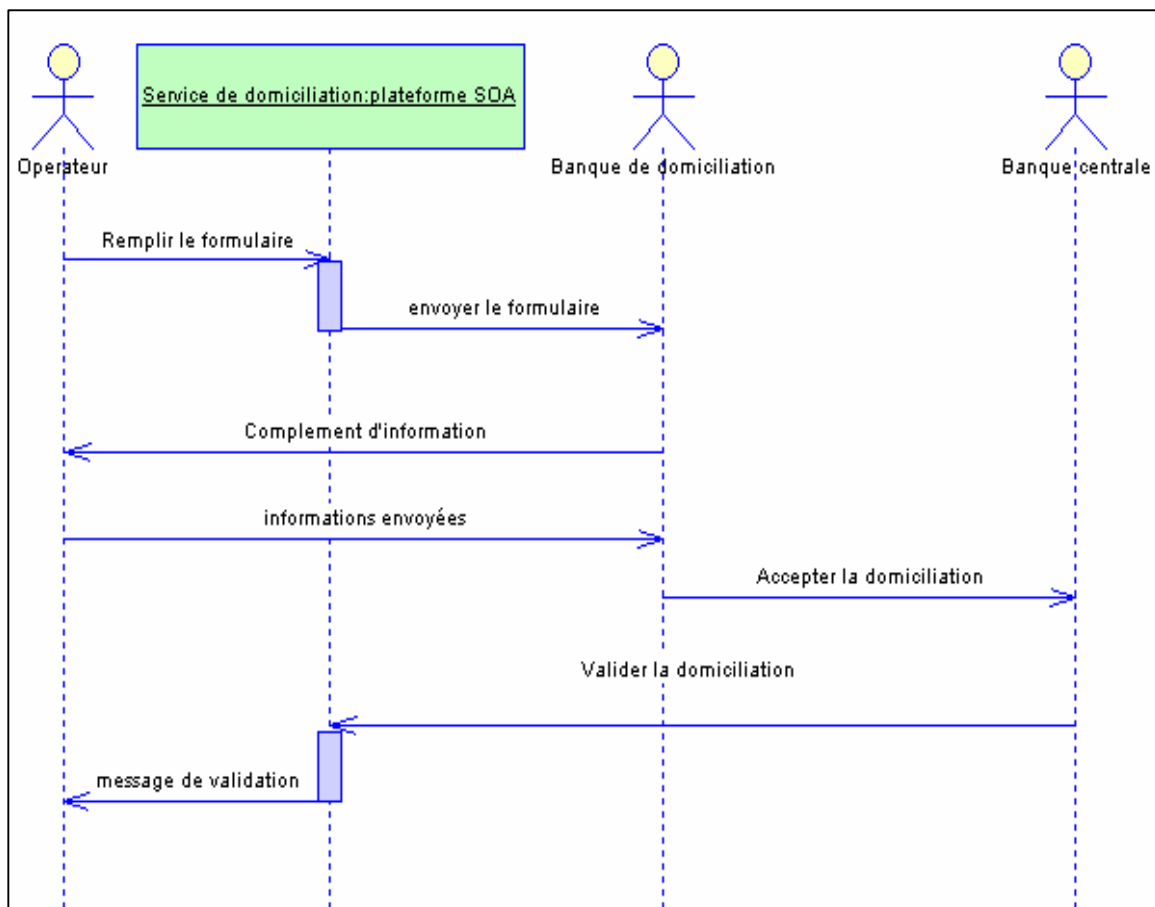


Figure 2.7. Diagramme de séquence décrivant le scénario : « demande de complément d'informations »

Conclusion

Dans ce chapitre, nous avons présenté en premier lieu une étude de l'existant pour le traitement des problèmes de gestion des ressources distribuées et d'intégration des services déployés dans des environnements différents. Dans un second lieu nous avons déterminé les besoins fonctionnels et non fonctionnels de notre plateforme SOA. En dernier lieu nous avons dégagé les cas d'utilisations correspondants. Le chapitre suivant présente les différentes phases de notre conception (conception générale et détaillée) ainsi que les différents diagrammes de séquences entre les différentes classes et objets de notre plateforme.

3.1.1. Services ESB

Ils sont illustrés au niveau de la partie gauche de la figure 3.1, il existe plusieurs types de services ESB. Les services ESB standards mettant en oeuvre la médiation sont les suivants : transformation XML, routage basé sur le contenu et médiation définie par l'utilisateur. Un service de transformation XML assure la conversion des données d'un document XML vers un autre conformément à une spécification XSLT (eXtensible Stylesheet Language Transformation). Un service de routage basé sur le contenu dirige les messages vers plusieurs autres extrémités de service conformément aux règles de routage configurées. Un service de médiation défini par l'utilisateur met en oeuvre une fonction telle que l'enregistrement ou la validation personnalisée, qui complète la logique métier de base. Un service ESB défini par l'utilisateur se connecte aux services d'entreprise (voir paragraphe 1.2) pour mettre en oeuvre cette logique de base.

Les services ESB s'exécutent dans le container ESB. Ce dernier instancie les services ESB et fournit l'interface de gestion de systèmes à ses services, la gestion des protocoles de communication, la fonctionnalité permettant de disposer de plusieurs threads exécutant des services à des fins d'évolutivité, et la fonctionnalité standard permettant d'exécuter des processus distribués.

3.1.2. Services d'entreprise connectés

Ils sont illustrés au niveau de la partie droite de la figure. Les services connectés mettent en oeuvre la logique métier des applications intégrées ESB. Il existe différents types de services d'entreprise : Web Services, Legacy Systems, JMS (Java Message Service) Applications et l'ensemble des ASA (Application Server Applications). Tous ces types peuvent communiquer les uns avec les autres via le bus ESB, et avec les services ESB à des fins de médiation.

3.1.3. Communications et processus ESB

Cette section de la figure 3.1 présente la fonctionnalité de base des services (ESB et d'entreprise) permettant de communiquer et d'exécuter un processus ESB, qui est une séquence d'étapes appelant des services ou d'autres processus ESB.

Un service ESB envoie et reçoit des messages via des références aux extrémités et adresses ESB.

Un processus ESB est défini par un itinéraire ESB contenant une séquence d'étapes de processus. Chaque étape peut appeler un ou plusieurs services ESB, d'autres processus ESB, ou un service Web externe. L'itinéraire ESB est effectué avec son message ESB associé en fournissant un environnement d'exécution entièrement distribué, et donc fiable, au processus ESB. Il n'existe pas de moteur de traitement centralisé pour l'itinéraire car chaque container ESB peut traiter et router l'étape suivante. Cela permet à la fois une exécution efficace, sans qu'il soit nécessaire de communiquer vers un site central, et une fiabilité optimale car le traitement se poursuit même en cas de perte de connectivité.

Les points d'entrée et de sortie de chaque service ESB peuvent être modifiés sans qu'il soit nécessaire de changer leurs mises en oeuvres.

La connexion ESB utilise une sémantique de livraison synchrone et asynchrone configurable par l'utilisateur, dont les modèles "publish-and-subscribe", "send-and forget" et "request-reply".

Lorsque les services requièrent des types de connexion différents, le bus ESB assure la médiation entre les différents protocoles.

La classe du message ESB et les associations d'envoi-reception entre l'extrémité et le message ESB créent la structure de base permettant à un service ESB d'envoyer un message vers un autre via un répartiteur. Ce répartiteur gère l'envoi et la réception des messages ESB entre les services et les extrémités externes. Il gère également les processus ESB.

Des définitions WSDL (Web Services Definition Language) standards peuvent être créés pour les services et processus ESB. Cela permet l'utilisation d'outils standards et l'intégration avec d'autres composants d'infrastructure SOA tels que le registre UDDI.

L'architecture de notre plateforme SOA qui a pour fonction l'implémentation des différents services d'intermédiation documentaire doit assurer au moins les fonctionnalités de base de l'architecture générale d'une solution SOA. Nous détaillerons dans les paragraphes suivants la conception détaillée de notre plateforme en packages et en diagrammes de classe pour chaque package.

3.2. Décomposition en packages

L'architecture de notre application ainsi que les diagrammes des cas d'utilisations nous a permis de décomposer notre système en quatre modules et l'utilisation d'un module prêt (Mule) comme c'est indiqué dans la figure 3.2 .

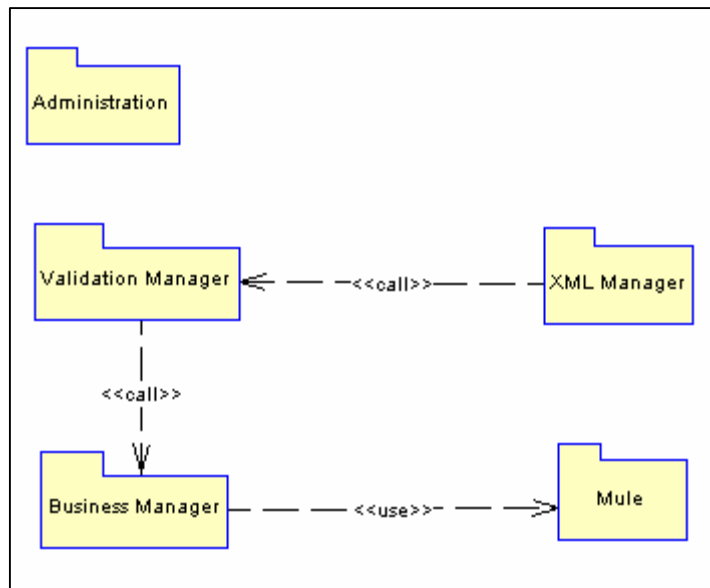


Figure 3.2. Diagramme des packages

- XML Manager : ce package assure la réception et l'envoi des messages XML avec l'utilisateur, ainsi que la vérification si ces messages sont dupliqués.
- Validation Manager : c'est un module permettant le traitement nécessaire pour la vérification syntaxique et sémantique du message XML reçu après son analyse.
- Business Manager : il englobe les traitements nécessaires pour la gestion des différents messages XML reçus, la correspondance avec le service à invoquer et la connexion avec ce service.
- Administration : ce package permet la gestion des différents services de la plateforme ainsi que les différents utilisateurs et de gérer l'authentification des utilisateurs.

L'utilisation du package Mule est présentée dans le chapitre suivant en détails.

3.2.1. Description des liens

Lors de la réception d'un message XML par le package XML Manager, ce dernier vérifie si ce message a été déjà reçu ou non et puis il appelle le package Validation Manager pour la vérification syntaxique et sémantique de ce message. Ensuite, le message est envoyé au module business pour invoquer le service correspond et retourner les résultats via les composants présents dans le module Mule.

3.3. Conception détaillée

3.3.1 Package « XML Manager »

Son diagramme de classe est donné par la figure 3.3.

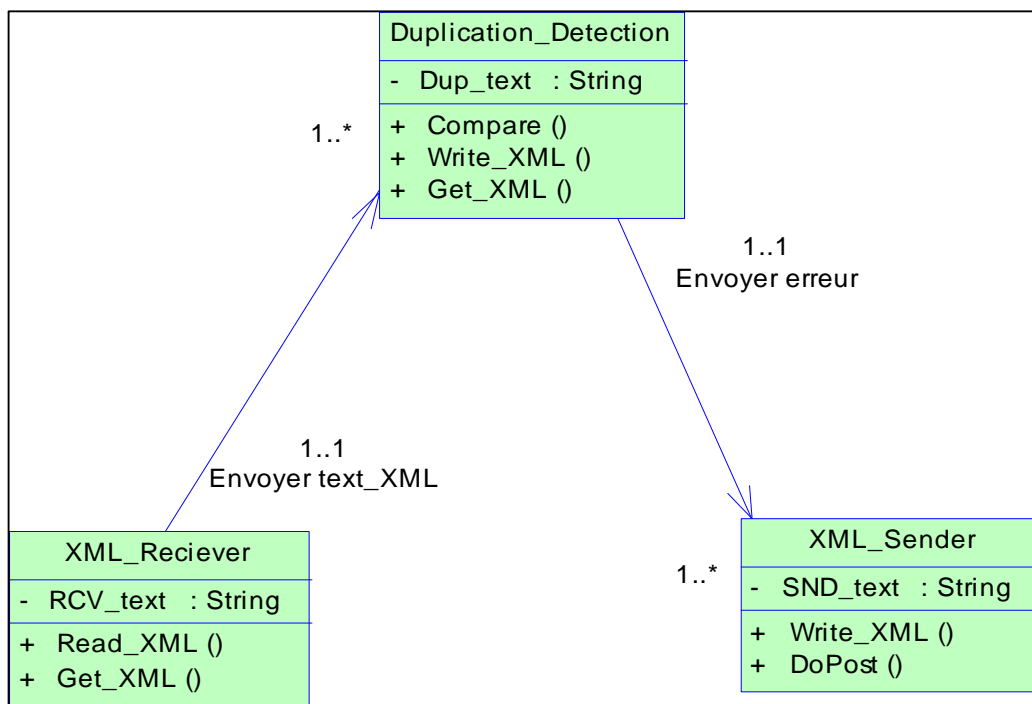


Figure 3.3. Diagramme de classe pour le package « XML Manager »

Ce diagramme de classe comprend :

- Deux classes pour l’envoi et la réception des messages XML avec l’utilisateur ou l’opérateur, elles implémentent des méthodes de lecture et d’écriture des messages XML ainsi que des méthodes pour la gestion de l’envoi et de la réception.
- Une classe qui implémente des méthodes pour la détection des messages dupliqués.

3.3.2. Package « Validation Manager »

C’est un package de contrôle pour les messages XML. Son diagramme de classe est donné par la figure 3.4.

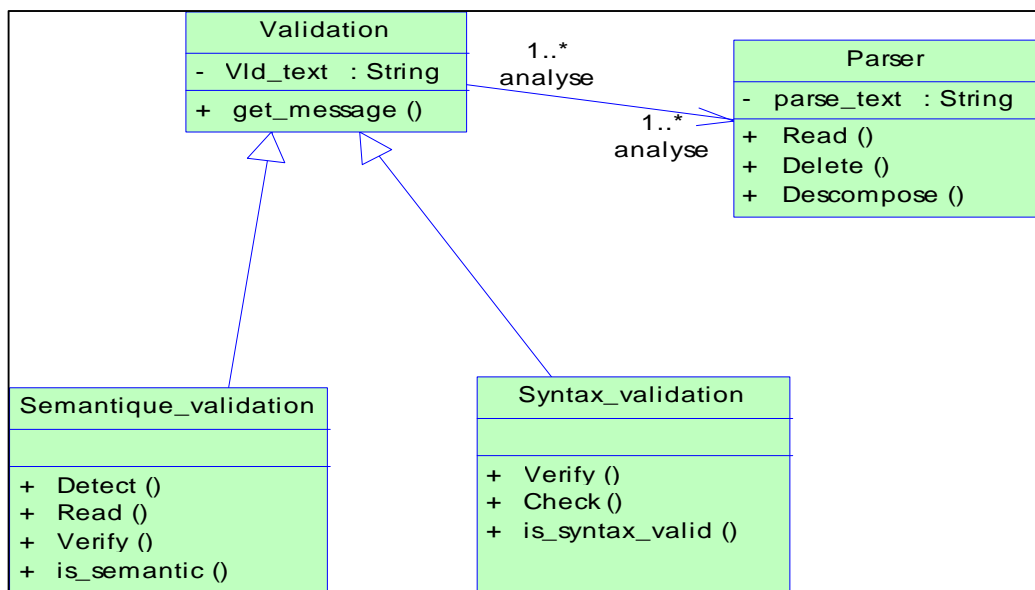


Figure 3.4. Diagramme de classe pour le package « Validation Manager »

Ce diagramme de classe contient les classes suivantes :

- Syntax_validation qui est responsable de la vérification syntaxique du message reçu.
- Parser qui effectue l’analyse des différents blocs du message XML
- Semantique_validation qui assure la validation sémantique du message reçu après son analyse.

3.3.3. Package « Business »

Ce module représente le cœur ou la fonctionnalité principale de notre plateforme. Son diagramme de classe est donné par la figure 3.5.

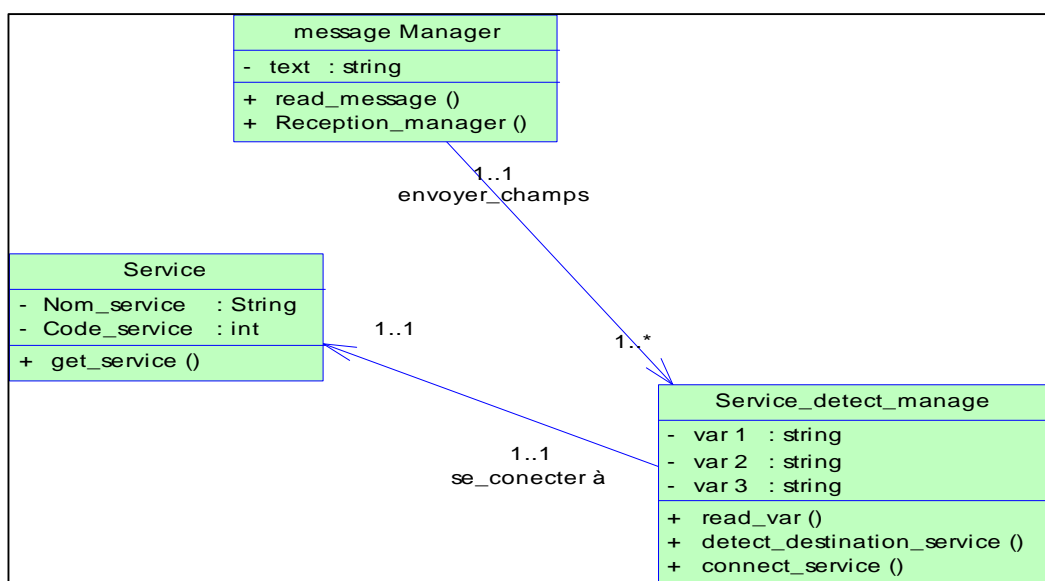


Figure 3.5. Diagramme de classe pour le package « Business »

Ce diagramme de classe représente les classes et leurs méthodes qui sont indispensables pour la fonction de détection du service à invoquer. Il contient les trois classes suivantes :

- message Manager : qui est responsable de la gestion des messages de réception et leurs décompositions en champs significatifs.
- Service_detect_manage : responsable de la détection du service à invoquer après l'analyse des différents champs qui lui sont envoyés par la table précédente, en plus elle contient une méthode qui permet de se connecter sur le service ainsi désigné.
- Service : responsable de l'exécution du service et du retour du résultat ou autrement la réponse au service source.

3.3.4. Package « Administration »

Son diagramme de classe est donné par la figure 3.6.

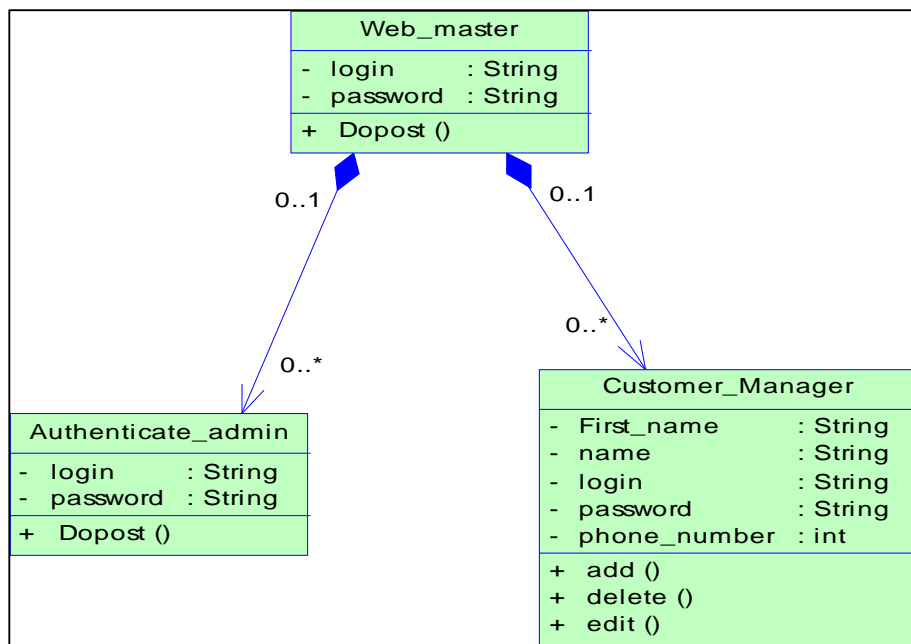


Figure 3.6. Diagramme de classe pour le package « Administration »

L'administrateur de la plateforme SOA doit s'authentifier, si l'opération d'authentification est menée à juste l'administrateur accède à l'interface Customer_Manager pour ajouter, modifier ou supprimer de nouveaux opérateurs. Donc, la conception d'une petite base de données qui contient les différents utilisateurs ou opérateurs est nécessaire.

3.4. Diagrammes de séquence

Un diagramme de séquence montre chronologiquement (de haut en bas) les interactions entre un ensemble d'objets pour un cas d'utilisation spécifique. Chaque objet dispose d'une ligne de vie (ligne verticale). Sur ces lignes de vie, des périodes d'activité sont indiquées par de rectangles fins.

Donc, il sera utile de présenter quelques scénarios qui présentent les différentes interactions entre les différents objets des classes développées précédemment.

3.4.1. Diagramme de séquence pour un scénario du package « XML Manager »

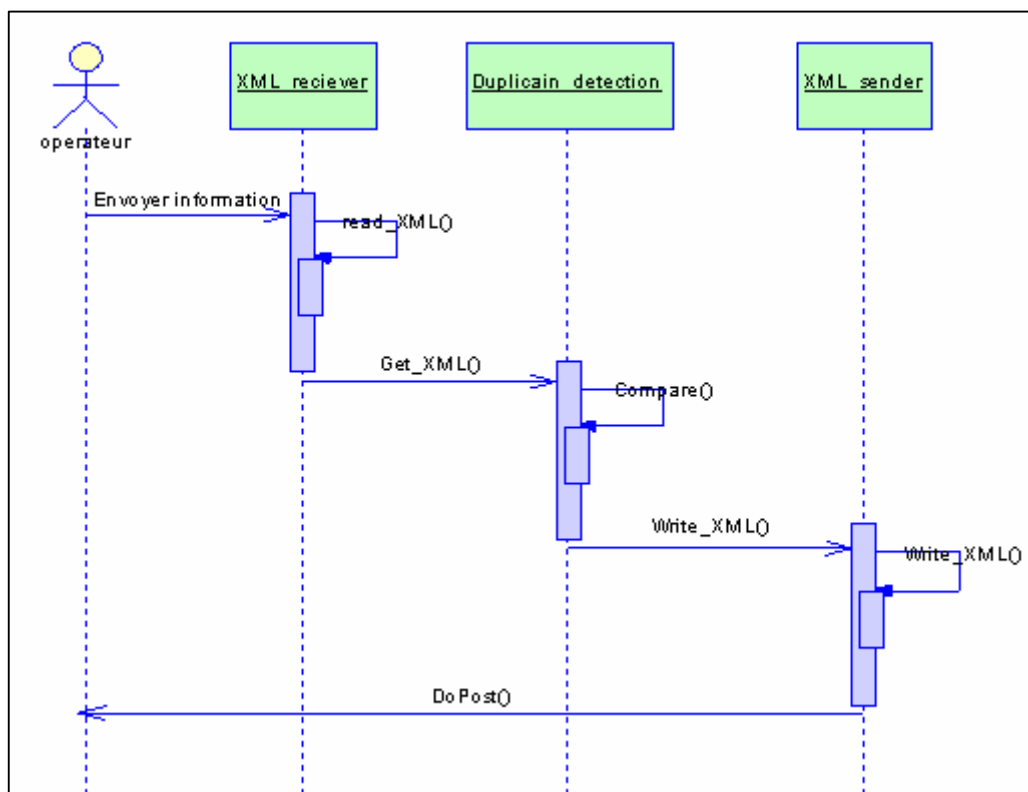


Figure 3.7. Diagramme de séquence pour un scénario du package « XML Manager »

La table « XML receiver », après la lecture du message reçu de la part de l'opérateur, va envoyer ce message à la classe « Duplication detection » qui va appliquer la méthode « compare() » et détecter que ce message a été déjà envoyé. Ainsi la table « XML sender » est informée et elle est sensée de structurer ou écrire le nouveau message XML pour informer par la suite l'opérateur de l'erreur via la méthode « Do Post() ».

3.4.2. Diagramme de séquence pour un scénario du package « Validation »

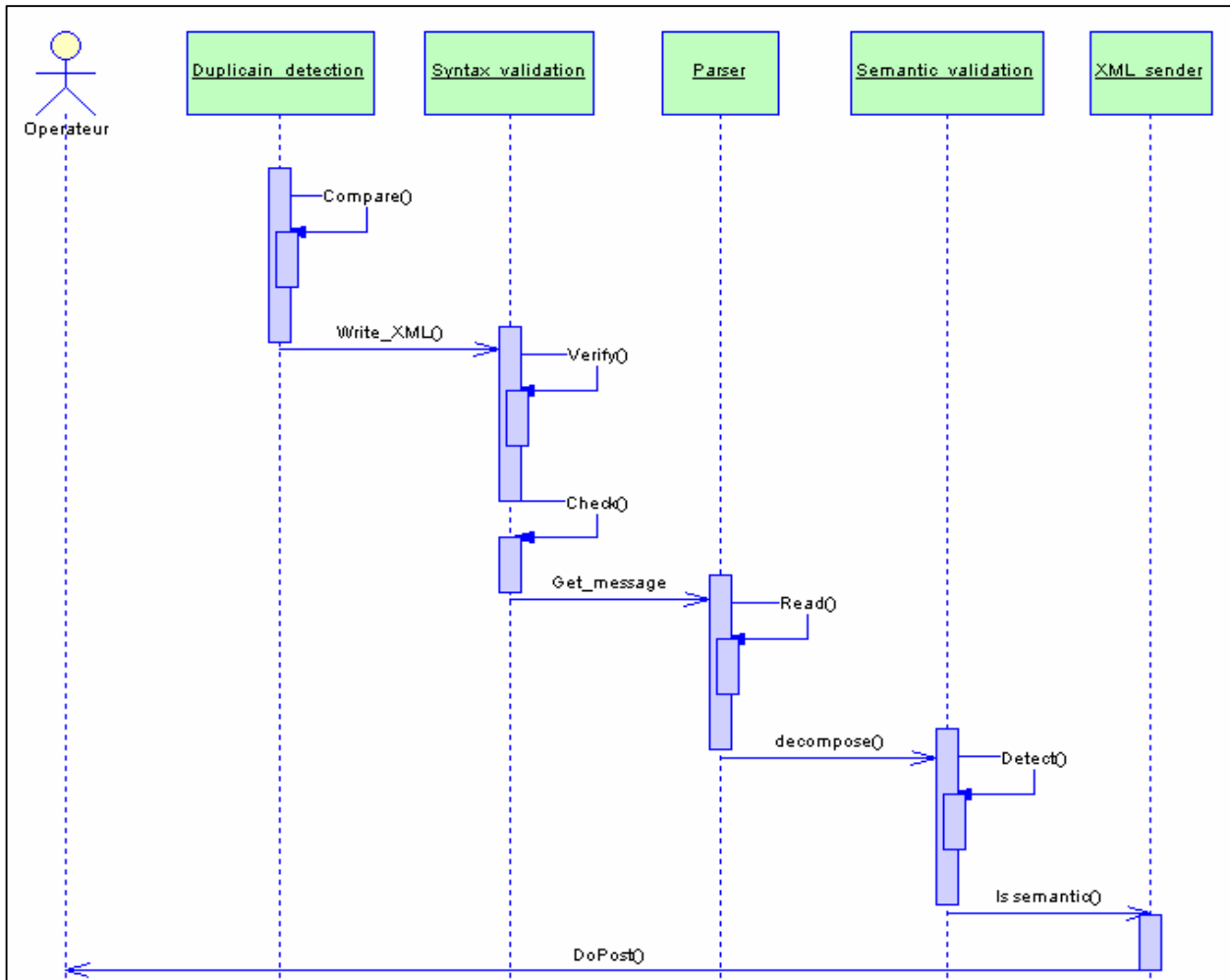


Figure 3.8. Diagramme de séquence pour un scénario du package « Validation »

La classe « Duplication detection », après avoir vérifié que le message reçu n'est pas dupliqué, envoie de nouveau un message XML à la classe « Syntax validation » qui vérifie que le message est syntaxiquement correct par l'application des méthodes « verify() » et « check() ». A son tour, elle envoie de nouveau le message XML à la classe « parser » pour l'analyser et le décomposer puis l'envoyer de nouveau à la classe « Semantic validation » qui détecte que le message n'est pas sémantiquement correct, donc elle informe la classe « XML sender » (envoi d'un message d'erreur) par l'application de la méthode « is semantic() ».

3.4.3. Diagramme de séquence pour un scénario du package « Business »

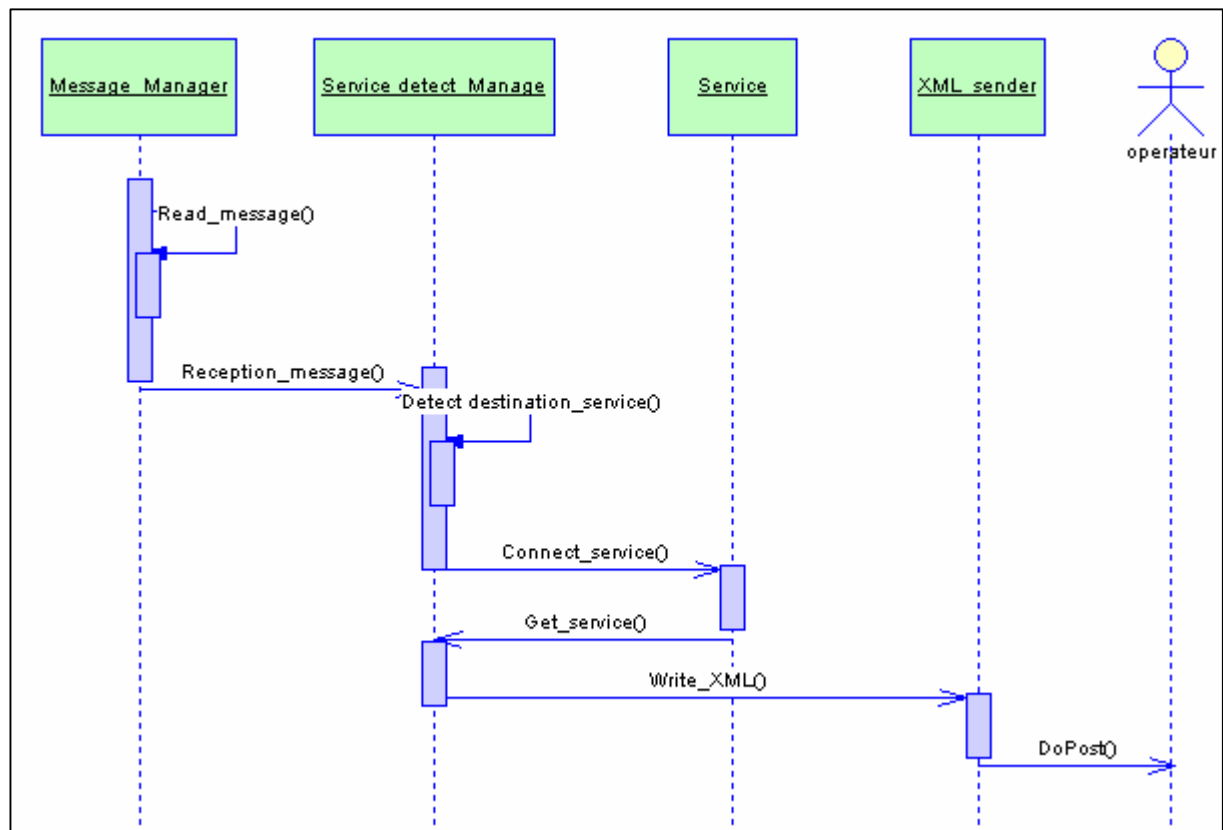


Figure 3.9. Diagramme de séquence pour un scénario du package « Business »

Ce diagramme de séquence montre comment invoquer un service désigné après avoir reçu le message XML qui est correct syntaxiquement et sémantiquement de la part de la classe « Validation ». Tout d'abord, la classe « Message Manager » applique la méthode « Read message() » pour dégager les différentes variables ou champs utiles dans le message XML, puis elle les envoie à la classe « Service detect Manage ». Cette classe va appliquer la méthode « Detect destination_service() » pour connaître le service destinataire et qui sera invoqué par la méthode « Connect_service() » qui consulte la classe « service » et retourne la réponse par la méthode « Get_service() ». Finalement, la classe « Service detect Manage » va envoyer la réponse à la classe « XML sender » qui à son tour doit envoyer de nouveau cette réponse à l'utilisateur par la méthode « Do Post »

Conclusion

Après avoir terminé la phase de conception, il sera facile d'entamer la phase d'implémentation surtout qu'on a défini toutes les classes avec leurs méthodes et les relations entre elles.

Le chapitre suivant, sera consacré alors pour la mise en pratique des résultats de la partie conception.

Chapitre 4

Réalisation

Introduction

Une des étapes de la vie d'un projet, aussi importante que la conception, est l'implémentation. Cette étape constitue la phase d'achèvement et d'aboutissement du projet. Pour accomplir cette tâche avec succès il faut savoir utiliser les outils adéquats et nécessaires. Ce choix d'outils peut influencer sur la qualité du produit obtenu et donc nécessite une attention particulière et doit se baser sur les besoins du projet et le résultat escompté.

Ce chapitre présente alors l'environnement technique du travail ainsi que le choix pris en matière d'environnement logiciel. Par la suite, nous présentons une démonstration pour le test du fonctionnement de notre plateforme.

4.1. Description de l'environnement de travail

Dans cette partie nous s'intéressons à l'étude de l'environnement technique disponible pour la réalisation de notre projet ensuite nous justifions les choix pris en matière d'environnement logiciel pour mener à terme la partie applicative.

4.1.1. Environnement technique

Pour la réalisation et la concrétisation de tel projet, nous devons au début choisir notre middleware ou ESB (Entreprise Service Bus), nous avons été face à choix entre ces différents middlewares [5] :

- Cape Clear Enterprise Service Bus 6.0 de l'entreprise Cape Clear : Suite complète couvrant l'orchestration des processus, le transport, le routage et la transformation des messages. Les standards supportés sont: JMS (Java Message Service), SOAP (Simple Object Acces Protocol), BPEL (Business Process Execution Langage), JCA (J2EE Connector Architecture). Il Gère la sécurité des services et dispose d'une console de supervision.
- NetZyme Enterprise de l'entreprise Creative Science : OOM (Object Oriented Middleware)

très complet, comprenant presque toutes les fonctions d'un ESB. Support natif C, .Net et Java. Supporte JMS, JCA, SOAP, BPEL.

- Fiorano ESB de l'entreprise Fiorano : Outils de modélisation, mapping, administration, etc. Supporte J2EE (Java 2 Enterprise Edition), .Net, JMS, SOAP.

- Websphere MQ 6.0 de la société IBM : Bâti sur une souche MQ Series. 80 plates-formes supportées. S'intègre à Websphere Application Server 6.0. Supporte SOAP, JMS. Outil de configuration basé sur Eclipse.

- Artix 3.0 de la société Iona : Expose des services Corba et Java sous la forme de services SOAP, XML, etc. Il assure l'agrégation de services et dispose d'un environnement de développement livré avec le bus. Il garantit la génération automatique des proxies de services (WSDL-to-Corba, etc.). Transport sur la plupart des protocoles (HTTP (Hypertext Transfer Protocol), Servlet, IIOP (Internet inter_ORB Protocol), JMS, Websphere MQ, Tibco, Tuxedo).

- Sonic ESB 6.1 de la société Sonic Software : S'appuie sur SonicMQ. Caractérisé par un routage intelligent des messages, architectures distribuées, moteur de transformation de messages à la volée, couche de sécurité fournie par RSA. Il dispose de nombreux connecteurs. Sonic est considéré comme l'éditeur à l'origine du concept d'ESB.

- Mule de la société SymphonySoft : il est un projet d'ESB libre, supporte plusieurs protocoles et expose des services Java et Corba sous la forme de services SOAP, XML.

La plupart des outils cités en avant sont propriétaires donc sont plus riches que les outils open source. Mais, nous avons été obligé de choisir un ESB ou middleware à source libre à savoir Mule et plus précisément la version Mule 1.2.

4.1.1.1. Présentation et architecture générale de Mule

Le but final de mule est de fournir une méthode unifiée d'agir sur des données ou des sources hétérogènes sans encombrer le réalisateur avec les détails du sujet, de la façon dont les données sont envoyées ou reçues ou les protocoles impliqués. Le résultat est donc un serveur d'ESB (autobus de service d'entreprise) qui est léger, rapide et simple pour reprendre et démarrer l'exécution des différents services. L'architecture de la mule a été conçue avec le modèle d'ESB d'origine, son foyer primaire est simple et accélère le processus de développement des réseaux distribués de services. Généralement, la notion d'ESB est liée aux

Chapitre 4 : Réalisation

grands projets d'entreprise mais Mule rend l'utilisation d'ESB possible pour les petits projets d'entreprise grâce à son architecture simple donnée par la figure 4.1 [6].

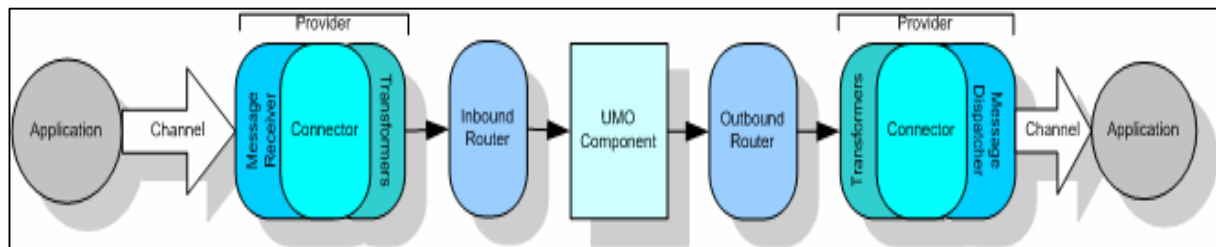


Figure 4.1. Architecture générale de Mule

Le but primaire de cette architecture est de permettre l'intégration des différentes applications utilisant différentes normes par l'utilisation des modèles bien définis. Pour réaliser ce but, Mule définit un ensemble de composants qui peuvent être employés pour effectuer la majeure partie et permettre ainsi aux différents services de parler ensemble.

Cette architecture montre une topologie de bout en bout et une façon simple pour la connexion et l'adaptation entre les différents composants cités ci-dessous :

L'Application

L'application peut être de n'importe quel type, tel qu'une application web, bureautique, un serveur d'application.

Le Canal

Peut être n'importe quelle méthode de communiquer des données entre deux points. Des canaux sont utilisés dans Mule pour câbler des composants d'UMO ensemble aussi bien que pour câbler différents nœuds de Mule ensemble à travers un réseau local ou l'Internet.

Le Récepteur de message

Le récepteur de message est employé pour lire ou recevoir des données de l'application. Dans Mule un récepteur est juste un élément d'un fournisseur de transport et Mule fournit beaucoup de transports tels que les JMS, le savon, le HTTP, le TCP (Transmission Control Protocol), le XMPP (Extensible Messaging and Presence Protocol), le SMTP (Simple Mail Transfer Protocol), le dossier, etc.

L'intérêt de Mule est sa capacité de communiquer aux systèmes distribués sans savoir leurs logiques d'affaires (composants d'UMO), sans savoir l'endroit de système, le format des données et le mécanisme de livraison ou des protocoles impliqués. De même quand le composant d'UMO a effectué son travail, il n'a besoin de s'inquiéter où les données iront après ou dans quel format elles sont prévues.

Le Routeur d'arrivée

Des routeurs d'arrivée peuvent être utilisés pour connaître les événements reçus par un composant souscrivant sur un canal et les commander. Des routeurs d'arrivée peuvent être utilisés pour filtrer, agréger et assurer le séquencement des événements avant qu'ils soient reçus par le composant d'UMO.

Le Connecteur

Le connecteur doit décider comment envoyer et recevoir des données au-dessus d'un canal particulier. Un récepteur de message est couplé à un connecteur pour se concentrer aux données venant d'une source détectée par le connecteur.

Les Transformateurs

Des transformateurs sont utilisés pour transformer des messages utiles en des différents types. Mule ne définit pas un format standard de message (bien que la mule puisse soutenir le message standard de définition de processus d'affaires). Ainsi la transformation fournie par la boîte extérieure est comme le type transformation du message JMS à des transformateurs standard de XML. La transformation de données est très subjective à l'application.

Le point final

Un point final est vraiment un emballage de configuration qui lie un connecteur, un URI de point final, des transformateurs, des filtres et une information transactionnelle pour fournir un adaptateur de canal (équivalent à un fournisseur).

Le routeur de départ

Des routeurs de départ sont utilisés pour éditer des messages ou des événements aux différents fournisseurs selon les différents aspects des événements ou d'autres règles définies dans la configuration.

Pour le développement de notre plateforme SOA, nous avons utilisé ces différents composants implémentés au niveau de Mule ou ce qu'on a appelé au niveau de la décomposition en packages au niveau du chapitre 3 (paragraphe 2) le module « Mule » qui va

être appelé par la package « Business » et en plus nous avons développé nos propres classes ou autrement dit nos propres composants dont on précisera les outils logiciels de développement au paragraphe suivant.

4.1.2. Environnement logiciel

Les choix pris en matière d'environnement logiciel concernent les langages de développement, l'environnement de développement, le SGBD (Système de Gestion de Base des données) gérant la base des données interne à l'application, le serveur web déployant l'application et le protocole de communication client /serveur à utiliser.

4.1.2.1. Les langages de développement

- **Java**

Java est à la fois un langage de programmation et une plateforme d'exécution. Le langage Java a la particularité principale d'être portable sur plusieurs systèmes d'exploitation. Il permet aussi de développer des applications autonomes et surtout, des applications client/serveur.

Pour toutes ces raisons et pour les besoins de notre application, le choix du java s'est avéré le plus judicieux et le mieux approprié.

- **XML (eXtensible Markup Language) (Voir annexe B)**

C'est une famille de langages de représentation de données dans un format particulier. Ce format repose sur la présence de balises (langages de balisage).

XML n'est pas un langage à proprement parler comme peut l'être HTML : XML est une famille de langages ayant en commun le respect de certaines règles. Nous allons voir que là où HTML (Hypertext markup language) est simple, bien défini et non contraignant à la fois, XML est extensible et rigoureux.

Les buts et avantages de XML sont, entre autres, de :

- ✓ Représenter des données pour les manipuler, favoriser l'interopérabilité, l'échange ;
- ✓ Rendre pérennes les données ;
- ✓ Les rendre manipulables à la fois par les hommes et les machines.

4.1.2.2. Outils CASE

Les outils CASE (Computer Aided Software Engineering) sont des outils de génie logiciel assisté par ordinateur. Nous allons présenter ceux que nous avons utilisé pendant la conception et jusqu'à l'implémentation de notre application.

4.1.2.2.1. PowerDesigner 9

Power Designer fournit un ensemble d'outils de modélisation graphique adaptés au développement de solutions pour des environnements de systèmes temps réel. Power Designer est le leader mondial en outil de modélisation UML. Cet outil permet entre autre de :

- ✓ Concevoir et identifier les objets métier.
- ✓ Concevoir la répartition des composants à travers un réseau.
- ✓ Générer des architectures préfabriquées (frameworks).
- ✓ Utiliser des possibilités de rétro-ingénierie pour créer des modèles à partir de composants d'applications existantes.
- ✓ Analyser des scénarios métier avec des diagrammes de séquences et de collaboration.
- ✓ Modéliser les états.
- ✓ Générer du code.

4.1.2.3. Environnement de développement

Notre application est destinée à s'exécuter sous une plate-forme Windows donc il est judicieux de choisir un environnement de développement java sous Windows pour tirer au maximum de profit de la plate-forme de travail. Par conséquent, JBuilderX 2005 Entreprise Edition a été choisi.

JBuilderX est un environnement complet permettant la génération, à l'aide du langage java, des solutions manipulant des bases de données et le développement des services Web par différents langages : HTML, JS, JSP, ect.

4.1.2.4. Serveur web Axis (Boîte à outils Apache Axis)

Il est implémenté au niveau de Mule pour le développement des services web et qui implémente le protocole SOAP et supporte le langage java. Apache Axis est une nouvelle implémentation de la spécification SOAP (Simple Object Access Protocol) développée par la fondation Apache (The Apache Software Foundation), qui succède à Apache SOAP. Axis est à la fois un environnement d'hébergement de services Web et un toolkit complet de développement pour la création de services et l'accès à des services tiers.

4.1.2.5. Protocole de communication Client/Serveur

Puisque on a choisi le serveur axis donc on est obligé de choisir SOAP comme protocole pour le transfert des données.

4.1.2.6. Système de Gestion de Base de Donnée MySQL

MySQL est une base de données open source la plus populaire dans le monde. Elle se caractérise par sa performance, sa haute fiabilité et sa simplicité d'utilisation. Elle est utilisée par des grandes entreprises transnationales. MySQL fonctionne sur plus de vingt plateformes, notamment Linux, Windows, etc.

4.2. Test du fonctionnement de la plateforme

Au niveau de notre travail, on s'est intéressé au développement des différentes nouvelles classes à intégrer au niveau de notre middleware Mule. Ces classes assurent les différentes fonctionnalités d'une architecture SOA citées dans le chapitre 2 (paragraphe 2) et permettant ainsi d'intégrer d'une manière générique n'importe quel service d'intermédiation documentaire.

Nous n'avons pas développé des applications clients, donc il suffit d'appeler un service (notre cas le service TCE : Titre de Commerce Extérieur) par un navigateur ou générer son propre fichier (.bat) qui sera exécuté par l'invite de commande Ms Dos pour tester son bon fonctionnement. Pour le déploiement d'un nouveau service sur notre plateforme SOA (avec les nouvelles classes développées) on doit ajouter les fichiers suivants :

- Fichier « source » : Contient les différentes classes du service à déployer.
- Fichier « bin » : Contient le fichier (.bat) pour l'exécution du service. Son contenu est indiqué par la figure 4.2 pour le cas du service TCE.
- Fichier « config » : Contient le fichier de configuration XML du service et qui est appelé par le fichier (.bat) lors de l'exécution du service. Un exemple de ce fichier est donné par la figure 4.3 pour le cas du service TCE.

```
@echo off
REM There is no need to call this if you set the MULE_HOME in your environment properties
SET MULE_HOME=..\..\..

REM Set your application specific classpath like this
SET CLASSPATH=%MULE_HOME%\Services\TCE\conf;%MULE_HOME%\Services\TCE\classes

call %MULE_HOME%\bin\mule.bat -config ../conf/TCE-mule-config.xml

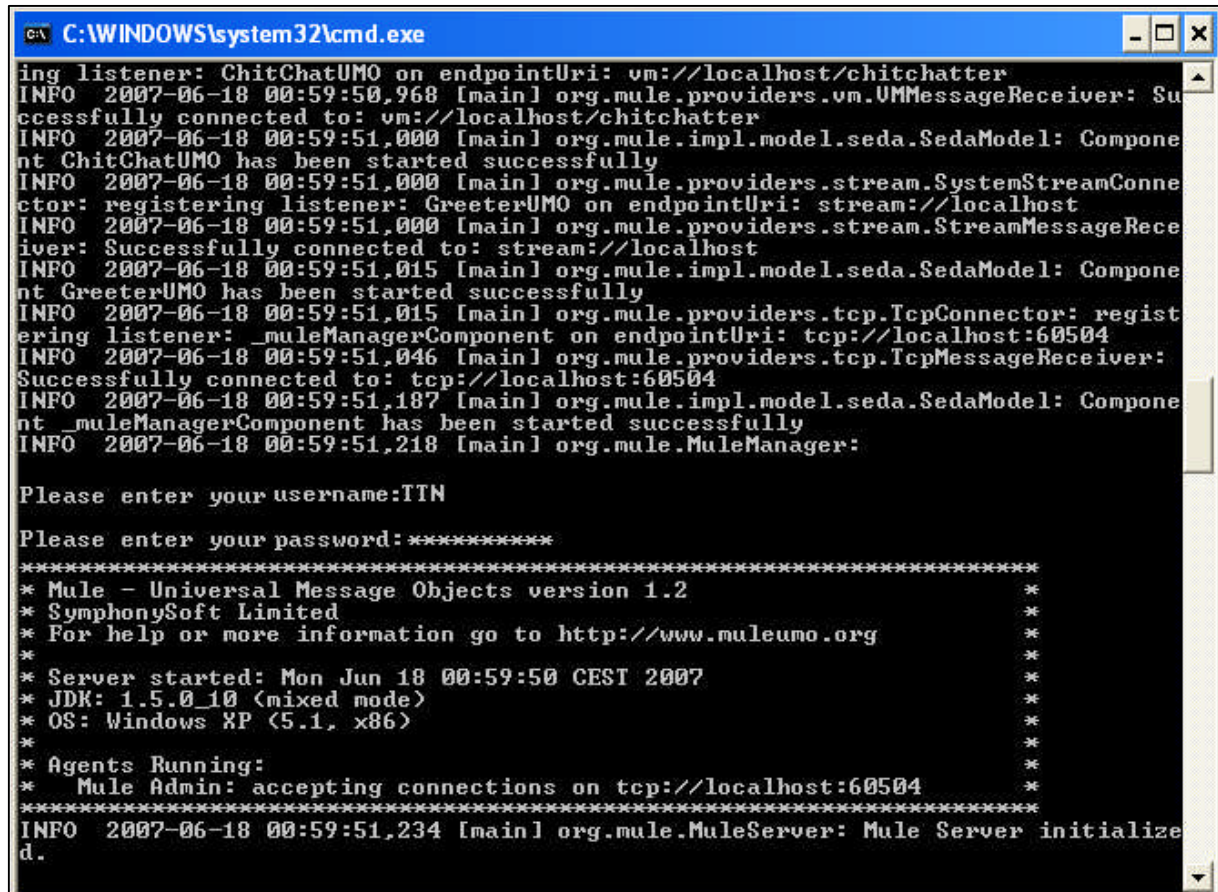
SET CLASSPATH=
```

Figure 4.2. Contenu du fichier TCE.bat

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE mule-configuration PUBLIC "-//SymphonySoft //DTD mule-configuration XML V1.0//EN"
      "http://www.symphonysoft.com/dtds/mule/mule-configuration.dtd">
<mule-configuration id="Mule_intermediation_Sample_over_Http" version="1.0">
  <mule-environment-properties synchronous="true"/>
  <interceptor-stack name="default">
    <interceptor className="org.mule.interceptors.LoggingInterceptor"/>
    <interceptor className="org.mule.interceptors.TimerInterceptor"/>
  </interceptor-stack>
  <model name="Intermédiation_sample">
    <mule-descriptor name="TCE_Service"
      implementation="org.mule.based.exemple.TCE">
      <inbound-router>
        <endpoint address="axis:http://localhost:8082/services"/>
      </inbound-router>
      <properties>
        <property name="style" value="wrapped"/>
        <property name="use" value="literal"/>
      </properties>
    </mule-descriptor>
    <mule-descriptor name="Invoque_service"
      implementation="org.mule.components.simple.BridgeComponent">
      <inbound-router>
        <endpoint address="vm://Invoque_service"/>
      </inbound-router>
      <outbound-router>
        <router className="org.mule.routing.outbound.OutboundPassThroughRouter">
          <endpoint
            address="axis:http://localhost:8082/services/TCE?method=acceder_service">
            <properties>
              <property name="methodNamespace"
                value="http://Services.TCE.mule.org"/>
              <property value="wrapped" name="style"/>
              <property value="literal" name="use"/>
            </properties>
          </endpoint>
        </router>
      </outbound-router>
      <interceptor name="default"/>
    </mule-descriptor>
```

Figure 4.3. Fichier de configuration pour le service TCE

Après avoir déployé le service TCE, on l'exécute à l'aide de l'invite de commande Ms Dos en introduisant le nom de son fichier.bat (TCE.bat). Le résultat de l'exécution est donné par la figure 4.4.



```
C:\WINDOWS\system32\cmd.exe
ing listener: ChitChatUMO on endpointUri: vm://localhost/chitchatter
INFO 2007-06-18 00:59:50,968 [main] org.mule.providers.vm.UMLMessageReceiver: Su
ccessfully connected to: vm://localhost/chitchatter
INFO 2007-06-18 00:59:51,000 [main] org.mule.impl.model.seda.SedaModel: Compone
nt ChitChatUMO has been started successfully
INFO 2007-06-18 00:59:51,000 [main] org.mule.providers.stream.SystemStreamConne
ctor: registering listener: GreeterUMO on endpointUri: stream://localhost
INFO 2007-06-18 00:59:51,000 [main] org.mule.providers.stream.StreamMessageRece
iver: Successfully connected to: stream://localhost
INFO 2007-06-18 00:59:51,015 [main] org.mule.impl.model.seda.SedaModel: Compone
nt GreeterUMO has been started successfully
INFO 2007-06-18 00:59:51,015 [main] org.mule.providers.tcp.TcpConnector: regist
ering listener: _muleManagerComponent on endpointUri: tcp://localhost:60504
INFO 2007-06-18 00:59:51,046 [main] org.mule.providers.tcp.TcpMessageReceiver:
Successfully connected to: tcp://localhost:60504
INFO 2007-06-18 00:59:51,187 [main] org.mule.impl.model.seda.SedaModel: Compone
nt _muleManagerComponent has been started successfully
INFO 2007-06-18 00:59:51,218 [main] org.mule.MuleManager:

Please enter your username:TTN

Please enter your password:*****
*****
* Mule - Universal Message Objects version 1.2 *
* SymphonySoft Limited *
* For help or more information go to http://www.muleumo.org *
* *
* Server started: Mon Jun 18 00:59:50 CEST 2007 *
* JDK: 1.5.0_10 (mixed mode) *
* OS: Windows XP (5.1, x86) *
* *
* Agents Running: *
* Mule Admin: accepting connections on tcp://localhost:60504 *
*****
INFO 2007-06-18 00:59:51,234 [main] org.mule.MuleServer: Mule Server initialize
d.
```

Figure 4.4. Exécution du service TCE

Pour visualiser le résultat du service invoqué, on utilise un navigateur dans lequel on introduit l'adresse de localisation du service. Un exemple de réponse est donné par la figure 4.5 dans le cas où le service est déjà invoqué avec les mêmes paramètres ou autrement les mêmes champs remplis par l'utilisateur. C'est une duplication du message XML envoyé qui est détectée par la classe « Duplication_Detection ».

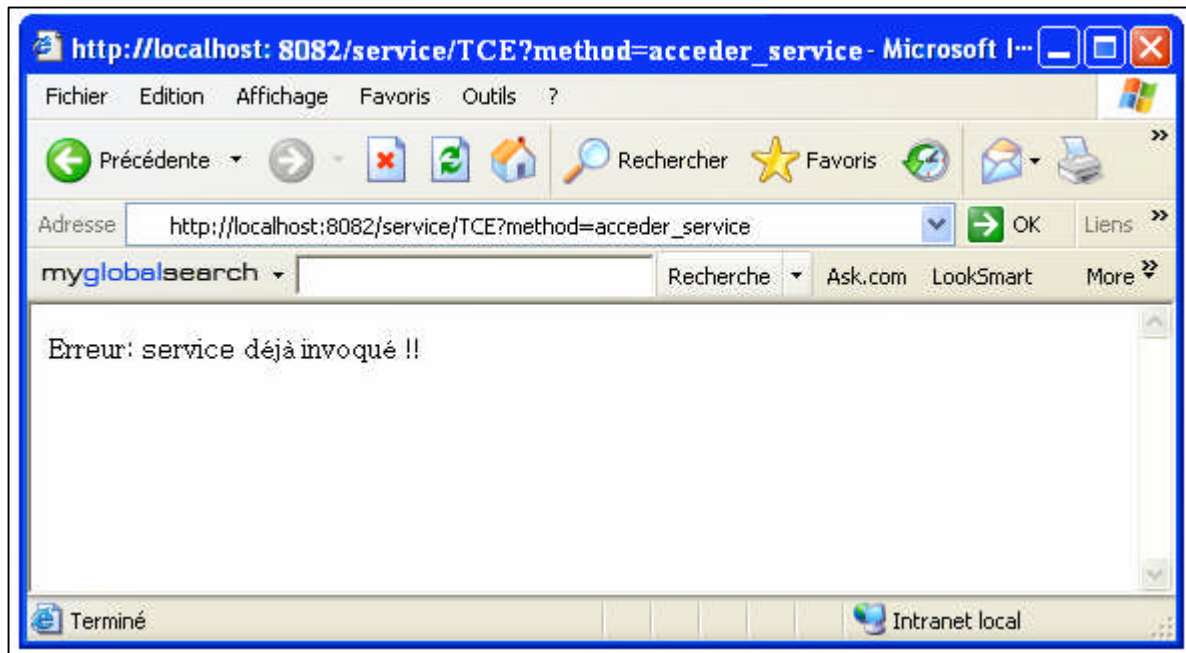


Figure 4.5. Réponse du service TCE après son invocation

Conclusion

Dans ce chapitre nous avons présenté l'environnement technique et logiciel utilisé lors du projet. Nous avons également présenté les étapes de la réalisation de notre solution SOA pour l'implémentation des services d'intermédiation documentaire ainsi que quelques captures d'écran montrant le bon fonctionnement de notre plateforme pour le service TCE.

Conclusion générale

Ce projet avait pour objectif la réalisation d'une plateforme générique permettant d'implémenter de la même manière les services d'intermédiation documentaire. Dans une première étape, nous avons choisis l'architecture orientée service (SOA) et les services Web comme solutions techniques pour assurer la connexion entre les différents agents intervenant dans un service. Dans une deuxième étape, on a pris le service TCE (Titre de Commerce Extérieur) comme exemple de service d'intermédiation documentaire pour vérifier le bon fonctionnement de notre plateforme SOA.

On tient à préciser que ce projet nous a été bénéfique puisqu'il nous a permis de découvrir un champ d'application pratique, vaste et riche en procédures, il nous a donné aussi l'occasion de bien tester nos connaissances théoriques.

De plus, ce projet nous a permis d'étudier la norme SOAP dont l'importance est vitale pour assurer la communication entre un service Web et application appelante, maîtriser les langages de développement orientés objet « Java », apprendre le langage de description des données « XML », maîtriser l'utilisation du système de gestion de base de données MySQL, se familiariser avec plusieurs environnements techniques: JBuilderX, Axis, Power Designor.

La réalisation de notre plateforme SOA ne signifie pas qu'elle n'est pas susceptible à être améliorée. En effet, la notion de la sécurité nous inspire pour ajouter d'autres fonctionnalités à notre plateforme et d'améliorer celles qui existent.

Annexe A

Architecture des services web

Les protocoles HTTP et HTTPS servent à transporter les données sur le réseau entre le consommateur et le service web. Les données sont sous le format XML et/ou SOAP. Le serveur où se trouve le service web devra être capable d'exécuter le code de celui-ci et nécessitera IIS et le framework. La machine du consommateur devra quant à elle disposer de IIS (figure A.1).

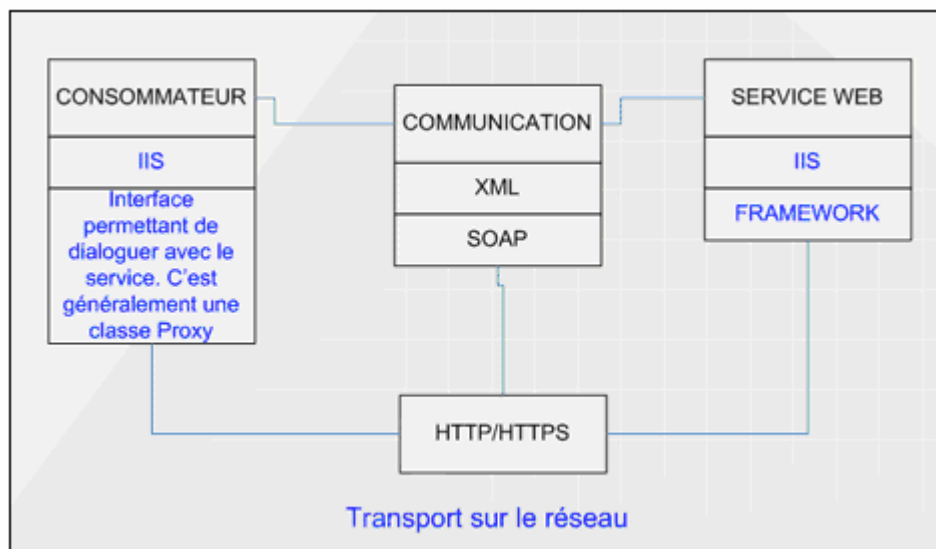


Figure A.1. Architecture des services web

A.1. Qu'est-ce que SOAP?

SOAP est un protocole de transmission de messages. Il définit un ensemble de règles pour structurer des messages qui peuvent être utilisés dans de simples transmissions unidirectionnelles, mais il est particulièrement utile pour exécuter des dialogues requête-réponse RPC (Remote Procedure Call). Il n'est pas lié à un protocole particulier mais HTTP

est populaire. Il n'est pas non plus lié à un système d'exploitation ni à un langage de programmation, donc, théoriquement, les clients et serveurs de ces dialogues peuvent tourner sur n'importe quelle plate-forme et être écrits dans n'importe quel langage du moment qu'ils puissent formuler et comprendre des messages SOAP. En tant que tel, il s'agit d'un important composant de base pour développer des applications distribuées qui exploitent des fonctionnalités publiées comme services par des intranets ou internet.

A.2. Messages SOAP

Un message SOAP valide est un document XML au bon format. (Pour plus de détails. Le prologue XML peut être présent, mais dans ce cas, ne doit contenir qu'une déclaration XML (c'est à dire qu'il ne doit contenir ni référence à un DTD, ni instruction XML). Le message doit utiliser l'enveloppe SOAP et les namespaces d'encodage SOAP, et doit avoir la forme suivante:

- Une déclaration XML (optionnelle), suivie de
- une Enveloppe SOAP (l'élément racine) qui est composée de:
 - Une En-tête SOAP (optionnel)
 - Un Corps SOAP

Un dialogue RPC encodé par SOAP contient un message de requête et un message de réponse. Considérons la méthode d'un service simple qui double la valeur d'un entier donné (figure A.2).

Signature de la Methode

```
int doubleAnInteger ( int numberToDouble );
```

Requête

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
  SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doubleAnInteger
      xmlns:ns1="urn:MySoapServices">
      <param1 xsi:type="xsd:int">123</param1>
    </ns1:doubleAnInteger>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Réponse

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doubleAnIntegerResponse
      xmlns:ns1="urn:MySoapServices"
      SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:int">246</return>
    </ns1:doubleAnIntegerResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Figure A.2. Message SOAP pour l'exemple d'un service
qui double la valeur d'un entier**

Notez que la plupart des packages SOAP que vous utiliserez prendront en charge les détails de syntaxe des messages SOAP pour vous, mais afin d'être capable de lire, de comprendre et de déboguer des dialogues SOAP, regardons ces messages point par point. Le prologue XML contient seulement une déclaration XML `<?xml version="1.0" encoding="UTF-8" ?>` spécifiant la version de XML et l'encodage des caractères du message XML.

Le tag de l'Enveloppe SOAP `<SOAP-ENV:Envelope ... >` dans le message de requête spécifie tout d'abord que le style d'encodage de ce message SOAP suit le schéma défini dans <http://schemas.xmlsoap.org/soap/encoding/>. Notez que c'est optionnel et que ce sera présumé si ce n'est pas inclus comme c'est le cas dans le message de réponse. L'Enveloppe SOAP contient également des définitions de namespaces. Les identifiants des namespaces sont standards et la spécification SOAP demande à ce que ces namespaces soient définis correctement ou pas du tout (c'est à dire qu'un message SOAP dans lequel manquent des définitions de namespaces est correct et peut être exploité mais un message contenant des définitions incorrectes, c'est à dire non standards, est mauvais et refusé). Notez que la définition du namespace SOAP-ENC est absente du message de réponse mais cela ne signifie pas que le message est invalide.

Il n'y a pas de tag header (en-tête) SOAP dans cet exemple. Les en-têtes SOAP sont optionnelles et sont typiquement utilisées pour transmettre des données d'authentification ou de gestion de session. Il est important de se rappeler que l'authentification et la gestion de session sont en dehors du cadre du protocole SOAP, même si les designers de SOAP autorisent une certaine flexibilité dans la transmission de messages SOAP, de telle façon que les personnes qui les implémentent puissent inclure de telles informations.

Vient ensuite le tag SOAP Body (le corps) `<SOAP-ENV:Body>` qui n'a rien de remarquable en lui-même mais encapsule un unique tag de méthode qui porte le nom de la méthode elle-même `<ns1:doubleAnInteger ... >` (ou, le même nom suivi du mot "Response" dans le cas du message de réponse). Notez que le tag de la méthode reçoit typiquement le namespace correspondant au nom du service, dans notre cas `urn:MySoapServices` pour assurer l'unicité (un service web, qui peut contenir n'importe quel nombre de méthodes nommées différemment, a un nom de service unique à l'URL sur laquelle il est accessible - plus de détails peuvent être trouvés sur les URLs de service ainsi que les noms de services et de méthodes dans la section SOAP Côté Serveur).

Le tag de méthode encapsule à son tour n'importe quel nombre de paramètres, tel que le tag `<param1 ... >` dans l'enveloppe de requête. Les noms des tags de paramètres peuvent être n'importe quoi, sont typiquement autogénérés et n'ont pas de namespace. Dans le message de réponse, il n'y a jamais qu'un seul tag de paramètre (représentant la valeur de retour de la méthode) et il est typiquement appelé `<return>`.

A.3. Eléments du Système

Dans cette section, nous verrons ce que sont les éléments principaux du système, ce qu'ils font et comment ils interagissent.

Voyons le Côté Client d'abord (figure A.3). Comme nous l'avons vu, tous nos dialogues avec le serveur exécutant le service web sont faits via SOAP. Rappelez-vous, SOAP ne spécifie pas le transport mais HTTP est commun. Cet exemple présume l'utilisation d'HTTP (mais vous pourriez tout aussi bien utiliser SMP). En tant que tels, nos messages au serveur seront des requêtes SOAP-XML enveloppées dans des requêtes HTTP. De même, les réponses du serveur seront des réponses HTTP qui renferment des réponses SOAP-XML. Maintenant, nous, développeurs côté client, nous ne voulons pas nous occuper de tous les détails de sérialisation SOAP et de l'encodage de HTTP. Nous employons alors un package SOAP qui le fera pour nous. Il s'agit typiquement d'une librairie que nous linkons dans notre propre code client. Nous invoquons ensuite le service, simplement en invoquant la méthode appropriée du package SOAP (typiquement en spécifiant l'URL du service, le nom du service et tous les paramètres requis). Le premier travail d'un package est de sérialiser l'invocation de ce service en requête SOAP. Il doit ensuite encoder ce message dans une requête HTTP et l'envoyer à l'URL spécifiée.

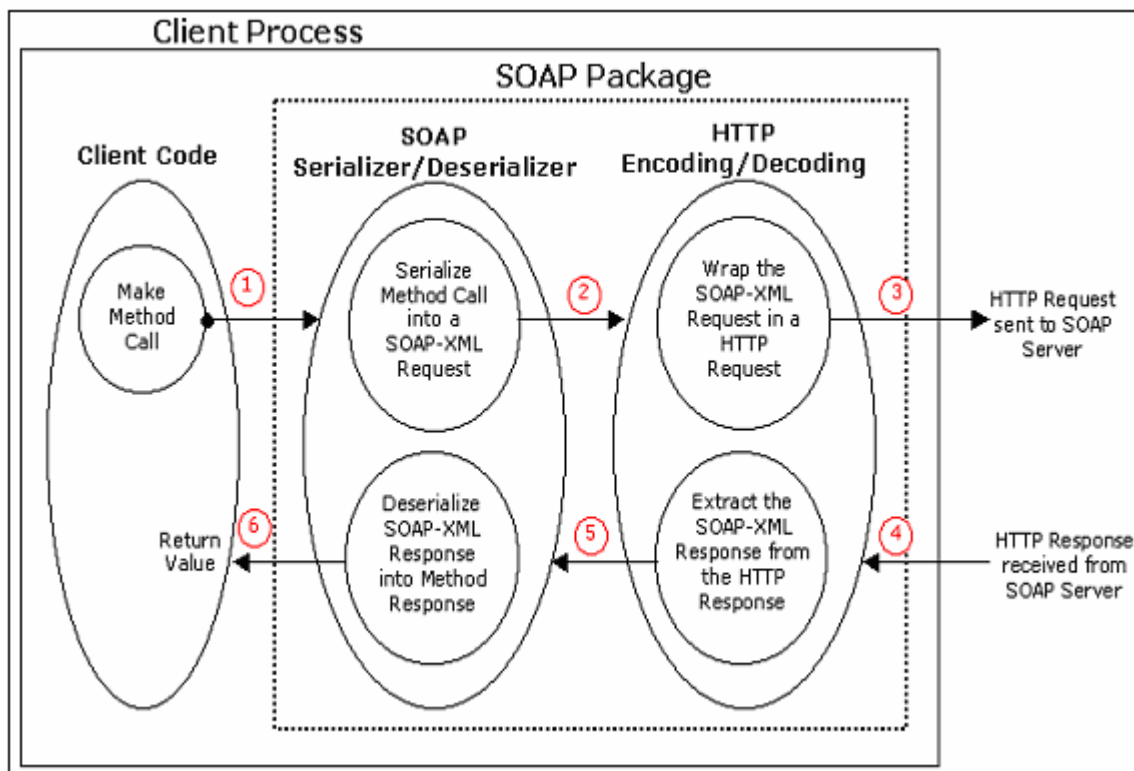


Figure A.3. Processus côté client

Voyons maintenant ce qui se passe du Côté Serveur (figure A.4). C'est légèrement plus complexe car nous avons besoin d'un process "listener" (Le Listener est le process serveur qui est en attente de connexion client). Nous avons également besoin d'une implémentation du service lui-même. A part cela, nous nous reposons sur un package SOAP de la même façon que du côté client.

Le listener est souvent implémenté au travers d'un servlet qui s'exécute comme une application web sur un appserver (serveur d'application web) (comme c'est le cas lorsque nous utilisons Apache SOAP du côté serveur). L'appserver sera configuré pour passer toutes les requêtes destinées à une certaine URL (l'URL du service SOAP) à un servlet particulier (appelons-le servlet SOAP). Le travail du servlet SOAP est d'extraire le message XML-SOAP de la requête HTTP, de le désérialiser (de ce fait, séparer le nom de la méthode et les paramètres fournis), et d'invoquer la méthode du service en conséquence. Le résultat de la méthode est alors sérialisé, encodé HTTP et renvoyé au demandeur.

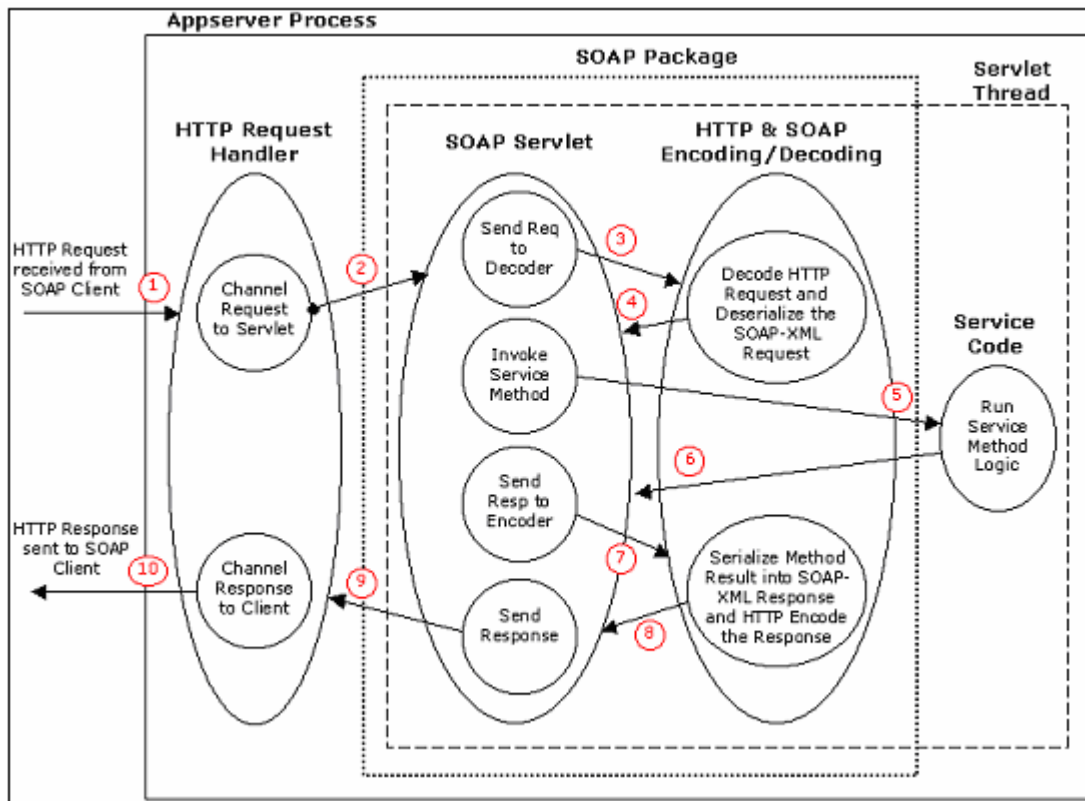


Figure A.4. Processus côté serveur

Annexe B

XML (eXtensible Markup Language)

B.1. Présentation

XML (entendez eXtensible Markup Language et traduisez Langage à balises étendu, ou Langage à balises extensible) est en quelque sorte un langage HTML amélioré permettant de définir de nouvelles balises. Il s'agit effectivement d'un langage permettant de mettre en forme des documents grâce à des balises (markup).

Contrairement à HTML, qui est à considérer comme un langage défini et figé (avec un nombre de balises limité), XML peut être considéré comme un métalangage permettant de définir d'autres langages, c'est-à-dire définir de nouvelles balises permettant de décrire la présentation d'un texte (Qui n'a jamais désiré une balise qui n'existait pas?).

La force de XML réside dans sa capacité à pouvoir décrire n'importe quel domaine de données grâce à son extensibilité. Il va permettre de structurer, poser le vocabulaire et la syntaxe des données qu'il va contenir.

En réalité les balises XML décrivent le contenu plutôt que la présentation (contrairement à HTML). Ainsi, XML permet de séparer le contenu de la présentation. Ce qui permet par exemple d'afficher un même document sur des applications ou des périphériques différents sans pour autant nécessiter de créer autant de versions du document que l'on nécessite de représentations.

XML a été mis au point par le XML Working Group sous l'égide du World Wide Web Consortium (W3C) dès 1996. Depuis le 10 février 1998, les spécifications XML 1.0 ont été reconnues comme recommandations par le W3C, ce qui en fait un langage reconnu.

B.2. Mise en page de XML

XML est un format de description des données et non de leur représentation, comme c'est le cas avec HTML. La mise en page des données est assurée par un langage de mise en

page tiers. A l'heure actuelle (fin de l'année 2000) il existe trois solutions pour mettre en forme un document XML :

- CSS (Cascading StyleSheet), la solution la plus utilisée actuellement, étant donné qu'il s'agit d'un standard qui a déjà fait ses preuves avec HTML.
- XSL (eXtensible StyleSheet Language), un langage de feuilles de style extensible développé spécialement pour XML. Toutefois, ce nouveau langage n'est pas reconnu pour l'instant comme un standard officiel.
- XSLT (eXtensible StyleSheet Language Transformation). Il s'agit d'une recommandation W3C du 16 novembre 1999, permettant de transformer un document XML en document HTML.

B.3. Structure des documents XML

XML fournit un moyen de vérifier la syntaxe d'un document grâce aux DTD (Document Type Definition). Il s'agit d'un fichier décrivant la structure des documents y faisant référence grâce à un langage adapté. Ainsi un document XML doit suivre scrupuleusement les conventions de notation XML et peut éventuellement faire référence à une DTD décrivant l'imbrication des éléments possibles. Un document suivant les règles de XML est appelé document bien formé. Un document XML possédant une DTD et étant conforme à celle-ci est appelé document valide.

B.4. Décodage d'un document XML

XML permet donc de définir un format d'échange selon les besoins de l'utilisateur et offre des mécanismes pour vérifier la validité du document produit. Il est donc essentiel pour le receveur d'un document XML de pouvoir extraire les données du document. Cette opération est possible à l'aide d'un outil appelé analyseur (en anglais parser, parfois francisé en parseur).

Le parseur permet d'une part d'extraire les données d'un document XML (on parle d'analyse du document ou de parsing) ainsi que de vérifier éventuellement la validité du document.

B.5. Avantages de XML

Voici les principaux atouts de XML :

- La lisibilité : aucune connaissance ne doit théoriquement être nécessaire pour comprendre un contenu d'un document XML

- Autodescriptif et extensible
- Une structure arborescente : permettant de modéliser la majorité des problèmes informatiques
- Universalité et portabilité : les différents jeux de caractères sont pris en compte
- Déployable : il peut être facilement distribué par n'importe quels protocoles à même de transporter du texte, comme HTTP
- Intégrabilité : un document XML est utilisable par toute application pourvue d'un parser (c'est-à-dire un logiciel permettant d'analyser un code XML)
- Extensibilité : un document XML doit pouvoir être utilisable dans tous les domaines d'applications

Ainsi, XML est particulièrement adapté à l'échange de données et de documents. L'intérêt de disposer d'un format commun d'échange d'information dépend du contexte professionnel dans lequel les utilisateurs interviennent. C'est pourquoi, de nombreux formats de données issus de XML apparaissent (il en existe plus d'une centaine) :

- OFX : Open Financial eXchange pour les échanges d'informations dans le monde financier
- MathML : Mathematical Markup Language permet de représenter des formules mathématiques
- CML : Chemical Markup Language permet de décrire des composés chimiques
- SMIL : Synchronized Multimedia Integration Language permet de créer des présentations multimédia en synchronisant diverses sources : audio, vidéo, texte,...

Bibliographie

- [1] F.Gonzalez, SOA: « Comprendre l'approche orientée service », Extern ZDNet, Paris, 17 Février 2006.
- [2] ZHU Ning, «Entreprise Service Bus», Université JOSEPH FOURIER, Grenoble, 07 Novembre 2006.
- [3] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson, «Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WSAddressing, WS-BPEL, WS-Reliable Messaging, and More», Prentice Hall PTR, March 22, 2005.
- [4] David Chappell, Tyler Jewell, « Java Web Services», O'Reilly, First Edition March 2002.
- [5] Frédéric Bordage, «L'open source monte dans le bus d'entreprise», 01net, 30 juin 2005.
- [6] <http://mule.codehaus.org/display/MULE/Home>.