N° d'ordre : 07/2011- M/Inf

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMEDIAN
FACULTE ELECTRONIQUE ET INFORMATIQUE

Mémoire

Présenté pour l'obtention du diplôme de MAGISTER

EN : INFORMATIQUE

Spécialité : Systèmes Intelligents et Ingénierie du Logiciel

Par : Haniche Fayçal

SUJET

Une contribution à l'agilité du système d'information de l'entreprise par intégration des legacy systems dans une architecture SOA

Soutenu publiquement, le **06/04/2011,** devant le jury composé de :

Mr Guessoum Ahmed	Maitre de Conférences/A, à l'USTHB	Président
Mme Drias Habiba	Professeur à l'USTHB	Directrice de Mémoire
Mme Mellah Hakima	Chargée de recherche à CERIST	Codirecteur de thèse
Mr Azzoune hamid	Maitre de Conférence/A, à l'USTHB	Examinateur
Mr Aklouf Youcef	Maitre de Conférence/A, à l'USTHB	Examinateur

باسم الله, الحمد لله, و الصلاة والسلام على محمد رسول الله

يسعدني أن أهدي هذا العمل المتواضع إلى كل أفراد عائلتي, أخص بالذكر الوالدين الكريمين و الزوجة العزيزة

À ma famille

Remerciements

Je voudrais tout d'abord exprimer ma gratitude et ma profonde reconnaissance à mon directeur de thèse Pr. DRIAS Habiba, pour m'avoir permis d'effectuer ce travail sans aucune contrainte et encouragé sans relâche et avec patience à tirer le meilleur de moi même.

Je voudrais également exprimer mes remerciements sincères à Mme MELLAH Hakima, Chargée de recherche à CERIST, pour m'avoir proposé et encadré ce travail. Ses remarques de fonds, ses conseils et ses encouragements qui m'ont poussé à accéder au monde passionnant de la recherche scientifique.

Mes vifs remerciements vont aussi à Messieurs les membres du Jury qui m'ont fait l'honneur de lire et d'évaluer ce modeste travail présenté dans cette thèse.

Je tiens aussi à remercier mon père pour son aide à la correction orthographique de cette thèse.

Résumé

Les architectures orientées services (SOA) représentent, de nos jours, le moyen des Technologies de l'Information (IT) qui contribue à l'agilité du système d'information (SI) de l'entreprise. Leur principal fondement étant de présenter les fonctionnalités des applications comme un ensemble de services alignés avec les fonctions du métier de l'entreprise. Cette présentation est prometteuse dans le sens où elle a tendance à diminuer les coûts d'intégration, donner plus d'importance à la réutilisation des fonctionnalités du système d'information de l'entreprise, et bien sûr faciliter son adaptation aux changements imprévus durant le cycle de vie de son SI. Comme, dans les entreprises, il existe un potentiel important d'applications traditionnelles (*legacy systems*), qu'il est impossible de négliger, il est intéressant de penser à exporter ces applications vers les nouvelles architectures orientées services par la transformation de leurs fonctionnalités sous forme de services capables d'être intégrés dans une SOA.

Dans ce travail nous présentons une approche multi-agents pour l'encapsulation des fonctionnalités des *legacy systems* et plus particulièrement celles qui sont basées sur la technologie CORBA (Common Object Request Broker Architecture). Cette encapsulation a pour objectif principal l'intégration de ce type d'application dans une SOA. Nous avons généré une interface avec JADE (Java Agent DEvelopment framework), laquelle permet la génération automatique du code des clients CORBA et des ontologies utilisées par le WSIG **(**Web Service Integration Gateway). Le système proposé crée pour chaque fonctionnalité à encapsuler un agent représentant appelé *Agent service*, et permet la composition des fonctionnalités selon des modèles prédéfinis. Il utilise la passerelle WSIG pour publier les capacités des agents représentants comme services web qui seront utilisables dans un système à base de SOA.

Mots clés: Legacy system; Architecture orientée services, Encapsulation; system Multi-agents; Agilité.

Abstract

Nowadays, Service-Oriented Architectures (SOA) represents the means of Information Technologies (IT) which contribute to the agility of the enterprise information system (IS). Their main principle is to present the applications functionalities as services aligned with the company business. This presentation is promising in the sense that it tends to reduce the costs of integration, to give more importance to the reuse of information system functionalities in different business processes and, of course, to facilitate its adaptation to the unforeseen changes during the life cycle of the information system.

Since, in companies, there exists a significant number of earlier applications (*legacy systems*), that are commit to be neglect, it is interesting to think of exporting these applications to new SOA-based systems by transforming their functionalities into services that are able to be integrated in a SOA.

In this work, we present an approach based on multi-agent systems (MAS) for encapsulating the features of applications. We focus our interest particularly on legacy systems that are based on CORBA (Common Object Request Broker Architecture) technology.

The encapsulation main objective is to simplify the possibilities for integrating this kind of application in a service-oriented architecture (SOA). We design an interface using the Jade framework, which enables automatic generation of code for CORBA clients and ontologies used by the WSIG (Web Service Integration Gateway). The proposed system creates for each feature to wrap a representative agent, and allows the composition of functions according to predefined templates. The system uses the gateway WSIG to publish capabilities of representative agents as web services that will be used in a SOA-based system.

Keywords: Legacy system; Service-Oriented Architecture SOA, Encapsulation; Multi-Agents System; Agility.

Table des Matières

Liste des figures

Introduction générale

Introduction générale

Depuis quelques années, les entreprises n'étaient guère équipées d'un système d'information intégré. Elles endurent jusqu'à ce jour des systèmes mixtes et hétérogènes. Elles possèdent différentes applications développées sur différentes plateformes, utilisant différentes technologies et langages de programmation (Frank, et al., 2007). Ces applications sont sous différents modèles ou architectures

- Combinaison de monolithique, client/serveur, et application multi-tiers,
- Mélange de solutions procédurales, orientées objet et à base de composants,
- Mélange de langages de programmation,
- Différents types de SGBD (relationnel, objet, hiérarchique ….) et de produits
- Différentes solutions de middleware pour la communication (MOM : Message-Oriented Middleware, ORB : Object Request Broker, RPC : Remote Procedural Calls …etc.)
- Différents middleware de gestion de transactions et de sécurité,
- Différentes manières de partage de données, et
- Multiples protocoles et modèles de transmission d'information.

Un système traditionnel (Legacy System) est défini comme une vieille méthode, une technologie, un système informatique, ou un programme d'application qui continue à être employé, typiquement parce qu'il satisfait toujours les besoins des utilisateurs, alors que de nouvelles technologies ou des méthodes plus efficaces soient maintenant disponibles (http://en.wikipedia.org/wiki/Legacy_system). Une autre définition qui attire l'attention est donnée dans (Cetin, et al., 2007) qui considèrent tout logiciel implémenté avec une technique pré-SOA comme application legacy « *any software artifact that was built using pre-soa techniques is legacy* ».

Les systèmes legacy ne permettent pas une agilité des systèmes d'information de l'entreprise ce qui est du à la difficulté de s'adapter rapidement et à moindre coût avec le changement du métier. Autre problème, l'intégration des applications de l'entreprise : des solutions très utiles ont été proposées telles que MOM, ORB, RPC, EAI qui sont des technologies d'intégration très utilisées dans les applications traditionnelles, mais cette

intégration n'a pas pu surmonter les deux chausse–trappes en particulier le couplage technique et fonctionnel entre consommateur et fournisseur de services (M.Eveno, et al., 2007) :

- Le couplage technique impose au consommateur de connaître le protocole d'échange du fournisseur;
- Le couplage fonctionnel impose au client de connaître le format d'échange du fournisseur.

Ces deux pièges impliquent que toute évolution du fournisseur a un impact potentiel sur chacun des consommateurs, ce qui rend la maintenance pour l'adaptation très dure.
Les systèmes d'information doivent s'adapter rapidement en fonction des changements du métier et du marché. La solution est dans l'architecture orientée-service SOA qui se base sur le couplage lâche entre les services, donc une adaptabilité plus rapide et à moindre coût. Pour assurer l'agilité des SI et permettre une utilisation des services, de façon à satisfaire les processus métier et les utilisateurs, il est important d'avoir une architecture software orientée-service (SOA).

Problématique

Cette orientation dans le développement des systèmes est exigée de nos jours, car le monde, le métier et le marché changent rapidement. Mais est-ce que les entreprises doivent remplacer définitivement les systèmes existants dits *legacy systems* par des systèmes créés à zéro? La réponse est non pour plusieurs raisons :

- Ils sont le résultat d'un énorme investissement ;
- Ils enregistrent la connaissance, l'expertise, et les principes économiques qui peuvent ne pas être disponibles partout ailleurs que dans le code source
- Les coûts et les risques pour développer un système de rechange peuvent être inaccessibles et le développement en question prendra de toute façon plusieurs années avant que la fonctionnalité du code legacy soit remplacée de façon sûre et fiable par le nouveau système;

Ce qu'il faut faire est donc d'intégrer ces systèmes traditionnels dans le nouveau système à base de SOA en présentant leurs fonctionnalités sous forme de services interrogeables

et réutilisables en attendant que le nouveau système soit reconstruit conformément aux nouvelles normes.

Organisation du mémoire

La première partie de ce mémoire est un état de l'art à trois volets. Au chapitre I nous donnons une classification des architectures des systèmes logiciels traditionnels en présentant quelques limites concernant leur agilité. Le deuxième chapitre donne une description des systèmes d'informations agiles et une présentation de l'architecture orientée-services (SOA). Dans le troisième chapitre seront présentées les méthodes de modernisation permettant l'intégration des applications legacy dans une architecture orientée-services.

Nous présenterons notre contribution à ce sujet dans le chapitre quatre, puis, une expérimentation et réalisation pratique au chapitre cinq, en mettant en relief la contribution apportée au problème de l'intégration des legacy dans un système d'information

En fin nous conclurons notre travail.

Chapitre I
Caractéristique du potentiel informatique dans les entreprises traditionnelles

Chapitre I Caractéristiques du potentiel informatique dans les entreprises traditionnelles

Plusieurs types d'applications traditionnelles peuvent être rencontrés dans les entreprises ; nous citons les modèles les plus connus :

1. Les applications HM monoposte

Le plus simple type est l'application monoposte. Dans ce cas, toute l'application avec toutes ses ressources sont installées sur un seul ordinateur y compris le programme principal et la base de données. L'utilisateur de l'application peut bénéficier des fonctionnalités de l'application en utilisant une interface de conversation homme-machine.

Avantages

- Simple à implémenter
- Ne nécessite pas une plateforme complexe. Dans le pire des cas un SGBD (qui joue le rôle d'intermédiaire entre le programme et la base de donnée
- Exécution plus rapide.

Inconvénients

- Dans la plupart des cas, la modification d'une fonctionnalité nécessite la modification de toute l'application (dans le cas où le code source est inexistant, ce sera impossible)
- Difficile à intégrer dans des workflow automatisés car les fonctionnalités ne sont pas dispersées et ne peut être interrogé que via l'interface HM du programme principal.

2. Les applications client/serveur :

L'architecture **client/serveur** désigne un mode de communication entre plusieurs ordinateurs d'un réseau qui distingue un ou plusieurs postes clients du serveur : chaque logiciel client peut envoyer des requêtes à un serveur. Un serveur peut être spécialisé en

serveur d'applications, de base de données, de fichiers, de terminaux, ou encore de messagerie électronique.

Caractéristiques d'un serveur :

- Il est initialement passif (ou esclave, en attente d'une requête) ;
- Il est à l'écoute, prêt à répondre aux requêtes envoyées par des clients ;
- Dès qu'une requête lui parvient, il la traite et envoie une réponse.

Caractéristiques d'un client :

- Il est actif le premier ou maître dans le sens où il est l'initiateur de la communication avec le serveur ;
- Il envoie des requêtes au serveur ;
- Il attend et reçoit les réponses du serveur.

Le client et le serveur doivent bien sûr utiliser le même protocole de communication. Un serveur est généralement capable de servir plusieurs clients simultanément.

Avantages

- Toutes les données sont centralisées sur un seul serveur, ce qui simplifie les contrôles de sécurité, des mises à jour des données et des logiciels ;
- Les technologies supportant l'architecture client/serveur sont plus matures que les autres.

Inconvénients

- Si trop de clients veulent communiquer avec le serveur au même moment, ce dernier risque de ne pas supporter la charge (alors que les réseaux pair à pair fonctionnent mieux en ajoutant de nouveaux participants) ;
- Si le serveur n'est plus disponible, plus aucun des clients ne marche.
- Le client et le serveur sont en couplage dur qui implique que l'ajout d'un client n'est possible que par la connaissance des détails du programme serveur et des protocoles de communication utilisés.

Exemples

— La consultation de pages sur un site web fonctionne sur une architecture client/serveur. Un internaute connecté au réseau via son ordinateur et un navigateur web est le client. Le serveur est constitué par le ou les ordinateurs contenant les applications qui délivrent les pages demandées. Dans ce cas, c'est le protocole de communication HTTP qui est utilisé.

— Les courriers sont envoyés et reçus par des clients et gérés par un serveur de messagerie. Les protocoles utilisés sont le SMTP, et le POP ou l'IMAP.

— La gestion d'une base de données centralisée sur un serveur peut se faire à partir de plusieurs postes clients qui permettent de visualiser et saisir des données.

3. Applications Distribuées

Une application distribuée est un ensemble composé d'éléments reliés par un système de communication ; les éléments ont des fonctions de traitements (processeurs), de stockage (mémoire), de relation avec le monde extérieur (capteurs, actionneurs)

Les différents éléments du système ne fonctionnent pas indépendamment mais collaborent à une ou plusieurs taches communes (une partie au moins de l'état global de l'application est partagée entre plusieurs éléments, sinon, on aurait un fonctionnement indépendant)

Les applications distribuées sont conçues pour atteindre les propriétés souhaitées suivantes :

— Le système doit pouvoir fonctionner (au moins de façon dégradée) même en cas de défaillance de certains de ses éléments

— Le système doit pouvoir résister à des perturbations du système de communication (perte de messages, déconnexion temporaire, performances dégradées)

— Le système doit pouvoir résister à des attaques contre sa sécurité (tentatives de violation de la confidentialité et de l'intégrité, usage indu de ressources, déni de service)

— Le système doit pouvoir facilement s'adapter pour réagir à des changements d'environnement ou de conditions d'utilisation

— Le système doit préserver ses performances lorsque sa taille croit (nombre d'éléments, nombre d utilisateurs, étendue géographique)

4. Les solutions d'intégration des applications de l'entreprise

La problématique d'intégration des applications de l'entreprise peut être synthétisée en deux questions essentielles :

— Comment assurer la consistance et la propagation des données entre plusieurs sous-systèmes ?

— Comment déclencher, depuis un sous-système donné, un traitement dans un sous-système tiers ?

Pour répondre à ces défis, des solutions dites **Middleware** ont été proposées destinées à assurer la communication « temps réel » entre systèmes hétérogènes.

Les middlewares orientés messages :

Apparus dès le début des années 80, les MOM (Message-Oriented Middleware) ont une sémantique asynchrone : le client construit un message et le transmet au middleware, qui se charge de le router vers le ou les systèmes cibles. La communication est scindée en deux, évitant le couplage technique des participants. La garantie de livraison du message est confiée au MOM.

Pendant longtemps, cependant, les MOM sont restés des solutions largement propriétaires, forçant chaque partie à connaître les modalités d'interfaçage avec le broker, et limitant la capacité d'intégration aux environnements et langages supportés par l'éditeur de la solution

Les Object Request Brokers (ORB)

CORBA est une spécification adoptée par l'Object Management Group (OMG), proposé dès le début des années 90, une approche plus riche et plus ambitieuse dans le sens où elle supporte virtuellement tous les langages et enrichit l'architecture d'intégration de services techniques de haut niveau, en particulier la localisation, les transactions et la sécurité. CORBA propose une sémantique d'invocation point-à-point synchrone ou asynchrone, utilisant un protocole et un encodage standardisés ; CosNotification, le service de messaging CORBA, spécifie un MOM interopérable.

Les EAIs (Entreprise Application Integration) :

Malgré leurs qualités respectives, MOMs et ORBs restent des solutions très techniques ; ils permettent certes à la fois la propagation des données et l'intégration des traitements, mais la sémantique des échanges y reste fondamentalement point-à-point : le client doit connaître le format du message qu'il adresse aux systèmes tiers. Ce couplage fonctionnel des systèmes devient rapidement un cauchemar pour la maintenance et l'exploitation, en particulier s'il est étendu à l'ensemble du SI.

Pour faire face à ces enjeux, une nouvelle catégorie de solutions middleware est apparue : les EAI (Enterprise Application Integration). Tirant parti des deux précédentes approches, les EAI proposent une architecture hub and spoke – à opposer à l'architecture centrées Réseaux des ORBs et MOMs -, dans laquelle un composant central assure la médiation physique entre le client et sa cible. Ce composant central prend à son compte l'ensemble des problématiques techniques de bas niveau (localisation, disponibilité, cache, communication, transcodage, interopérabilité au moyen de connecteurs spécialisés, audit, traces, sécurité voire transactions) (M.Eveno, et al., 2007).

5. Conclusion

Malgré le progrès connu dans le développement des applications de l'entreprise et dans le développement des plateformes d'intégration, ces solutions souffrent de leur caractère très propriétaire :

- Le protocole utilisé pour les échanges et le transport des messages est propriétaire.
- La technologie interne est propriétaire. Ainsi, l'accès aux applications se fait par l'intermédiaire de connecteurs encore largement spécifiques à chaque éditeur malgré des tentatives de standardisation
- Les formats et encodages de données utilisés sont propriétaires.

Les applications informatiques forment une brique essentielle dans les systèmes d'information de l'entreprise, elles ne doivent pas former un frein contre l'agilité de ce

dernier, elles doivent être intégrées, flexibles et adaptables aux changements organisationnel et environnemental et aux changements du métier.

Chapitre II
Système d'Information Agile et Architecture Orientée Services

Chapitre II Système d'Information Agile et Architecture Orientée Services

1. Système d'Information Agile

Les entreprises sont aujourd'hui confrontées à un environnement de plus en plus complexe où les exigences de rapidité et d'efficacité conditionnent leur compétitivité.
Une entreprise agile est une entreprise dotée d'un système d'information synchronisé avec son business. Gartner définit l'agilité de l'entreprise comme la capacité d'une organisation de sentir le changement environnemental et de répondre efficacement à ce changement (McCoy, et al., 2006). Les évolutions de la stratégie des entreprises, parfois accompagnées de modifications de leur périmètre, et la complexité croissante de leurs métiers imposent souvent de piloter des changements importants et rapides du système d'information. Au sein de ce système d'information, les transformations touchent aussi bien le système informatique et les technologies mises en place que les applications, les processus et l'organisation. L'ensemble influence la stratégie de l'entreprise.
Tous ces éléments doivent néanmoins s'articuler de manière intime tout en pouvant unitairement être modifiés mais sans toutefois devoir défaire les autres. L'agilité du système d'information devient l'objectif principal de toute direction des systèmes d'information (Cigref, 2003)**.**

Un système d'information agile est un système qui est capable de s'adapter aux évolutions fonctionnelles et techniques, qui est plus ouvert afin d'absorber les processus de gestion avec les tiers, qui permettra l'évolution de l'organisation sans remise en cause des applicatifs métiers. Ce système d'information d'un nouveau genre devra être capable de se recycler au cours du temps, il sera prévu pour se reconfigurer proprement sans générer de scories applicatives. C'est un système d'information durable.

2. Architecture orientée services, clef de l'agilité

2.1. Introduction

SOA, Acronyme de Service Oriented Architecture, est traduit en français par Architecture Orientée Services. Le SOA formalise le concept d'échange et de partage inter-application dans une logique proche de l'EAI. Le SOA, terme suggéré par le Gartner[1] Group dès 1996, propose de définir les échanges en terme de services (http://www.piloter.org). Elle été définie pour répondre à des besoins de réutilisation et d'adaptabilité.

Une architecture orientée services est un style d'architecture fondée sur la description de services et de leurs interactions. Les caractéristiques principales d'une architecture orientée services sont le couplage faible, l'indépendance par rapport aux aspects technologiques et l'extensibilité (Bley-Fornarino et al, 2007).

La propriété de **couplage faible** implique qu'un service n'appelle pas directement un autre service; les interactions sont gérées par une fonction d'orchestration. Il est donc plus facile de réutiliser un service puisqu'il n'est pas directement lié aux autres.

L'indépendance par rapport aux aspects technologiques est obtenue grâce aux contrats d'utilisation qui sont indépendants de la plate-forme technique utilisée par le fournisseur du service.

Enfin, **l'extensibilité** est rendue possible par le fait que de nouveaux services peuvent être découverts et invoqués à l'exécution

[1] **Gartner**, Inc., fondée en 1979, est une entreprise américaine de conseil et de recherche dans le domaine des techniques avancées. Ayant environ 10 000 clients, elle mène des recherches, fournit des services de consultation, tient à jour différentes statistiques et maintient un service de nouvelles spécialisées.(http://fr.wikipedia.org/wiki/Gartner)

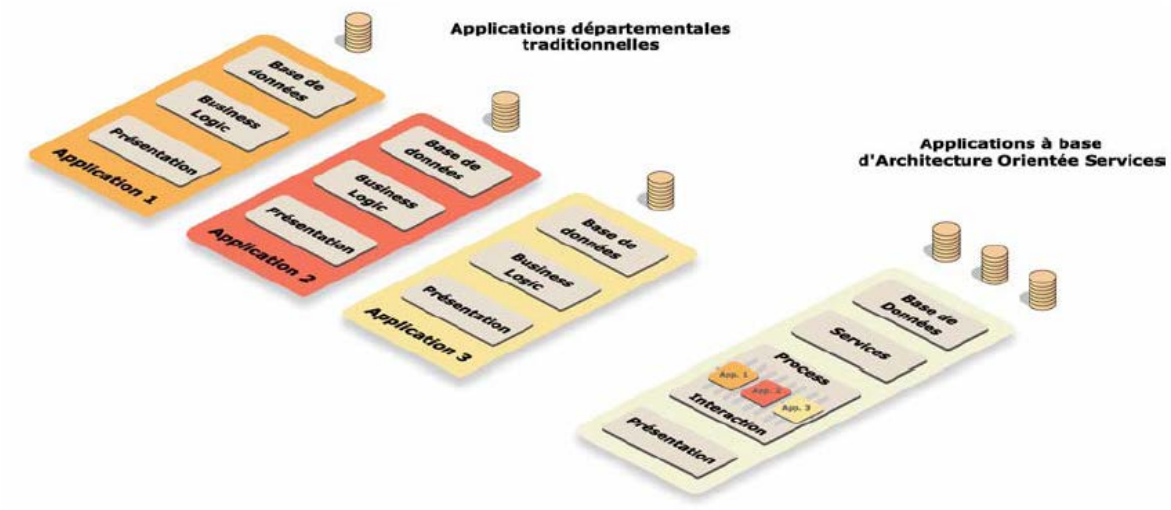


Fig II.1 Application traditionnelle contre application à base de SOA (Audrey, 2006)

L'application à base de l'architecture SOA n'est plus une application départementale traditionnelle **(**Fig II.1**)** mais un ensemble de services indépendants les un des autre plus des orchestrations pour la réalisation des processus métiers transversales plus des outils de gouvernance et de médiation.

2.2. Concepts

2.2.1. Notion de service

Dans une architecture SOA, la notion de service fait référence à une fonction encapsulée dans un composant interrogeable à l'aide d'une requête paramétrée et fournissant une ou plusieurs réponses. Idéalement chaque service doit être indépendant des autres afin de garantir sa réutilisation et son interopérabilité. Les processus sont constitués par un assemblage de services.

L'implémentation de nouveaux processus et de services s'effectue en respectant certains standards, dits ouverts, adoptés par tous les acteurs d'un même écosystème.

Le service est l'unité atomique d'une architecture SOA. Une application est un ensemble de services qui dialoguent entre eux par des messages. (http://fr.wikipedia.org.wiki/)

Le couplage entre services est un couplage lâche et les communications peuvent être synchrones ou asynchrones.

Un service est une entité de traitement qui respecte les caractéristiques suivantes (Kanchanavally, et al., 2005) :

Résout un problème : Un service est une pièce de code qui résout un problème métier. Par cette définition, chaque pièce de code dans une application est un service (par exemple: le code qui accède à une base de données pour enregistrement ou extraction de données, le code qui imprime un rapport, qui produise des données pour un rapport, et ainsi de suite). Cependant, toutes les autres caractéristiques doivent être considérées.

Indépendant de la technologie d'implémentation: Ceci signifie que le service peut être écrit en n'importe quel langage de programmation et qu'il peut être invoqué par un autre service écrit dans un autre langage de programmation.

Indépendant du protocole de transmission : Un service est également indépendant des protocoles de transmission de requêtes ou de résultats. Ceci signifie que le service peut être appelé en utilisant n'importe quel protocole de transmission (HTTP, TCP/IP, RMI, et ainsi de suite). Le point d'interaction entre un consommateur de service et le service lui même est les message SOAP. Aujourd'hui, le protocole primaire de transport des données entre les services est le HTTP.

Adressable en réseau : Ceci signifie qu'un service peut être appelé à travers un réseau Intranet ou Extranet. Par cette définition, n'importe quel EJB (Enterprise Java Beans) dans votre système est un service, et n'importe quel objet de COM/CORBA est un service. Cependant, il est important de comprendre qu'il doit avoir toutes les caractéristiques décrites dans cette définition pour être un service. Par exemple, s'il est adressable en réseau mais pas Indépendant d'une technologie, il n'est pas un service. Ceci élimine les objets CORBA et les EJBs d'être des services.

Décrit par une Interface publiée : Un service a une interface publiée. Ce qui signifie que le service doit être décrit par une interface accessible en réseau. Cette Interface permet aux services client d'extraire le format d'échange et les points d'entrée au service.

Est transparent à son endroit : Un service est transparent à son endroit. Ceci signifie que le client ne devrait pas savoir quand, où, ou comment le service est établi. Tout ce que le client a besoin de faire est d'appeler le service en se basant sur son interface publiée.

2.2.2 Processus métier :

Un processus métier est un ensemble d'activités incluant une interaction entre des participants sous la forme d'échange d'informations. Les participants peuvent être :

- Des applications ou des services du SI,
- Des acteurs humains
- D'autres processus métiers.

Dans l'Architecture Orientée Services, la modélisation des processus métiers consiste à créer de nouveaux services qui modélisent des opérations métiers complexes à partir des services simples.

La modélisation des processus d'affaires offre la capacité d'orchestrer des interactions de système complexes et constitue en elle-même un service capable de communiquer et d'échanger des données (Groupe CGI inc. 2006)..

2.2.3. Couplage lâche entre processus et services

Au cœur du principe des architectures orientées services, apparaît la notion de couplage lâche (Syntec informatique, 2007).

- Les protocoles de communication sont standards et universel,
- Les services impliqués dans un processus métier peuvent être sélectionné et mis en interaction de manière dynamique grâce à l'interface publié du service

2.3. Les couches SOA

L'architecture orienté service SOA est une architecture conceptuelle où la logique de l'application ou les fonctionnalités du métier sont mises à la disposition des utilisateurs et des consommateurs en tant que services partagés et réutilisables dans le réseau IT.

Dans cette architecture les couches suivantes sont envisagées Fig II.2:

Consumer : ce sont les consommateurs des offres du système (Service et processus), ça peut être une application externe ou interne, un service, un portail ...etc.

Les Processus Métiers : ce sont les processus métiers qui définissent les fonctionnalités du système, ce sont des orchestrations de services.

Services : ce sont les descriptions des services disponibles ;

Les composants des Services: ce sont les implémentations des différents services, et peuvent être dans n'importe quels langage ou technique de programmation

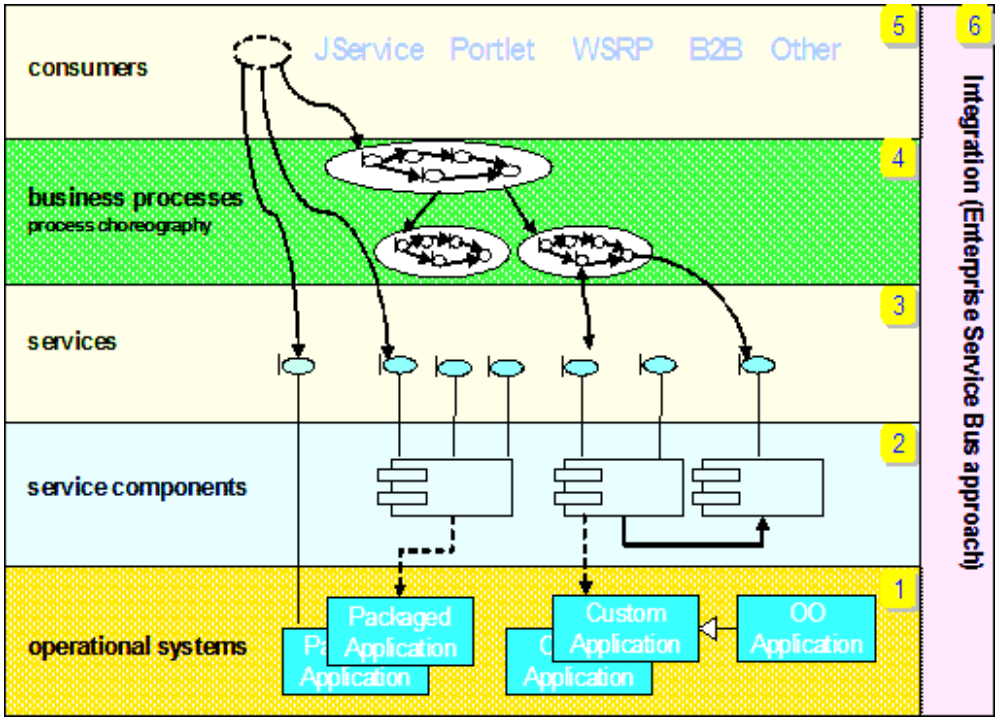


Fig II.2 les Couches d'une architecture SOA (Audrey, 2006)

Les Systèmes Opérationnels: ce sont les systèmes installés (autre applications, des SGBD etc.) dans les différents sites distribués.

Intégration : joue le rôle de médiateur entre les différentes couches.

2.4. Standards de l'architecture

La réalisation des systèmes à base d'architecture SOA n'est pas lié d'origine à une technologie spécifique, mais pour le moment la technologie des Service Web (SW) vient de s'imposer pour devenir la technologie leader dans la mise en œuvre de ce type de systèmes.

Les services Web (voir Annexe A) sont considérés comme la meilleur technologie supportant les principes et les spécifications de l'architecture SOA et par conséquence ses standards sont devenus les meilleurs standards de l'architecture SOA:

- ✓ Le WSDL (Web Services Description Language) : est un Langage XML de description de services web
- ✓ L'UDDI (Universal Description Discovery and Integration) : est un annuaire de services fondé sur XML particulièrement destiné aux services Web.
- ✓ SOAP (Simple Object Access Protocol) : est un protocole de communication basé sur XML permettant aux services de s'échanger des informations via http ou smtp.
- ✓ BPEL ou WS-BPEL (Web Services Business Process Execution Language) : est un langage à base de XML qui définit les règles de dialogue entre les services. Il permet l'orchestration des services web.

3. Conclusion

Plusieurs raisons ont amené à dire que l'architecture orientée services est la clef de l'agilité des systèmes d'information :

- Une modularité permettant de remplacer facilement un composant (service) par un autre
- Une réutilisabilité possible des composants (par opposition à un système tout-en-un fait sur mesure pour une organisation).
- De meilleures possibilités d'évolution (il suffit de faire évoluer un service ou d'ajouter un nouveau service)
- Une plus grande tolérance aux pannes
- Une maintenance facile

L'Architecture Orientée Services est aussi considérée comme solution pour intégrer des applications diverses et hétérogènes, développées sur différents architectures et langages de programmation, et sur différentes plateformes

Chapitre III
Les Méthodes de modernisation des Legacy Systems

Chapitre III Les Méthodes de modernisation des Legacy systems

1. Introduction

Un système traditionnel (Legacy System) est défini comme une vieille méthode, une technologie, un système informatique, ou un programme d'application qui continue à être employé, typiquement parce qu'il satisfait toujours les besoins des utilisateurs, alors que de nouvelles technologies ou des méthodes plus efficaces soient maintenant disponibles (http://en.wikipedia.org/wiki/Legacy_system).

Comment profiter des avantages de l´architecture SOA lorsque les applications legacy existantes semblent être un obstacle ? Certains pensent d´abord à remplacer les applications existantes mais il ne s´agit pas de la seule option.

Utiliser une méthode ou approche pour intégrer un tel système ou application dans une Architecture Orientée-Service (SOA) n'est pas nouveau mais plusieurs travaux ont été réalisés dans ce domaine. Certaines approche sont concentrées sur le développement des outils d'analyse qui permettent de mieux comprendre le code source et de faire sortir les règles métier incluent dans le code source du système legacy. D'autres se concentrent sur la proposition de techniques qui permettent de présenter les fonctionnalités du legacy sous forme de services invocables et réutilisables (bien sûr ces solutions ne sont pas généralisées pour toutes les formes ou modèles d'applications legacy). Une autre classe de réalisation, consiste au développement d'outil d'aide pour faciliter la tâche d'intégration aux développeurs.

2. Modernisation des Legacy systems:

Le maintient et l'évolution des systèmes Legacy est devenu un des plus évidents obstacles pour les départements de technologie de l'information (TI) de plusieurs organismes. Cependant en dépit du haut profil du problème dans toute l'industrie, aucune solution effective n'a été mise en pratique à une grande échelle pour éviter le barrage de route des Legacy.

Selon Erradi (A.Erradi, et al., 2006) Les méthodes d'intégration des systèmes legacy dans une architecture SOA peuvent être divisées en deux catégories **(**Fig III.1**).** Les méthodes envahissantes nommées chez certains auteurs " white-box strategy " et les

méthodes non envahissantes nommées aussi " black-box strategy " (Zhuopeng.Zhang, et al., 2004), (Willem-Jan, 2007). Il est à noter que la plupart des réalisations dans ce domaine tentent à faire migrer les legacy sous forme de services web. Les auteurs dans (Grace, et al., 2006) indiquent que la migration comme elle peut être facile, elle peut être également compliquée et non automatique dans d'autres cas, selon les caractéristiques du legacy, et qu'une analyse doit être faite avant de procéder à la modernisation. Elle doit considérer :

- les exigences des utilisateurs du service : il est important de connaitre quelle application utilise le service et comment (exemple : quelle information et sous quel format),
- Les caractéristiques techniques de l'environnement cible : protocole de liaison, techniques de messagerie, protocole de communication, etc.,
- L'architecture du système Legacy : il est critique d'identifier les éléments architecturaux qui peuvent être problématiques dans l'environnement cible ou qui peuvent augmenter la difficulté des efforts de migration (exemple : la dépendance dans les produits commerciaux, un système d'exploitation spécifique, etc.),
- L'effort impliqué en écrivant l'interface du service : même si on s'attend à ce que le legacy demeure intact, il doit y avoir le code qui recevra les demandes des services client, les traduise en des appels au système legacy et qui produira les réponses à envoyer aux services demandeurs ; les outils pour générer ces codes peuvent ne pas exister pour un environnement particulier,
- L'effort dans la translation des types de données (dans le cas des types complexes ça sera pire),
- L'effort impliqué dans la description des services: pour un certain niveau de description on doit inclure des informations sur la qualité de service tel que: la performance, la rentabilité et la sécurité,
- L'effort impliqué dans l'écriture du code d'initialisation des services et des procédures opérationnelles,
- Estimation des coûts, des difficultés et des risques: les informations sur les points précédents permettant des estimations réelles.

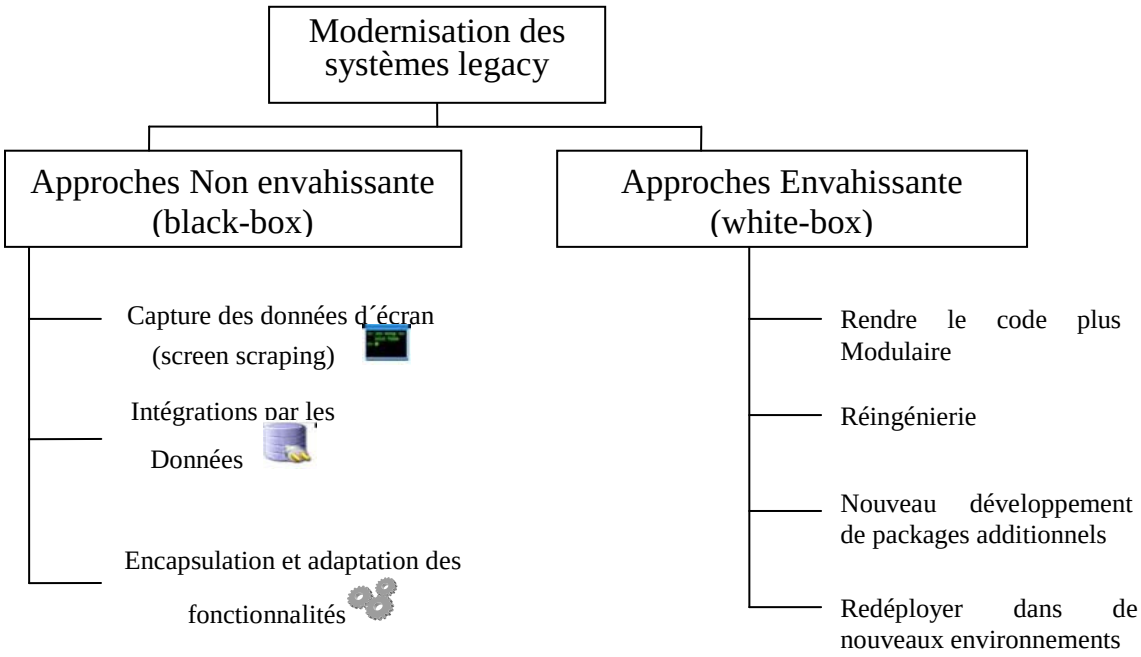


Fig III.1 Taxonomie des approches de modernisation des legacy

2.1 Les méthodes envahissantes (white-box strategy) :

Ces méthodes ont pour objectif de revitaliser et d'améliorer le système traditionnel pour soulager l'entretien et les prolongements par la reconception envahissante. Elles impliquent une analyse profonde et détaillée du code de base existant, une compréhension de la fonctionnalité du système et de l'architecture des données. Elles impliquent entre autres l'extraction et la rationalisation des données et des règles du métier. Ceci est suivi d'un processus de reproduction, de consolidation, de reformulation en code plus modulaire et de reconception.

C'est l'option la plus dure, la plus coûteuse et risquée, mais assez agile pour répondre aux changements des exigences du métier. Les issues à considérer incluent (Sucharov, 2007) :

- Grand coût de développement payé d'avance;
- Difficile d'acquérir le comportement du système Legacy;
- Risque de créer un système Legacy de deuxième génération;
- Possibilité de créer un nouveau système plus agile.

Avant de présenter quelques travaux sur les méthodes envahissantes il est intéressant de citer quelques outils d'analyse de code source utilisés avant l'étape de transformation ou de reconception Ce sont des outils dits d'ingénierie inverse à l'aide desquels les règles de base de l'ingénierie adoptée à la conception initiale du legacy, seront extraites à partir du code source:

- **"DOxygen"** peut extraire des données à partir du code C++, telles que les classes, les attributs, les dépendances, et les commentaires (http://www.dxoygen.org)
- **"Understand for C++"** : c'est un outil qui permet l'analyse du code C existant afin de l'évaluer ; il retourne en sortie plusieurs résultats (Grace, et al., 2006) :
    - Un dictionnaire de données
    - Des Métriques au projet, aux fichiers, aux classes, et au niveau des fonctions
    - Un arbre d'invocation
    - Une liste de fonctions et d'objets inutilisés
- **"ARMIN" (Architecture Reconstruction)** (L.O'Brien, et al., 2002) il permet la reconstruction de l'architecture d'un système en application à partir du code existant. Le but est d'aborder la question de dépendance fonctionnelle (entre classes, procédures et données) avec plus de détails.
- **"Perl"** il permet d'extraire les dépendances fonctionnelles entre les parties de code d'une application COBOL ainsi que les dépendances entre les données (Joris, et al., 2008)
- **SMART** (Service-Oriented Migration and Reuse Technique) (G.Lewis, et al., 2005) : c'est une approche qui permet de recueillir les informations nécessaires et d'identifier les risques pour l'effort de migration d'une manière plus systématique, en se basant sur un ensemble d'inputs (buts et objectifs, les besoins de l'utilisateur final, des informations sur le système legacy, tel que l'architecture, le code, les coûts et l'historique de l'ingénierie adopté) et des informations sur la SOA cible telles que les contraintes et les besoins, l'architecture, les standards et les services communs ; en procédant à certaines activités pour enfin aboutir à un ensemble d'outputs (liste des caractéristiques, liste des problèmes de migration, tableau des composants , tableau de services, description de *SOA*, tableau

d'options de services composants, tableau e solutions de migration possibles , stratégie de migration des services).

Dans (Y.Huang, et al., 2003) a été proposé une méthode semi automatique pour la conversion des routines écrites dans le langage de programmation *C* en leurs équivalentes en *JAVA*, ils ont utilisé deux outils : l'outil *JACAW (*Java-C Automatic Wrapper*)* qui permet d'envelopper des routines *C* Comme *code JAVA* , et l'outil *MEDLI* (MEdiation of Data and Legacy code Interface) qui permet d'envelopper un code java pour être conforme à un modèle de composant de *Triana* (*Triana* est une interface de composition des workflow pour des applications réseau distribuées basées sur *Java*) et pour utiliser les types de données de Triana pour leurs entrées\sorties. *MEDLI* peut être appliqué sur les routines *C* enveloppées produites par *JACAW* et ainsi permet au code *C* du legacy d'être accessible en tant que Composant indigène de *Triana*. Cette réalisation en plus du fait qu'elle ne concerne que du code legacy C, présente plusieurs limites ; nous en citons quelques unes :*i*- les erreurs dans les bibliothèques Legacy ne seront pas corrigées durant la transformation et peuvent entrainer des erreurs d'exécution. *ii*- la transformation automatique ne donne pas toujours des résultats justes, donc une solution non sûre. *iii*- Cette solution n'est valable que pour du code modulé.

Dans (Grace, et al., 2006) une stratégie de migration a été adoptée pour la migration de composants legacy vert une SOA, le cas traité est un système constitué de 8.000.000 lignes de code C++ avec 2500 classes, et dépendant d'une base de données commerciales. Les auteurs utilisent les deux outils "Understand c++" et "ARMIN" pour extraire les règles et l'architecture du code du système legacy puis suivant l'approche SMART, ils identifient la stratégie de migration à adopter. Ils ont pris en considération les caractéristiques du système SOA qui est en cours de développement. Parmi les contraintes essentielles qu'ils ont pris en considération est que les données dans toute la SOA devront être centralisées et l'accès aux données par les services devra être réalisé par l'invocation des services de magasin de données, chose qui a impliqué la restructuration des bases de données du legacy et de les migrer vers la base commune. Cette stratégie se base sur le fait que le code legacy est à base de composants, chose qui facilite un peut la tache de migration. Parmi les limites est que le code des langages procéduraux n'est pas impliqué, et que l'intervention manuelle est grande dans la transformation des composants et dans la décision sur le choix des composants à migrer.

Dans le travail présenté par He Guo (He, et al., 2005)un outil d'aide nommé WSW (Web Services Wrapper) pour l'analyse et l'emballage des applications client/serveur exécutées dans une plateforme .Net ; il permet de générer les codes sources des services web selon les propriétés des services web et les méthodes données par le développeur en se basant sur le code source de l'application legacy. L'analyseur permet de visualiser la structure globale du projet pour visualiser les différents fichiers et classes ; ainsi une analyse détaillée du code source permet de visionner les méthodes incluses dans les classes et les relations entres elles ; ce qui permet aux développeurs de décider quelle méthode à envelopper ou à regénérer ; vient après l'étape de génération du code du service web et à ce point-là une opération de validation des méthodes est effectuée permettant de vérifier si la méthode respecte les restrictions (les restrictions de validation sont au cœur du wrapper, elles sont proposées car certaines méthodes ne peuvent être converties en services web par exemple : la méthode abstraite ne peut être enveloppée par un service). L'outil est utile mais possède certaines limites ; il est destiné aux applications programmées en VB.Net, et dont le code source est disponible.

Dans (Zhuopeng.Zhang, et al., 2004)**,** une approche de réingénierie a été proposée qui applique un algorithme de groupement (de clusturing) hiérarchique développé pour restructurer le code du legacy et pour faciliter l'extraction de code legacy pour la construction de services web. Il permet l'identification et l'empaquetage de services. Il se base sur l'idée qu'un code source structuré en Orienté-objet est facile à être transformé en services ; donc leur approche consiste en un premier temps à diviser le système legacy en sous système puis de faire une analyse sur les procédures, les fonctions et sur les relations entre elles pour enfin les transformer sous un modèle objet, parallèlement une analyse du domaine d'application est faite pour extraire les modèles de service du domaine spécifique. Ces derniers sont alors mis en correspondance avec le modèle OO déduit à partir du code legacy pour construire les nouveaux composants orientés services. La méthode implique une grande intervention manuelle.

Dans (Harr, et al., 2008) une génération des services web a été adoptée en suivant trois étapes principales :

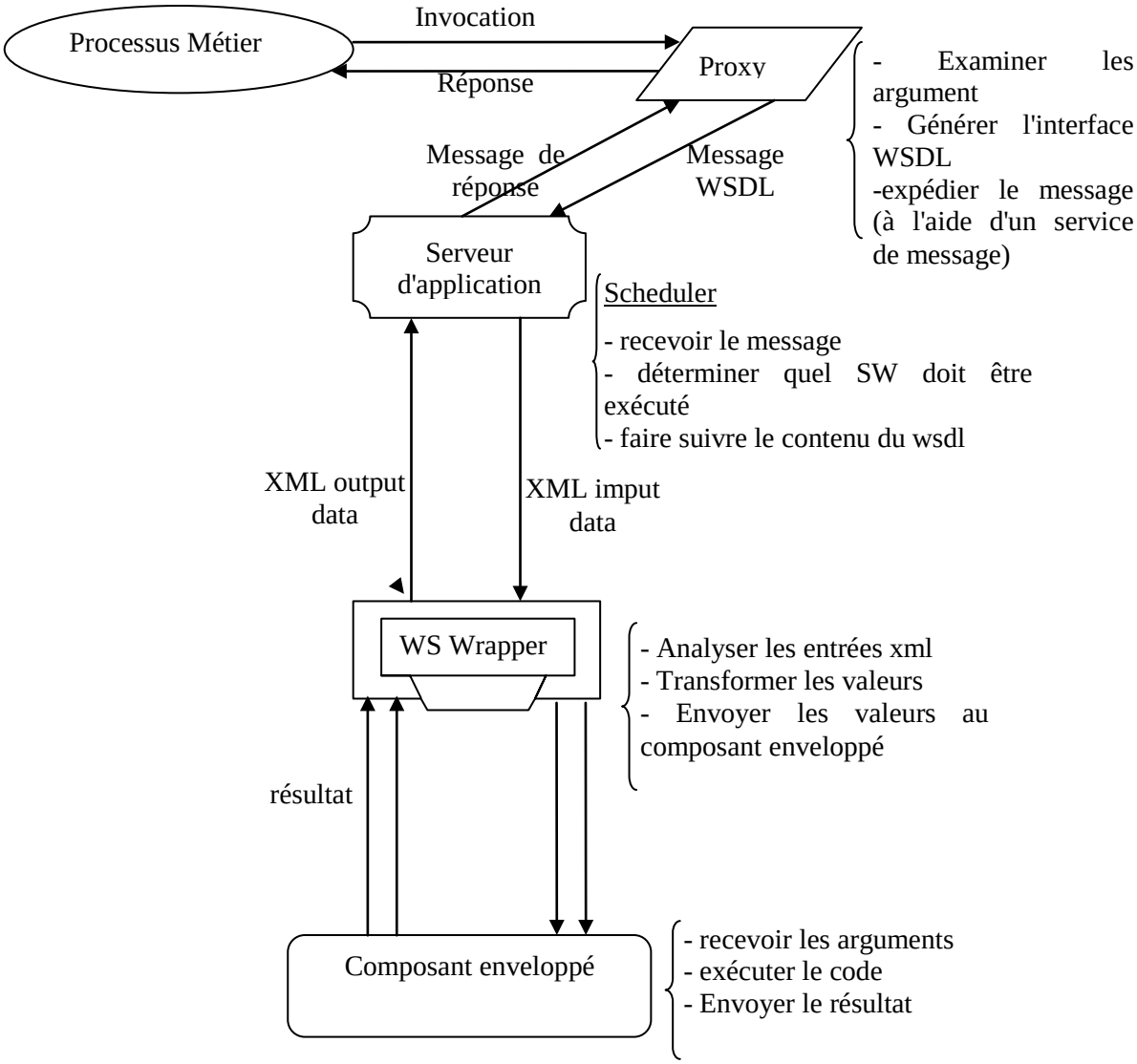


Fig III.2 Enchaînement du web service au processus métier

- Récupération des parties du code qui vont être transférées en services, bien sûr après une analyse du code source en utilisant une technique appelée "**code stripping**"
- Emballage du code récupéré : affecter aux composants extraits du code legacy une interface WSDL en utilisant des outils automatiques tels que **SoftWrap.** Cette étape s'accompagne par l'implémentation de deux modules supplémentaires, un pour analyser les messages d'entrées, extraire les données et les assigner aux

arguments correspondants dans le composant enveloppé ; l'autre module est utilisé pour créer les messages de retour à partir des résultats d'exécution. Ces deux sous programmes agissent en tant que pont entre l'interface WSDL et l'interface d'appel du code original

- Enfin, la dernière partie définit l'architecture adoptée pour l'enchaînement des services web développés au processus métier **(**Fig III.2**)**

Cette méthode apparaît simple mais exige une bonne compréhension du code legacy.

2.2. Méthodes d'intégration non envahissantes (black-box strategy):

A l'inverse des stratégies white-box, les stratégies black-box n'ont pas d'intervention sur le code legacy, ni sur leur environnement d'exécution ; elles laissent la configuration du système original intacte, ainsi que son environnement d'exécution**.** Le but de ces approches est de tendre le temps de vie des applications traditionnelles par l'exposition de leurs fonctionnalités telles quelles à l'aide de canaux additionnels jouant le rôle d'interface moderne pour permettre l'exploitation de ces fonctionnalités.

Les stratégies black-box exigent moins d'architecture et de conception payée d'avance, elles permettent de réduire les coûts d'intégration et d'élaborer des plans pour le transfert des composantes des applications traditionnelles en de nouvelles plates formes. Cependant elles peuvent seulement fournir une solution tactique à court terme car elles adressent seulement les points de conflit d'intégration et de flexibilité et pourraient même s'ajouter à la complexité du système et aux coûts de maintenance.

2.2.1 Capture des données d´écran « screen scraping »

Les écrans mainframe (aussi appelés « écrans verts ») constituent le point d´accès existant le plus largement disponible (Forrester Research Inc;, 2007)**.**

Dans (Willem-Jan, 2007) le *screen scraping* consiste à développer de nouveaux écrans associés aux anciens dans le but d'avoir une bonne lecture de ces derniers et permet de déployer l'application sous un nouveau environnement, l'outil qui a été utilisé est Seagull's WinJa, dans cette réalisation l'application VA-1 est redéployée dans un environnement Windows, alors qu'elle continue à s'exécuter dans son environnement original MVS (Multiple Virtual Storage) du système IBM OS-390 Z

Les premières solutions de captures des données d´écran étaient réputées peu fiables et exigeantes en matière d´entretien. En revanche, les outils actuels sont bien plus performants.

2.2.2. Intégrations par les données :

Le principe est qu'on peut bénéficier des fonctionnalités d'un système par l'accès aux données de la base de données du système qui représentent le résultat du traitement de la fonctionnalité, une réalisation de cette technique est faite par Gilles Laborderie (Gilles, 2008) où il a pris l'exemple d'un progiciel de comptabilité dont il souhaite exposer le référentiel de tiers au reste du SI. Il s'agit classiquement d'une application client/serveur déployée sur un ou plusieurs postes de travail avec en général une base de données hébergée sur un serveur de l'entreprise. Il a annoncé qu'on n'a probablement pas la main sur l'application cliente qui est livrée sous forme de binaire par l'éditeur ou l'intégrateur et qui a souvent comme seul moyen de communication un rudimentaire système d'import et d'export de fichiers actionnés par l'interface utilisateur. Il n'est donc pas possible (selon lui) d'intégré une autre application avec le client de logiciel de comptabilité (Gilles, 2008).

Et comme solution pour l'intégration d'une telle application il a proposé d'interposer entre la base de données et les applications clientes un artefact applicatif en façade des tables de la base de données. Chez OCTO, on donne souvent le nom d'**émissaire** à ce type d'application. La métaphore consistant à dire que lorsqu'une application du SI a besoin d'échanger avec une autre application, elle doit s'adresser à l'émissaire de cette dernière. Autrement dit, l'émissaire expose sous forme de services les fonctions / données de l'application qu'elle représente **(**Fig III.3**)**.

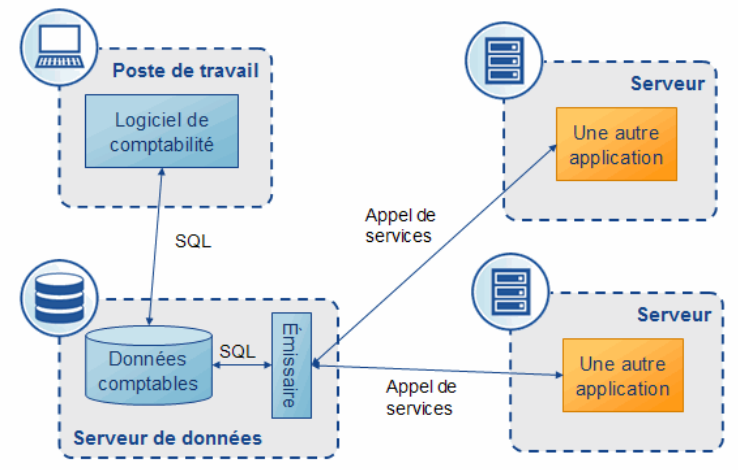


Fig III.3 Architecture de la solution proposée (Gilles, 2008)

Les avantages sont multiples:

- Faible couplage entre la base de données et les applications consommatrices.
- Possibilité d'intégrer dans l'émissaire des fonctions de filtrage, de transformation, de transcodage, de cache, etc.
- Possibilité d'exposer les services suivant plusieurs protocoles (RMI, HTTP, etc.)

Limitation de la méthode : La méthode est très facile à implanter et possède de multiples avantages mais elle présente les limites suivantes

- Dans le cas où le SGBD utilisé ne supporte pas des sessions multiples, on aura des erreurs lors des accès simultanés par plusieurs services, sauf si l'émissaire fonctionne en mode « invocation asynchrone ».
- Certaines fonctionnalités n'ont pas de résultats sur la base de données, donc selon le principe de la méthode nous ne pouvons pas bénéficier de cette fonctionnalité.

2.2.3. Encapsulation des applications : (connue sous le terme " business logic wrapping")

Le but principal d'un wrapper est d'assurer la liaison entre le service demandeur et le système legacy, en adaptant la requête selon les caractéristiques du système legacy, et de même pour la réponse qui sera adaptée selon le protocole du client et sera redirigée vers le client demandeur ; c'est le cas le plus simple. Dans le cas complexe, un wrapper est construit pour faire la liaison entre le demandeur de service et l'interface de legacy, effectuer les transformations sur les données échangées, garantir la bonne exécution des

transactions et prendre soin du cheminement des messages et de la manipulation des exceptions.

Afin de fournir un service d'encapsulation à un système legacy, une certaine forme de technologie d'adaptateur, ou "adaptateur de service" est exigée. Le but de cette technologie est de pouvoir s'intégrer avec les systèmes *non-SOA* et d'appliquer quelques transformations de données ou de protocoles qui soient exigées pour exposer une interface propre de service représentant la fonctionnalité du legacy ; cette interface est connue sous le terme "service d'encapsulation" ("Wrapper service"). Les consommateurs de service peuvent alors bénéficier des fonctionnalités du legacy en interrogeant l'adaptateur qui lui même s'occupe de la transformation de la requête selon la particularité du legacy**.** Un adaptateur de service est développé pour chaque fonctionnalité afin de permettre l'accès à cette dernière**.**

Dans (Dietmar, et al., 2002) une architecture conceptuelle a été proposée, et qui peut être employée comme modèle pour rendre les applications Legacy disponibles en tant que services Web, et donc possible d'être intégrées dans une SOA. La figure Fig III.4 montre la structure de cette architecture

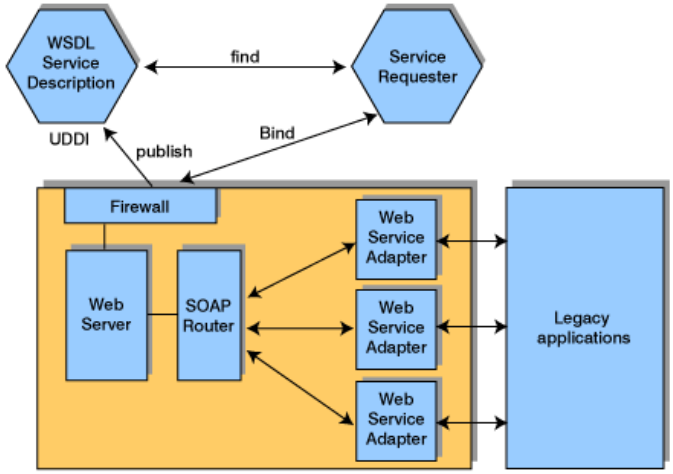


Fig III.4 Architecture conceptuelle pour l'accès aux applications legacy par l'intermédiaire de SOAP **(Dietmar, et al., 2002)**

Quand le demandeur de service choisit un service, il emploie la description WSDL pour découvrir comment accéder au service. Une fois trouvée, la description WSDL est

employée pour produire un message SOAP de demande envoyé au serveur d'applications qui agit en tant que fournisseur de service.

Cette requête SOAP est publiée comme requête POSTE HTTP par exemple. Employer le HTTP a l'avantage qu'il peut passer les pare-feu normalement disponibles sur les serveurs web. Le message de requête SOAP après dépassement du pare-feu est manipulé par le serveur HTTP. Ce serveur analyse l'information d'en-tête de HTTP, et trouve comme éléments le nom uniforme de ressource (URNE) et le nom du composant de routeur SOAP. Le message de demande est passé au routeur SOAP indiqué. Le routeur SOAP analyse l'en-tête de HTTP, et trouve l'endroit d'un adaptateur de service web. Le routeur passera alors la demande à l'adaptateur demandé.

L'adaptateur de service web doit être développé pour chaque service à fin de permettre l'accès à ces derniers. C'est typiquement un programme d'application qui se relie au serveur principal. Ce raccordement peut être n'importe quelle liaison soutenue par le serveur principal, c.-à-d., cela peut être un native canal MQSeries fonctionnant sur une connexion de protocole TCP/IP "Transmission Control Protocol/Internet Protocol" ou de protocole APPC « Advanced Program to Program Communication».

Dans (Balis.B, et al., 2004) (Bartosz Balis et al) ont proposé une solution pour adapter des applications legacy à des environnements de services web, l'architecture prend en compte et améliore les critères fondamentaux tels que la performance, la sécurité, la scalabilité, et la tolérance des fautes, elle fournit un support pour la migration de processus en cas de panne et pour le traitement transactionnel, ainsi des modèles de méthodes d'invocations concourantes et asynchrones sont soutenus. La **figure** Fig III.5 montres l'architecture de la solution

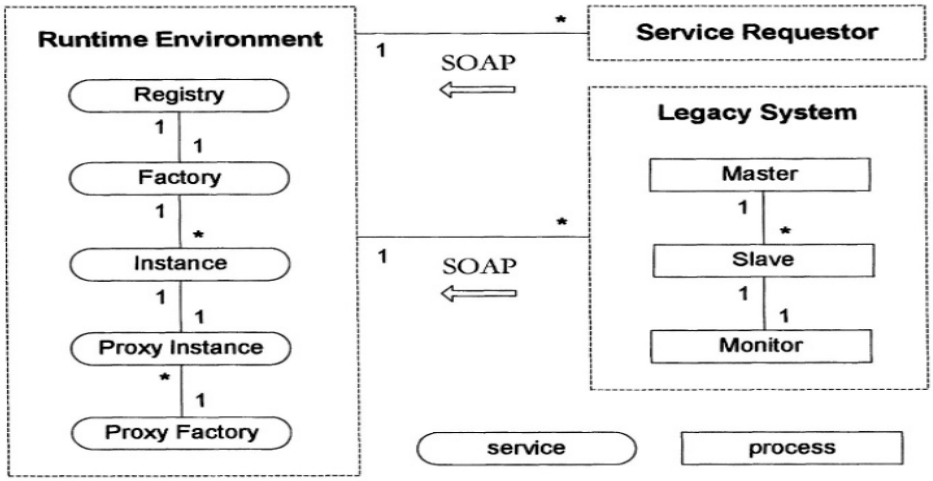


Fig III.5 Architecture proposée pour l'encapsulation des applications legacy C/C++

Ici le système Legacy constitue un environnement dans lequel le logiciel Legacy est exécuté et non pas le logiciel lui même, les processus qu'il contient jouent le rôle d'adaptateurs, ils sont responsables de la demande réelle de traitement, ils ne sont pas permanents et ils sont créés par les services de l'environnement d'exécution (Runtime environment). Les clients peuvent bénéficier des fonctionnalités du legacy en interrogeant les services de l'environnement d'exécution ; ce dernier est constitué de trois services permanents déployés : *registry, factory* et *Proxy factory* et selon le nombre de clients simultanément servis que le nombre de services passagers change, à savoir *les* services i*nstance* et *Proxy instance*, qui sont en relation un à un. Les services passagers sont instanciés par les usines (factory) correspondantes et sont possédés par leurs créateurs. Le service web registry est responsable de contrôler le mapping un à un entre les demandeurs de service et les systèmes legacy. Il fournit l'interface qui peut être utilisée pour :

- assigner un des systèmes legacy enregistrés à un client en attente,
- annoncer que le système legacy est prêt à traiter les demandes

A fin de permettre un accès concurrent aux fonctionnalités du logiciel legacy les auteurs proposent d'exécuter plusieurs copy du legacy sur des machines différentes et à chaque copy un processus master est enregistré. Il est a noté que le framework exige que les développeurs doivent fournir l'implémentation des méthodes (interfaces) C/C++ qui vont

être exposées comme services après la génération des codes appropriés ; donc la solution est limitée aux applications orienté-objets et dont la structure et la sémantique du code source du système legacy soient connues et disponibles.

Dans (Bo, et al., 2008) Bo Zhang & al ont présenté une stratégie d'encapsulation des applications GUI (Graphic User Interface) où un outil très intéressant est utilisé pour invoqué les opérations invocables initialement via l'interface graphique de l'application legacy de manière programmatique ; c'est un module amélioré de Lua qui se compose d'un interpréteur Lua, un script lua et d'APIs écrite en C/C++ lié a l'interpréteur pour étendre leur fonctionnalité.

Les opérations *GUI* sont encapsulées pour être présentées en tant que services Web placés dans un conteneur de service. Un manuscrit *Lua* décrivant un ensemble d'opérations de GUI est exigé pour être identifié par le service d'encapsulation qui produit un manuscrit de *python* et un fichier *WSDL*. Quand le service d'opération de GUI est invoqué, le conteneur de service exécute le manuscrit de python qui appelle l'interpréteur *Lua* pour exécuter le manuscrit *Lua* correspondant**.**

Cette approche utilise une bonne technique pour l'exécution automatique des opérations de l'interface graphique en générant un script *Lua* à partir des entrées reçues du service web associé, mais pour des fenêtres très compliquées la génération des scriptes sera peut être non exacte ; un autre problème peut être signalé concernant le résultat de la conversation (dans le cas des scénarios comportant plusieurs étapes de conversation) sous quelle forme est envoyé le contenue de la fenêtre au client du web service?

Concernant toujours les applications interactives à interface graphique, en (Gerardo, et al., 2006) une méthodologie d'encapsulation pour rendre les fonctionnalités interactives des systèmes legacy accessibles comme services web a été présentée. Le système d'encapsulation agit en tant qu'interpréteur d'un automate d'état fini qui décrit le modèle d'interaction entre l'utilisateur et le système legacy ; ce modèle est obtenu par des techniques d'ingénierie inverse. Dans ce modèle, un état d'interaction représentera une étape du dialogue entre l'utilisateur et le système dans laquelle le legacy renvoie un écran et attend l'utilisateur pour soumettre un nouvel ensemble de données et de commandes sur cet écran. Par conséquent, un état d'interaction peut être associé à un modèle d'écran et un ensemble d'actions à exécuter sur ses champs (le modèle d'écran se compose d'un ensemble de champs, qui peuvent être des champs input, des champs output, ou des

champs label). Le système d'encapsulation (wrapper) est composé de plusieurs modules **(**Fig III.6**)** :

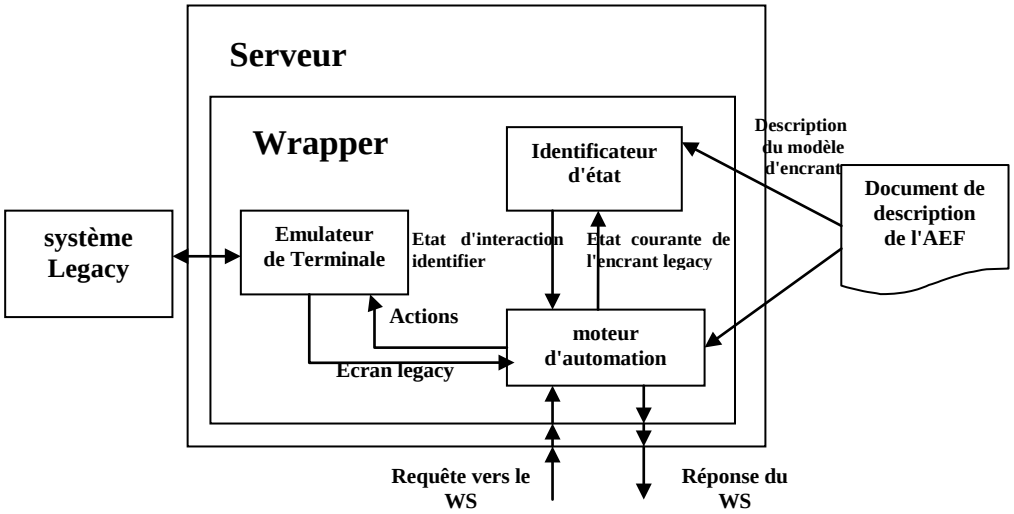


Fig III.6 Composantes du wrapper (Gerardo, et al., 2006)

- L'émulateur de terminal est un composant software développé pour implémenter un protocole de serveur de terminal, il est responsable de créer un terminal virtuel et de gérer son clavier et affichage virtuel.
- L'identificateur d'état a la responsabilité d'identifier quel modèle d'écran associé à ces états possibles assortit avec l'écran retourné par le système legacy.
- Le moteur d'automation constitue Le noyau du wrapper, il est responsable d'interpréter l'Automate d'Etat fini (AEF) lié à un service donné offert par le système legacy.

Le système proposé est une stratégie black-box pour l'intégration des legacy interactive à base d'interface utilisateur graphique dont le point le plus dure est de remplacée l'interaction utilisateur-legacy par une interaction automatique sans jouer sur le code source, la stratégie ne peut pas être appliquée dans tout les cas des applications GUI surtout pour les interfaces complexes et qui sont manipulées par la souris.

Une force de cette approche est que les composants d'encapsulation peuvent être réutilisés pour exporter différents cas d'utilisations mises en application par différents

systèmes. En effet, le moteur d'automate peut être réutilisé tel qu'il est, à condition que le modèle d'interaction décrivant le nouveau cas d'utilisation soit produit.

L'article (Giusy, et al., 2007) a adressé le problème de migration des fonctionnalités des applications web existantes vers les architectures orientées service par une approche d'encapsulation. Cette approche considère une application web comme un type spécial de système à base d'interface graphique, et il propose d'adopter les systèmes d'encapsulations à base d'Automate d'état fini pour transformer l'interface orientée-utilisateur originale de l'application web (non programmatique) en un interface programmatique qui expose la pleine fonctionnalité et données de l'application. La plateforme de cette approche emploi plusieurs modules:

- Le directeur d'interaction de WA est le composant mettant en application l'interaction automatique avec une page Web en employant le framework HttpUnit
- State Identifier est le composant qui permet l'identification de l'état d'interaction courante, il exploite la technique de classification de page Web.
- Les spécifications d'automates sont constamment stockées dans une base de données basée sur XML où l'accès est permis par des technologies génériques de contrôle de données (telles que des bibliothèques DOM ou SAX).

3. Conclusion

Vue l'énorme différence dans les architectures des systèmes legacy, dans la façon dont leurs fonctionnalités sont appelées, dans les caractéristiques de leur environnement d'exécution et dans beaucoup d'autres critères, il est impossible de définir une solution généralisée utilisée pour résoudre le problème d'intégration des fonctionnalités pour toute catégorie de système legacy mais il est possible pour chaque catégorie d'élaborer une solution spécifique.

Selon notre perspective, les applications traditionnelles pourront être divisées en deux catégories :

- Applications monolithiques : cette catégorie est la plus dure à encapsuler car les fonctionnalités ne sont pas dispersées et sont généralement invoquées à l'aide d'une interface graphique d'utilisateur.

- Applications à fonctionnalités dispersées : la conception d'adaptateur dans se cas est simple il suffit de connaître le format d'invocation des fonctionnalités. Cette catégorie englobe les applications distribuées et les applications client/serveur

Chapitre IV
Solution adoptée

Chapitre IV Solution adoptée

1. Introduction

Nous avons choisi la migration des applications CORBA vers les architectures SOA par l'exposition des méthodes de leurs objets en tant que services web. Cette classe de système legacy n'a pas été prise en charge par les approches citées dans le chapitre précédent.

Les applications CORBA sont aujourd'hui considérées comme des systèmes legacy car elles se basent sur une technique pré-SOA (voir la définition donnée dans (Cetin, et al., 2007)). D'autre part, bien qu'elles vérifient quelques principes de l'architecture SOA, elles souffrent de quelques problèmes de complexité. Dans (Kanchanavally, et al., 2005) l'auteur indique que les services CORBA ne sont pas des services SOA car ils ne vérifient pas la norme essentielle qui dit que le service ne doit pas être dépendant d'une technologie précise, il doit pouvoir être invoqué par d'autres codes écrits dans n'importe quel langage de programmation.

Par le choix des applications CORBA nous croyons que nous élargissons l'applicabilité de notre système d'encapsulation du fait que plusieurs organisations ont adopté les projets d'intégration des applications de l'entreprise dont CORBA est une des technologies utilisées dans ces projets.

Notre travail consiste à implémenter une plateforme générique à base du paradigme multi agents (voir l'annexe C sur les systèmes Multi-Agents), destinée aux développeurs propriétaires des systèmes legacy à base d'architecture CORBA afin de leur permettre la présentation de leurs fonctionnalités comme capacités d'agents et comme services web (Fig IV.1).

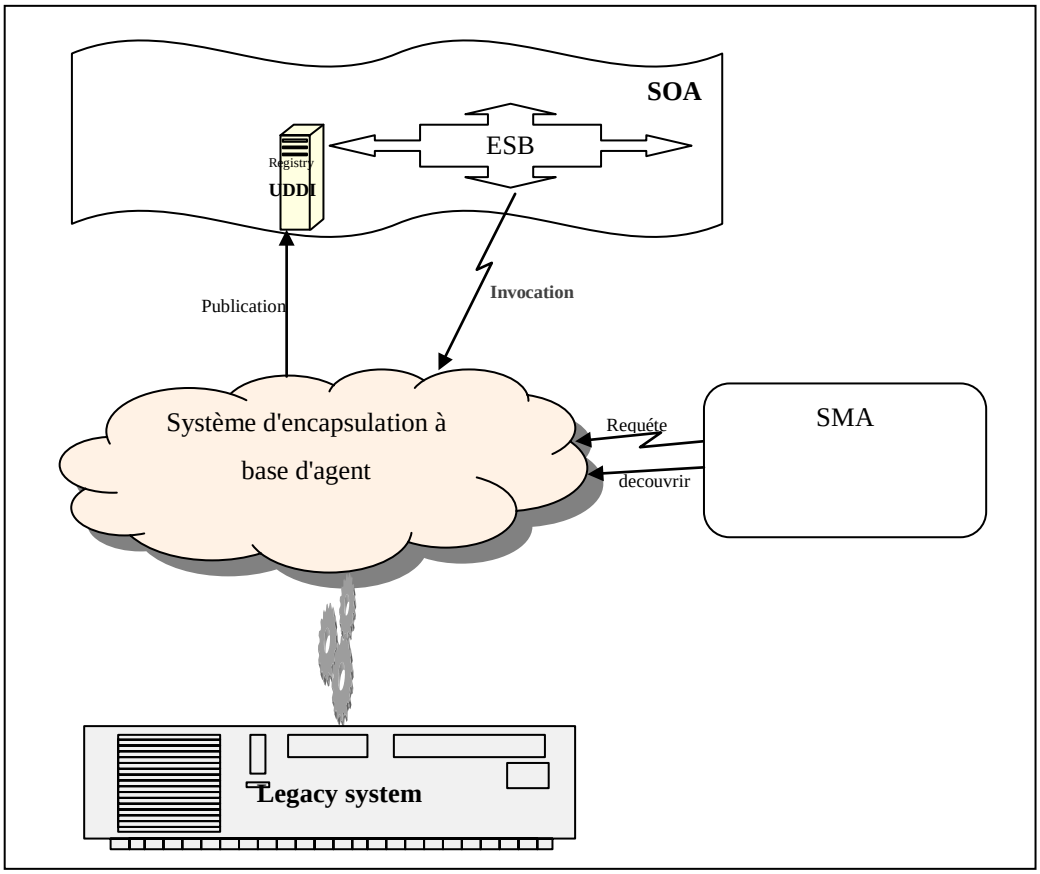


Fig IV.1 Encapsulation des fonctionnalités du legacy system

Nous avons choisi le paradigme multi agents pour les raisons suivantes :

- Partage du code
- Création dynamique des représentants
- Souplesse de gestion des représentants (ajouts, suppressions, modifications, réinitialisation et visualisation d'état)
- Facilité de la composition des fonctionnalités par la composition des agents représentants
- Performance :
    - possibilité de gestion des invocations simultanées et minimisation du taux de perte de requêtes en utilisant une stratégie de timing.
    - Possibilité d'implémenter un mécanisme de communication asynchrone.

Chaque agent dans le système proposé sera caractérisé par un ou plusieurs comportements, chaque comportement représentant une tâche qu'un agent peut effectuer**.**

2. Présentation de la solution

2.1. Architecture globale de la solution

En se basant sur les règles suivantes du comportement attendu de notre système, une formulation initiale du système adopté peut être dégagée **(**Fig IV.2**)** (F.Haniche, et al., 2010)

- Un objet CORBA est représenté par un agent: chaque Objet corba serveur est représenté par un Agent dans le système et à chaque méthode (fonctionnalité) de cet objet correspondra une capacité de cet agent, d'où le premier type d'agent : l'**Agent Service**.
- Définition de nouveaux services : l'utilisateur du système doit bénéficier d'une interface lui permettant de publier de nouvelles fonctionnalités du système legacy, d'où le deuxième types d'agent : l'**Agent Interface ;**
- Une interface entre le système legacy et notre système doit être définie : celle-ci permet de transformer les requêtes qui sont sous forme de ACL-message (Agent Communication Language message) en des demandes effectives vers le système legacy d'où le type d'agents nommé l'**Agent Adaptateur ;**
- Les capacités des agents service devront être publiées en tant que services web et les invocations, de ces capacités, qui viennent de l'extérieur devront être traduites et redirigées: là un autre agent doit intervenir ; il sera nommé l'**Agent passerelle.** Cet Agent travaille en collaboration avec un **servlet** qui s'occupe de la réception des messages SOAP, provenant des web services clients, ainsi que l'envoi des résultats d'invocations aux clients correspondants ;
- Les capacités des agents service devront être enregistrées dans un annuaire pour être découvertes et invoquées par d'autres agents du SMA: d'où la nécessité d'un agent dit **Agent facilitateur**.

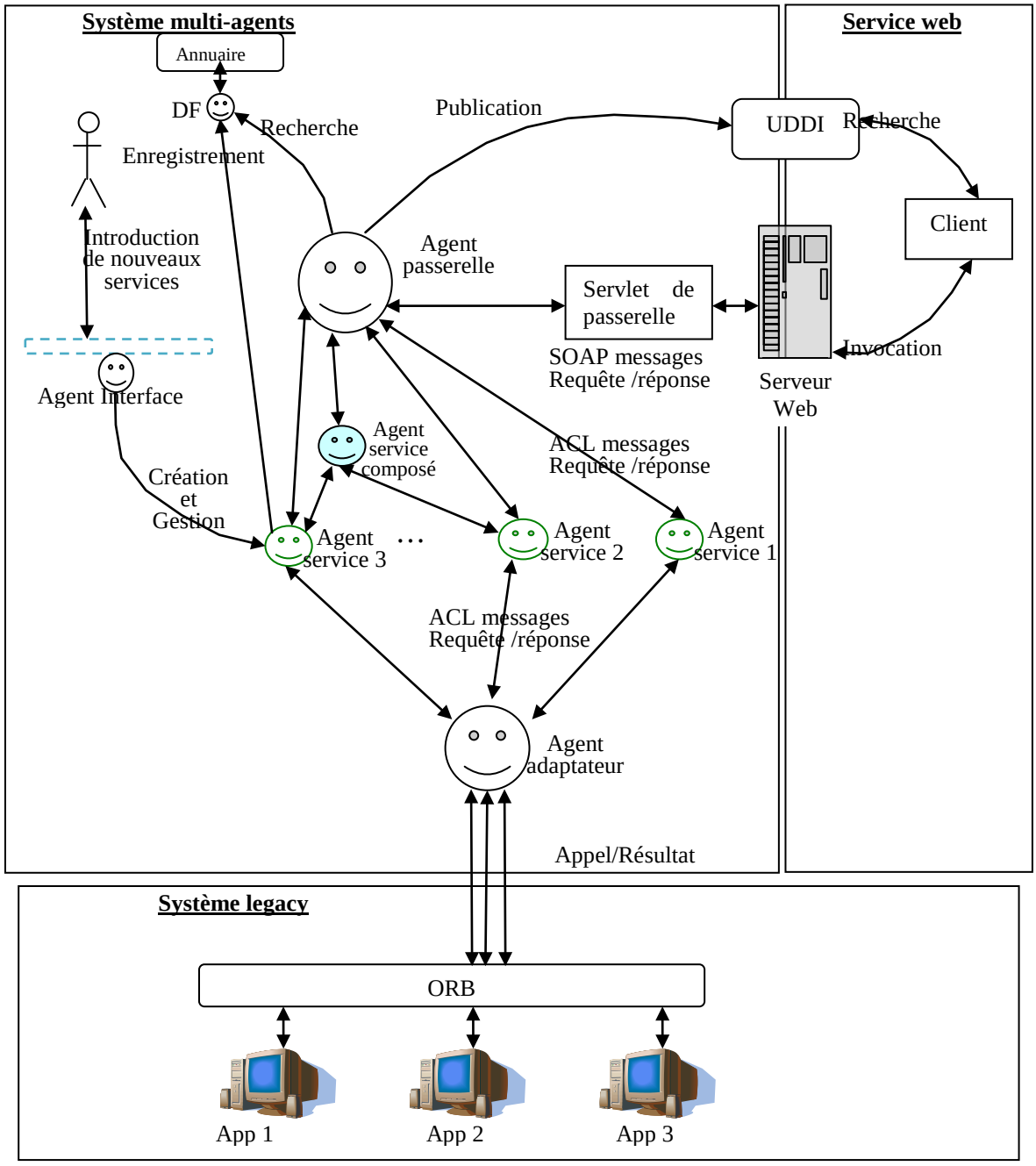


Fig IV.2 Architecture Multi-agents du système

2.2. Description des types d'agents du système

2.2. 1. L'Agent Interface

Cet agent est un outil d'administration des agents du système. Il permet de créer de nouveaux agents service simple ou composite via une interface graphique en imposant à l'utilisateur de fournir un fichier de scripte IDL (Interface Definition Language) contenant la définition de l'interface d'objet Corba, les noms du service et d'opérations utilisées à la publication, et la syntaxe d'appel des méthodes et leurs paramètres d'entré/ sortie. Certaines de ces informations seront utilisées par l'agent adaptateur pour l'invocation des fonctionnalités et d'autres par l'agent passerelle pour la génération de la description WSDL**.** La figure Fig IV.3 suivante expose l'architecture globale de cet agent où CP1, CP2 et CP3 sont les comportements de cet agent suivant l'événement déclencheur.

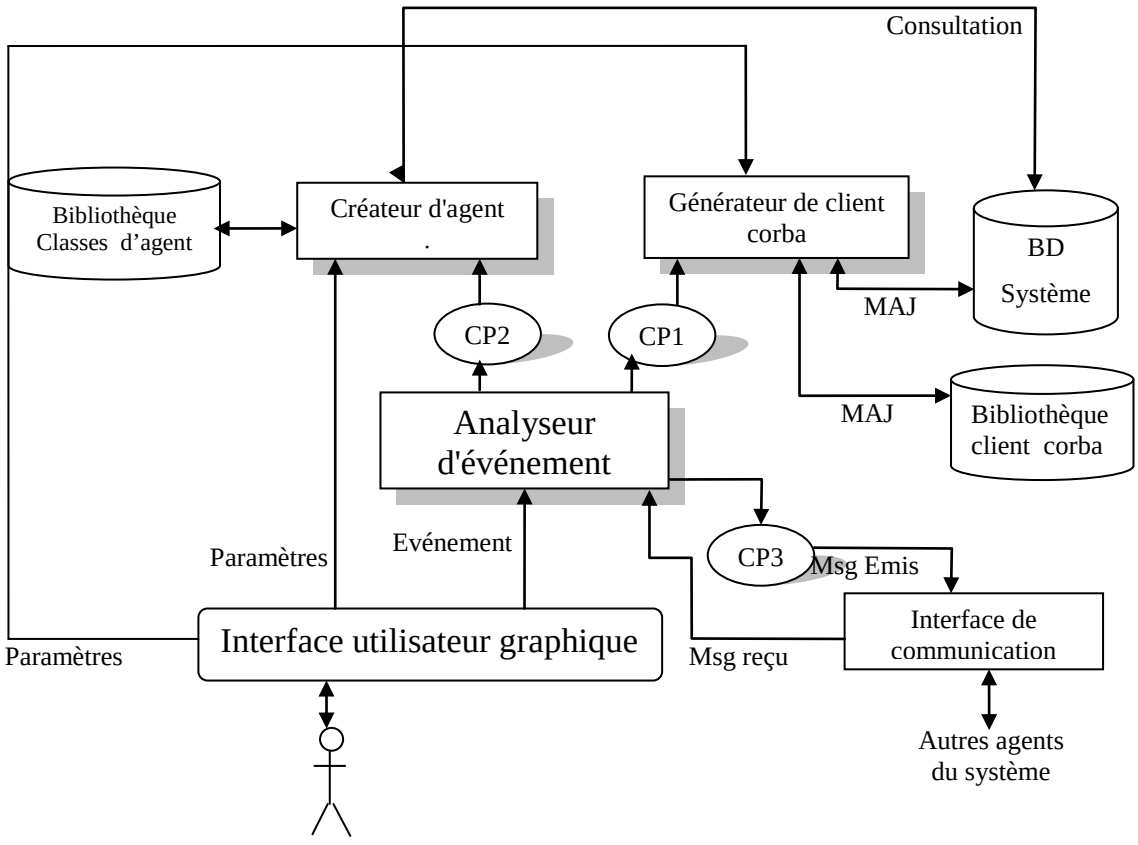


Fig IV.3 Architecture de l'Agent Interface

Responsabilités :

Type d'agent	responsabilités
Agent Interface	- créer l'Agent Service Simple ou Composé à partir de la bibliothèque des classes d'Agent Service (selon la demande de l'utilisateur) - génération des classes du client corba suite à la création d'un nouvel agent service - compilation des nouvelles classes générées - désinscrire un agent service à partir de l'annuaire du système (suite à une demande de l'utilisateur) - désinscrire un agent service à partir de l'UDDI - permettre à l'utilisateur de consulter la liste des agents service enregistrés - permettre à l'utilisateur de modifier les caractéristiques d'un agent service - Permettre à l'utilisateur de choisir la classe d'agents appropriée à partir de la liste des classes disponibles - relancer les agents service déjà créés après redémarrage du système (par la consultation de la base de données contenant les informations sur les services créés)

Ressources :

Plusieurs ressources sont manipulées par cet agent

- les fichiers de scripte IDL correspondant à chaque objet à encapsuler, utilisés par le système pour la génération des classes stub et client CORBA, nécessaire à l'invocation effective des méthodes de l'objet CORBA.
- Une base de données contenant les paramètres du système ainsi que les informations sur les agents service créés (Remarque: le schéma de la base de données sera exposé ci-après).

— Bibliothèque des classes d'agents service, enrichie par de nouvelles classes d'Agents Service Simple ou Composé créés chaque fois qu'un nouvel objet corba est sujet d'encapsulation,

— Bibliothèque des stubs et clients CORBA : les classes de cette bibliothèque sont générées dynamiquement de manière automatique par cet agent afin d'être utilisées par l'agent adaptateur lors de l'invocation des fonctionnalités correspondantes.

2.2.2. L'Agent Service

C'est le représentant d'un objet corba serveur du système legacy. Pour chaque objet à encapsuler et qu'on veut publier en tant que WS, un Agent Service sera créé, ce dernier aura comme capacités, ou services fournis, les fonctionnalités des méthodes de cet objet. Ces capacités seront réalisées sous forme d'appels de procédures, suivis de paramètres d'entrée, encapsulés dans un message ACL et envoyé à l'*Agent Adaptateur* qui s'occupe de les retransmettre à l'objet CORBA en question après transformation. Il est à noter que l'agent service peut fournir un service composé déduit d'une simple coopération des Agents Service existants et dans ce cas l'agent joue le rôle d'un coordinateur entre les agents impliqués. Nous distinguons donc deux catégories d'Agents Service :

a) Agent Service Simple : un agent service simple est un représentant d'un objet corba existant et dont les capacités représentent les fonctionnalités encapsulées par cet objet. Pour interroger une fonctionnalité du système legacy, il envoie un message de requête vers l'Agent Adaptateur contenant le nom de l'objet CORBA, le nom de la méthode et les paramètres d'entrée utilisés dans l'invocation effective de la fonctionnalité.

b) Agent Service Composé : Notre plateforme supporte plusieurs modèles d'agents service composé. Le modèle sera choisi par l'utilisateur créateur du service mais il doit indiquer les noms d'agents service inclus dans la composition dans un ordre bien défini.

Responsabilité :

Type d'agent	responsabilités
Agent Service Simple	- s'enregistre auprès de l'agent facilitateur (pages jaunes) par l'envoi d'un message d'enregistrement - s'enregistre auprès de l'UDDI par l'intermédiaire de l'agent passerelle en envoyant à ce dernier un message de demande d'enregistrement - envoie une requête à l'Agent Adaptateur suite à un message d'invocation de fonctionnalité, reçu de la part de l'agent passerelle - Envoi d'un message contenant le résultat de l'invocation de la fonctionnalité vers l'agent passerelle après la réception de la réponse de la part de l'agent adaptateur
Agent Service Composé	- s'enregistre auprès de l'agent facilitateur (pages jaunes) par l'envoi d'un message d'enregistrement - s'enregistre auprès de l'UDDI par l'intermédiaire de l'agent passerelle en envoyant à ce dernier un message de demande d'enregistrement - invoque les capacités des agents service composants selon le schéma du modèle de composition, suite à un message d'invocation reçu de la part de l'agent passerelle - Envoie un message contenant le résultat de l'invocation de la fonctionnalité vers l'agent passerelle après la réception de la réponse de la part de l'agent adaptateur

2.2.3. L'agent adaptateur

Il joue le rôle d'interface entre les *agents service* et le système legacy, il s'occupe de la transformation du message requête-ACL de l'agent service vers une invocation effective de la fonctionnalité correspondante du système legacy en se basant sur les classes des stubs et clients CORBA et, à la réception du résultat, il construira un message-Acl contenant le résultat afin de l'envoyer vers l'agent demandeur. Il est équipé de structures de données lui permettant de lier le résultat avec l'agent demandeur.

Responsabilité :

Type d'agent	responsabilités
L'Agent Adaptateur	- Initialisation de l'ORB suite à une requête d'invocation de l'Agent Service - invocation de l'objet CORBA distant suite à une requête de l'Agent Service - ré-invocation de l'objet CORBA distant en cas d'erreur de réception ou de temps dépassé. - Envoi d'un message contenant le résultat de l'invocation de la fonctionnalité vers l'Agent Service après la réception de la réponse de la part de l'objet corba distant

2.2.4. L'Agent Passerelle

Il joue le rôle d'interface entre le domaine des systèmes multi agents et du domaine des services web. Il est équipé de plusieurs modules lui permettant de réaliser les différentes transformations de messages et de descriptions

Responsabilité :

Type d'agent	responsabilités
L'Agent Passerelle	- Recevoir et traduire la description du service d'agent vers la description WSDL correspondante - publier la description WSDL dans un annuaire UDDI et déployer le service dans le serveur web - Recevoir les invocations provenant des services web externes (client), - transformer les messages SOAP en des messages ACL, - chercher l'identificateur de l'agent correspondant au service dans le DF et envoyer le message généré à cet agent - Recréer le message SOAP de retour à partir de la réponse de l'agent service et l'envoyer au client correspondant

2.2.5. Le Facilitateur d'Annuaire DF (Directory facilitator)

C'est un agent qui est muni d'un registre dans lequel les descriptions des capacités des agents sont enregistrées. Bien sûr, les agents sont différenciés par un identificateur unique. Le Facilitateur implémente des méthodes qui facilitent l'enregistrement, la suppression et la recherche des services d'agent dans l'annuaire (Fig IV.4)

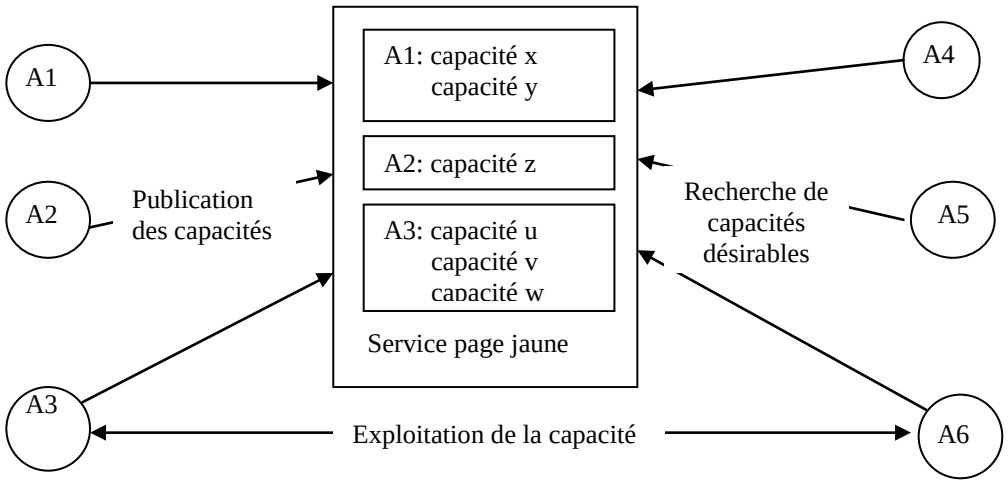


Fig IV.4 Principe du service facilitateur d'annuaire

Responsabilité :

Type d'agent	responsabilités
L'Agent facilitateur d'annuaire	- Recevoir les messages d'inscription, de désinscription et de recherche des agents service - mise à jour de l'annuaire des services d'agent suite à une demande d'inscription, de désinscription ou de modification. - envoi d'une liste d'identificateurs d'agents publiant un service désiré suite à une requête de recherche - envoi des notifications sur les nouvelles inscriptions vers les agents abonnés.

2.3. Les modèles d'agents service composite

Le discriminateur

Afin d'augmenter la disponibilité d'une fonctionnalité, certains legacy systems déploient une même fonctionnalité dans deux ou plusieurs sites (i.e plusieurs points d'accès pour une même fonction). Pour de telles fonctionnalités notre système permettra leur présentation en un service unique, lequel, lorsqu'il sera invoqué, lancera deux requêtes en parallèle sur les deux sites et le résultat du site qui répondra le premier sera envoyé au client demandeur et le dernier sera ignoré (Fig IV.5). Ce modèle de composition est dit le discriminateur (Hamadi, et al., 2003). Dans la figure la fonctionnalité dans le premier site est représentée par la capacité Cp1 de l'agent A1, et celle déployé dans le deuxième site par la capacité Cp2 de l'agent A2.

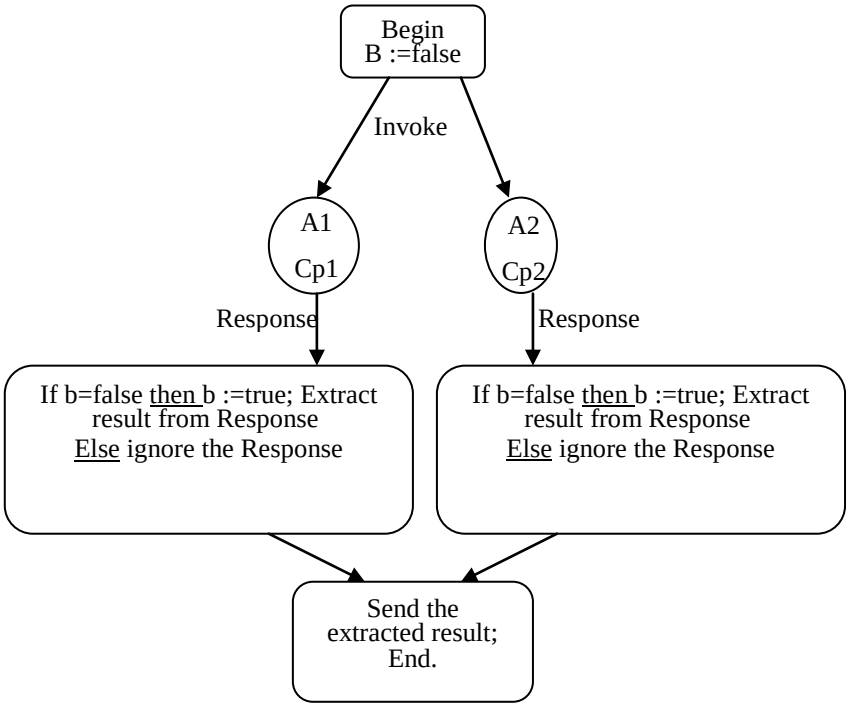


Fig IV.5 Modèle de composition « le Discriminateur »

Le séquenceur

L'agent service séquenceur permet l'exécution de deux fonctionnalités f 1 et f 2 dans l'ordre, c.-à-d., l'une après l'autre ; f 1 doit être accomplie avant que f 2 commence. C'est typiquement le cas où une fonctionnalité dépend du rendement de la fonctionnalité

précédente et où le résultat final sera celui de la deuxième fonctionnalité. La figure Fig IV.6 illustre le comportement d'un agent service composite de type séquenceur pour ordonner l'invocation des deux fonctionnalités f1 et f2. La capacité cp1 de l'agent A1 correspond à la fonctionnalité f1 et possède deux paramètres d'entrée p1, p2 ; la capacité cp2 de l'agent A2 correspond à la fonctionnalité f2 et possède trois paramètres d'entrée R1 (qui est le résultat de cp1), p3 et p4.

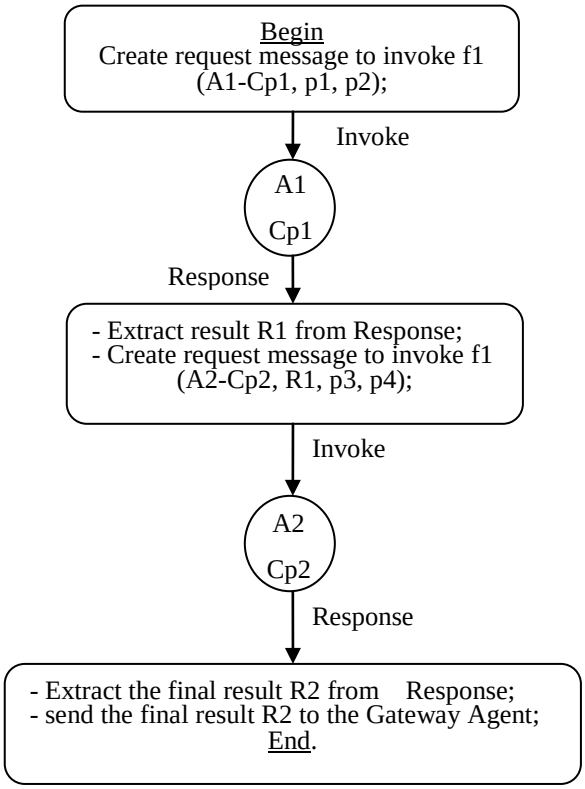


Fig IV.6 Modèle de composition « le Séquenceur »

L'Alternateur

Ce modèle de composition permet d'exécuter une de deux fonctionnalités selon une condition définie par l'utilisateur du service

La figure Fig IV.7 illustre le comportement d'un agent service composite de type Alternateur tel que f1 et f2 sont représentées respectivement par la capacité Cp1 de l'agent A1 et la capacité Cp2 de l'agent A2

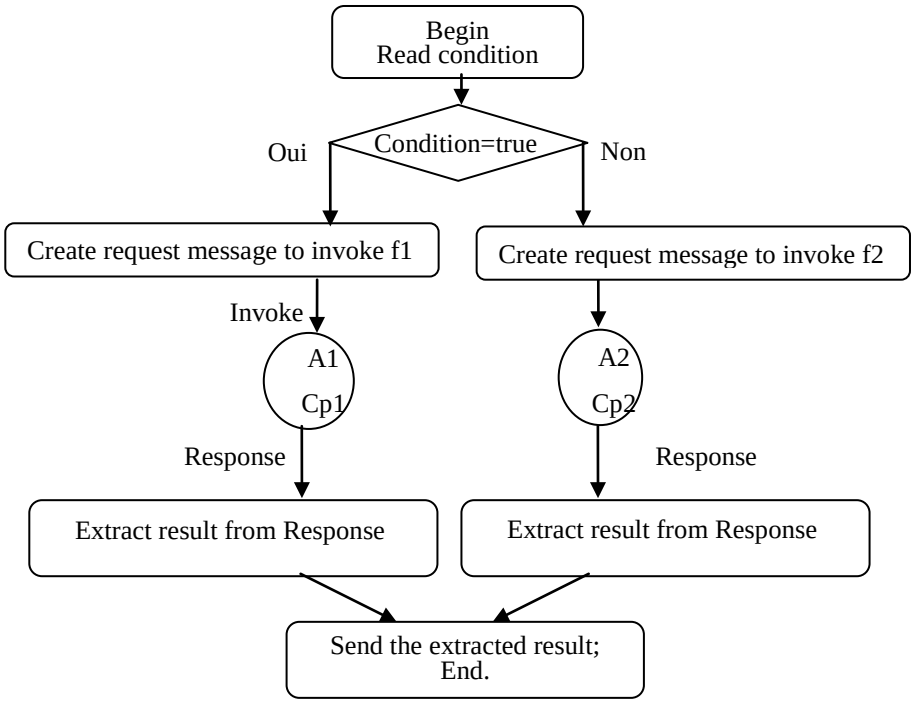


Fig IV.7 Modèle de composition « l'Alternateur »

L'Itération

Ce modèle permet d'exécuter une fonctionnalité (f), représentée par la capacité Cp1 d'un agent A1, un certain nombre (nb) de fois selon la spécification de l'utilisateur du service

La figure Fig IV.8 illustre le comportement d'un agent service composite de type Itération

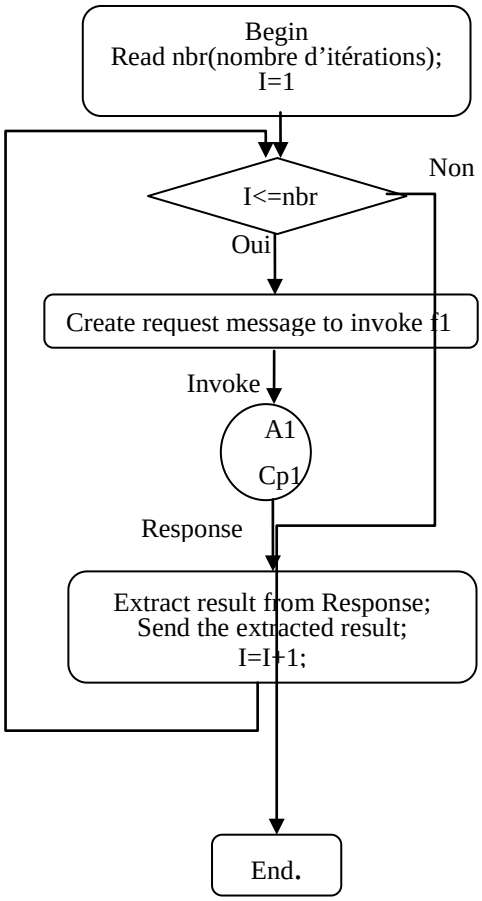
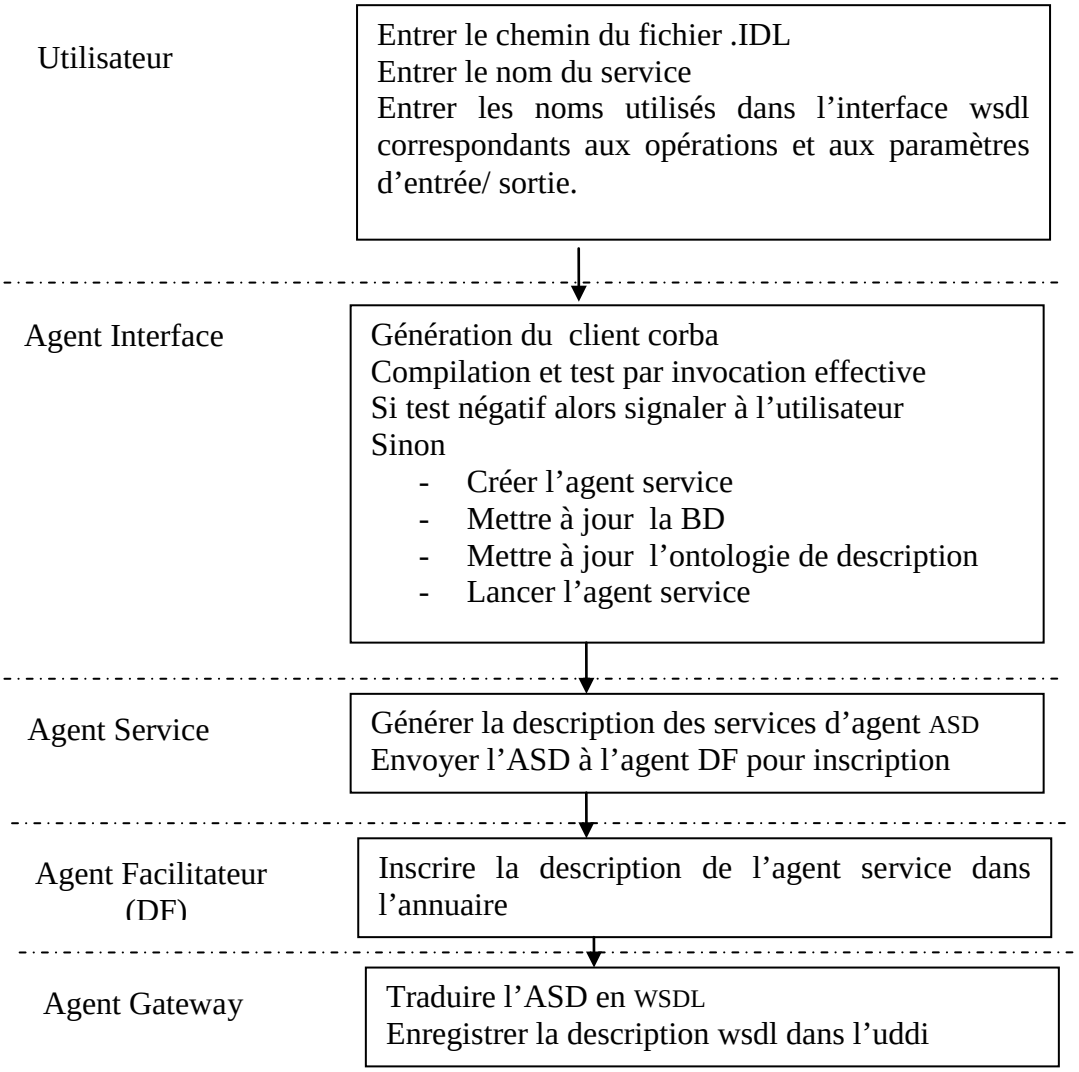


Fig IV.8 Modèle de composition « l'Itération »

2.4. Les scénarios

Il y a plusieurs scénarios indépendants qui peuvent être pris en charge par notre architecture dans de diverses circonstances. Nous nous limitons ici à présenter les principaux scénarios :

2.4.1. Création d'agent service simple

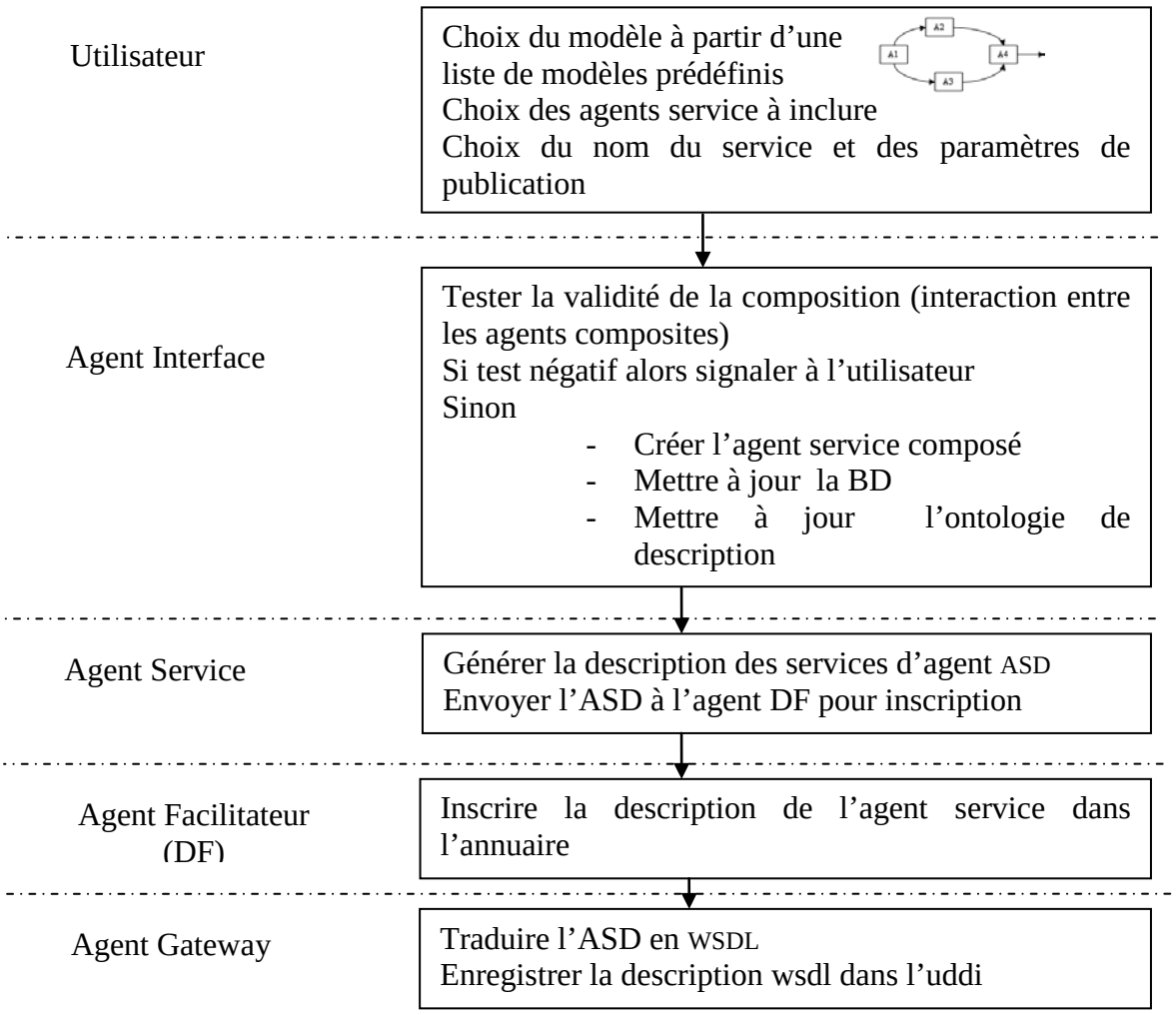


Remarque

Il est à noter que le concept d'ontologie utilisé ici est similaire à celui utilisé dans la plateforme JADE (Java Agent DEvelopment framework). Elle pourra être considérer comme une structure de données permettant de mettre en correspondance

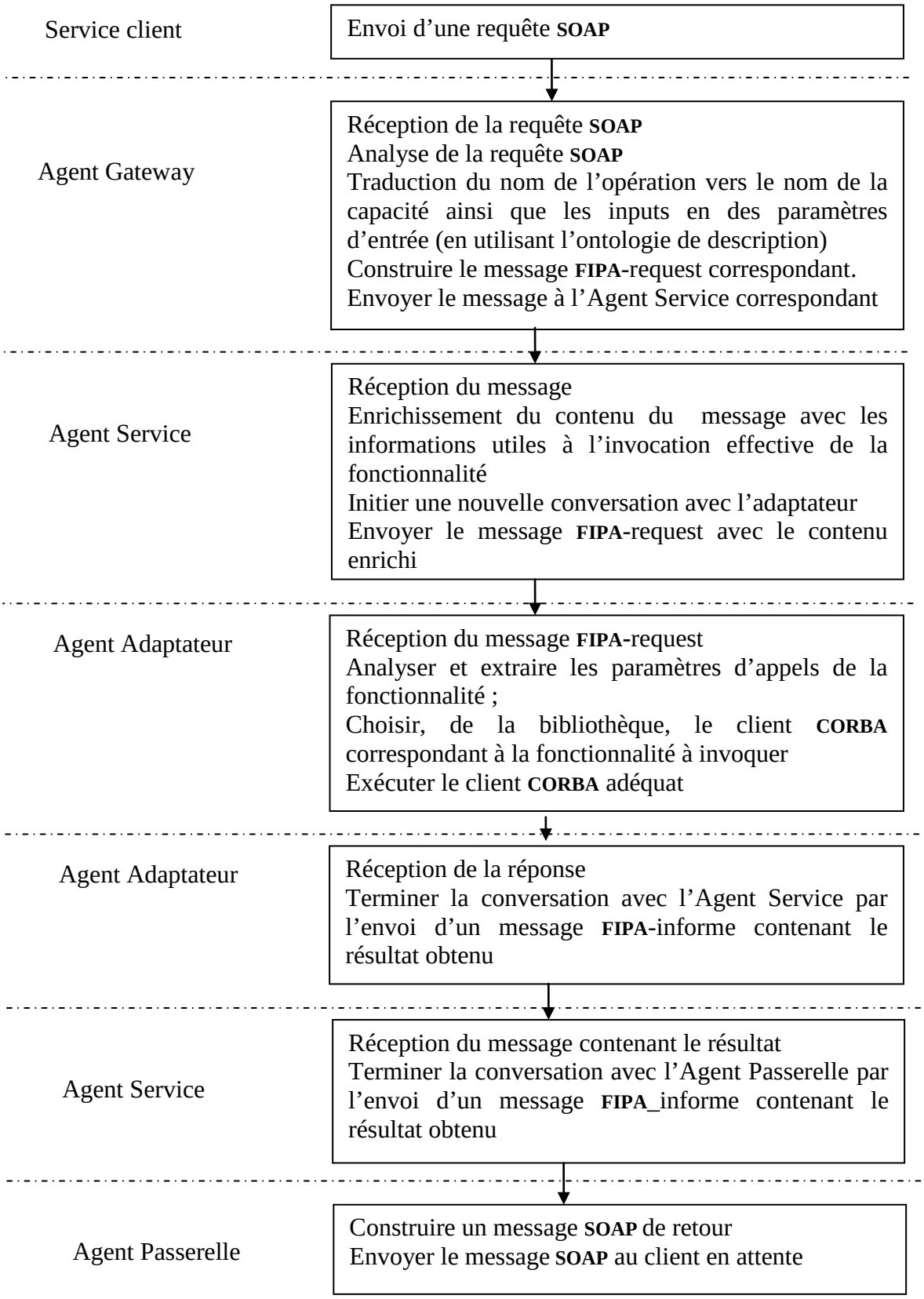
- Les noms et les identités de paramètres des actions d'un agent et
- Les noms et les paramètres des opérations utilisés dans l'interface WSDL correspondante à cet Agent.

2.4.2. Création d'agent service composite (agent représentant de fonctionnalité composite)



2.4.3. Invocation de fonctionnalité simple par un service web externe

La figure Fig IV.9 illustre l'invocation d'une simple fonctionnalité par un client de service web


Service client
Envoi d'une requête SOAP
Agent Gateway
Réception de la requête SOAP
Analyse de la requête SOAP
Traduction du nom de l'opération vers le nom de la capacité ainsi que les inputs en des paramètres d'entrée (en utilisant l'ontologie de description)
Construire le message FIPA-request correspondant.
Envoyer le message à l'Agent Service correspondant
Agent Service
Réception du message
Enrichissement du contenu du message avec les informations utiles à l'invocation effective de la fonctionnalité
Initier une nouvelle conversation avec l'adaptateur
Envoyer le message FIPA-request avec le contenu enrichi
Agent Adaptateur
Réception du message FIPA-request
Analyser et extraire les paramètres d'appels de la fonctionnalité ;
Choisir, de la bibliothèque, le client CORBA correspondant à la fonctionnalité à invoquer
Exécuter le client CORBA adéquat
Agent Adaptateur
Réception de la réponse
Terminer la conversation avec l'Agent Service par l'envoi d'un message FIPA-informe contenant le résultat obtenu
Agent Service
Réception du message contenant le résultat
Terminer la conversation avec l'Agent Passerelle par l'envoi d'un message FIPA_informe contenant le résultat obtenu
Agent Passerelle
Construire un message SOAP de retour
Envoyer le message SOAP au client en attente

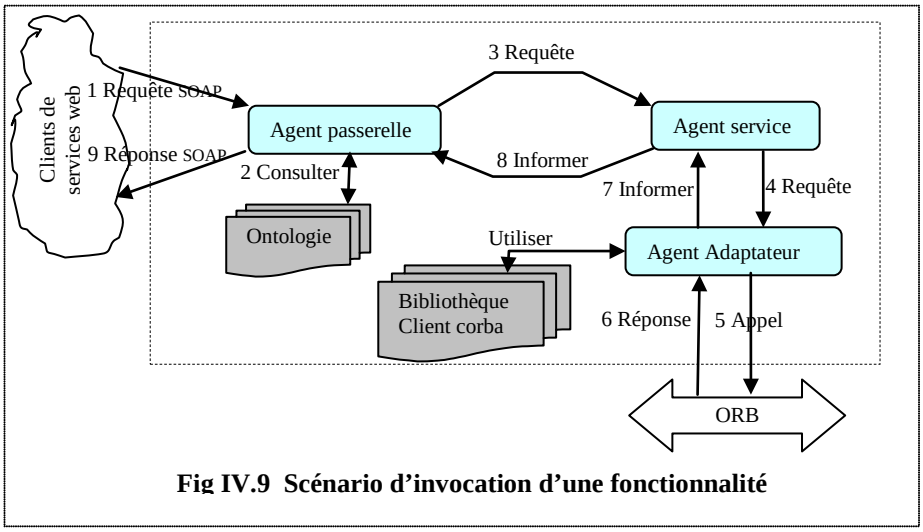


Fig IV.9 Scénario d'invocation d'une fonctionnalité

2.4.4. Invocations de fonctionnalité composite par un service web externe

La figure Fig IV.10 illustre l'invocation d'une fonctionnalité composite par un client de service web

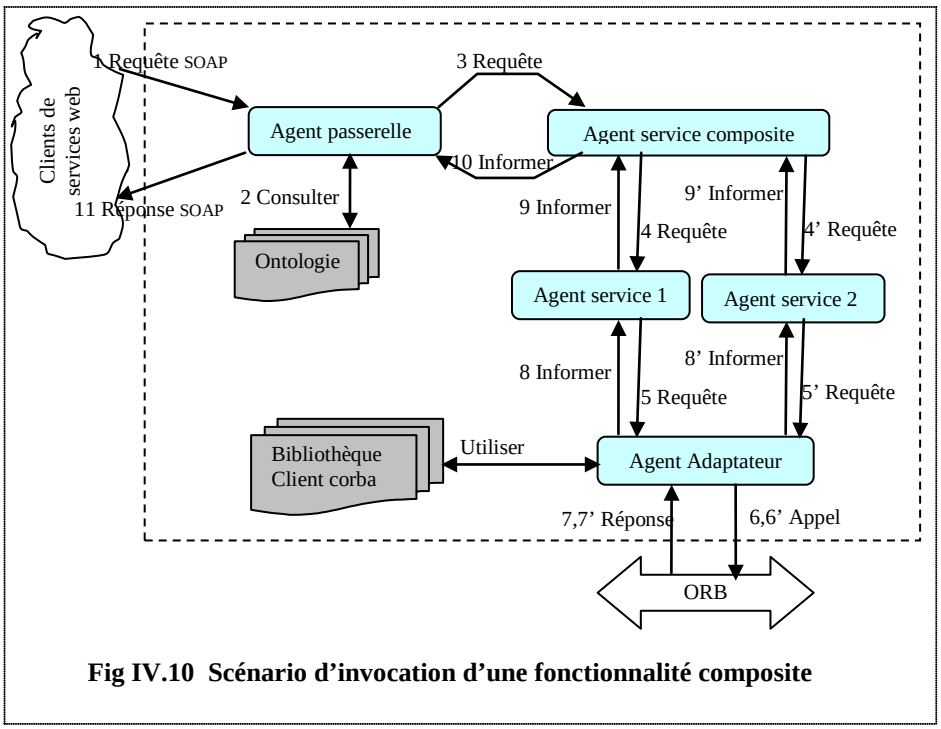


Fig IV.10 Scénario d'invocation d'une fonctionnalité composite

2.5. Base de données du système

Le schéma suivant correspond au modèle entité/association de notre système (Fig IV.11)

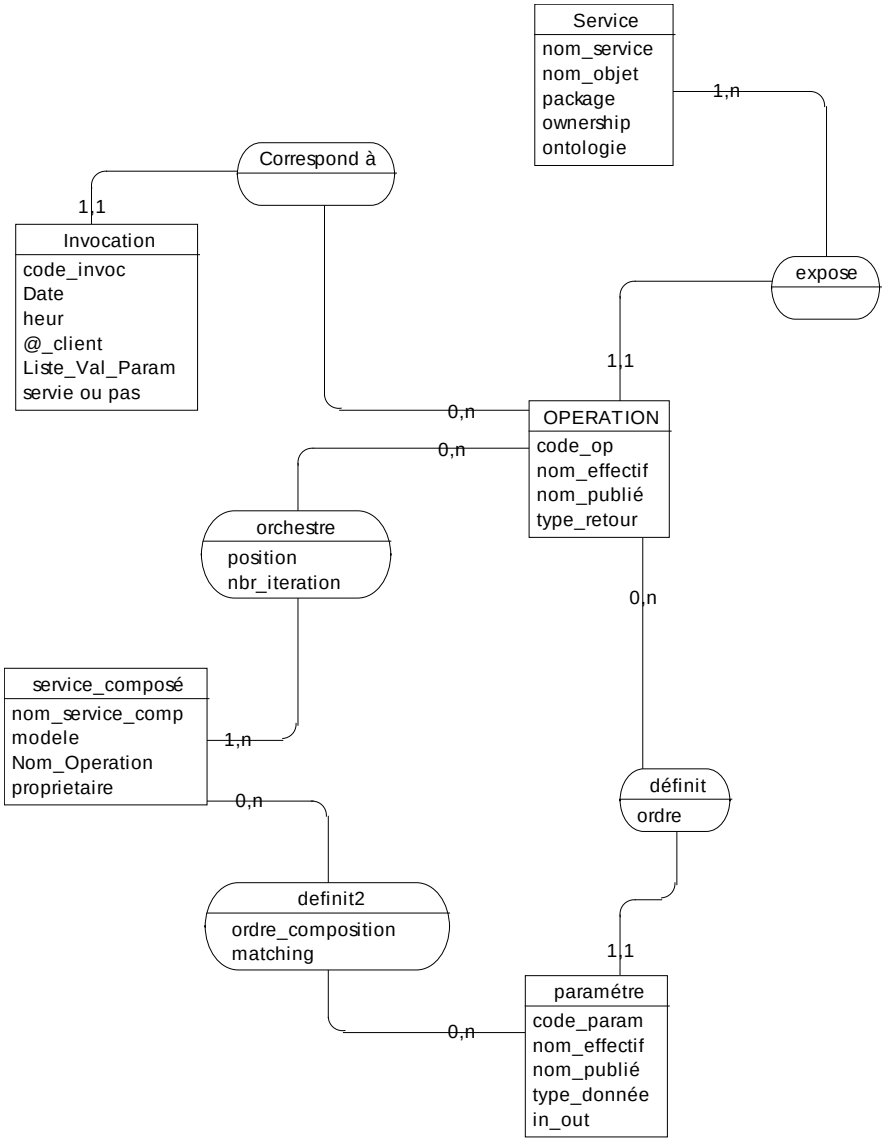


Fig IV.11 Modèle conceptuel de données

3. Conclusion

Nous avons présenté dans ce chapitre l'architecture conceptuelle de notre système d'encapsulation à base d'agents, où on a exposé l'architecture globale du système, le rôle et les responsabilités de chaque type d'agents et les scénarios possibles.

Nous avons également proposé des modèles de composition de fonctionnalités permettant de présenter de nouvelles fonctionnalités répondant aux besoins des utilisateurs du service

Chapitre V
Expérimentation

Chapitre V Expérimentation

1. Introduction

Afin de mettre en œuvre l'architecture proposée nous avons utilisé le framework **JADE** (Java Agent DEvelopment framework) pour l'implémentation de notre système multi agents, et le plug-in **WSIG** (Web Service Intégration Gateway) comme agent passerelle, conçu spécialement pour relier d'une manière transparente le domaine de services Web aux plateformes JADE en offrant la découverte bidirectionnelle et l'invocation à distance des services web par des agents JADE, et des services d'agents JADE par des services Web (Fabiob-Bellifemine, et al., 2007). Pour le déploiement des services web nous avons choisi le serveur Apache Tomcat5.5 (voir Fig V.1).

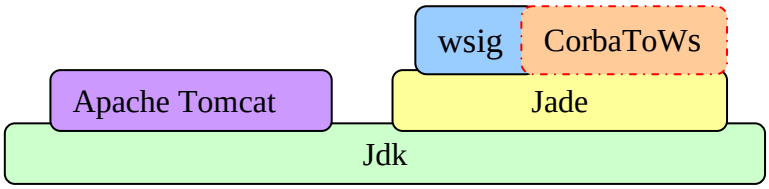


Fig V.1 Plateforme d'exécution du système CorbaToWs

2. Éléments de la Plateforme :

— **JDK (Java Development Kit) :** Un environnement de développement pour l'implémentation des applications, des applets, des composants en utilisant le langage de programmation Java (http://java.sun.com);

— **JADE** (Java Agent Development framework): une plateforme multi-agents développée initialement par le département de recherche et de développement de Telecom Italia s.p.a. Il est maintenant un projet de la communauté informatique, et est déstribué en tant que open source sous la licence LGPL pour le développement des applications et des SMA conforment au standards FIPA (Fabiob-Bellifemine, et al., 2007).

— **WSIG** (Web Service Integration Gateway) : est un produit additionnel de JADE conçu spécialement pour relier d'une manière transparente le domaine des services web aux plateformes JADE en offrant la découverte bidirectionnelle et l'invocation à distance des services web par des agents de JADE, et des services d'agents JADE par des services Web (Fabiob-Bellifemine, et al., 2007).

— **Apache Tomcat5.5 Servlet/JSP Container** : une plateforme pour le développement et le déploiement des applications web et des services web (http://jakarta.apache .org)

3. Interface CorbaToWS

3.1. Création d'agent service simple

La forme présentée dans la figure Fig V.2 correspond à la création d'un agent service simple et à sa publication en tant que service web. Dans l'exemple, l'agent correspond à un objet CORBA distant nommé 'résultat' dont le script idl est le suivant :

```
module scolarité
{
interface resultat
   {
      float  moyengenerale(in short matricule, in short annee );
      string  resultat(in short matricule, in short annee );
   };
};
```

L'utilisateur de notre système **''corbaToWs''** peut migrer des fonctionnalités CORBA en introduisant le chemin du fichier idl qui représente le contrat avec l'objet serveur de fonctionnalité, le nom du service, le nom du propriétaire de service (ownership) ainsi que le package (qui permet d'organiser les classes créées automatiquement par le système). En cliquant sur le bouton « *scanner l'idl* », le système extrait les méthodes et les paramètres d'entrée/sortie construisant l'interface du serveur ; l'utilisateur peut entrer les nouveaux noms des méthodes et des paramètres (qui seront utilisés pour la construction de l'interface wsdl du service web correspondant) ;

Enfin en cliquant sur le bouton « *créer le service* »**,** le système effectue les opérations suivantes :

- compiler le fichier idl à l'aide de l'utilitaire **IDLJ** pour créer le stub
- créer l'objet client CORBA utilisé pour l'invocation de l'objet distant
- compiler les fichiers générés à l'aide du compilateur **javac**
- générer l'ontologie de mise en correspondance utilisée d'une part par le WSIG pour générer l'interface WSDL et d'autre part pour la communication entre les agents du système
- créer et compiler la classe de l'agent service
- lancer l'agent service et déclencher l'enregistrement en DF et en UDDI

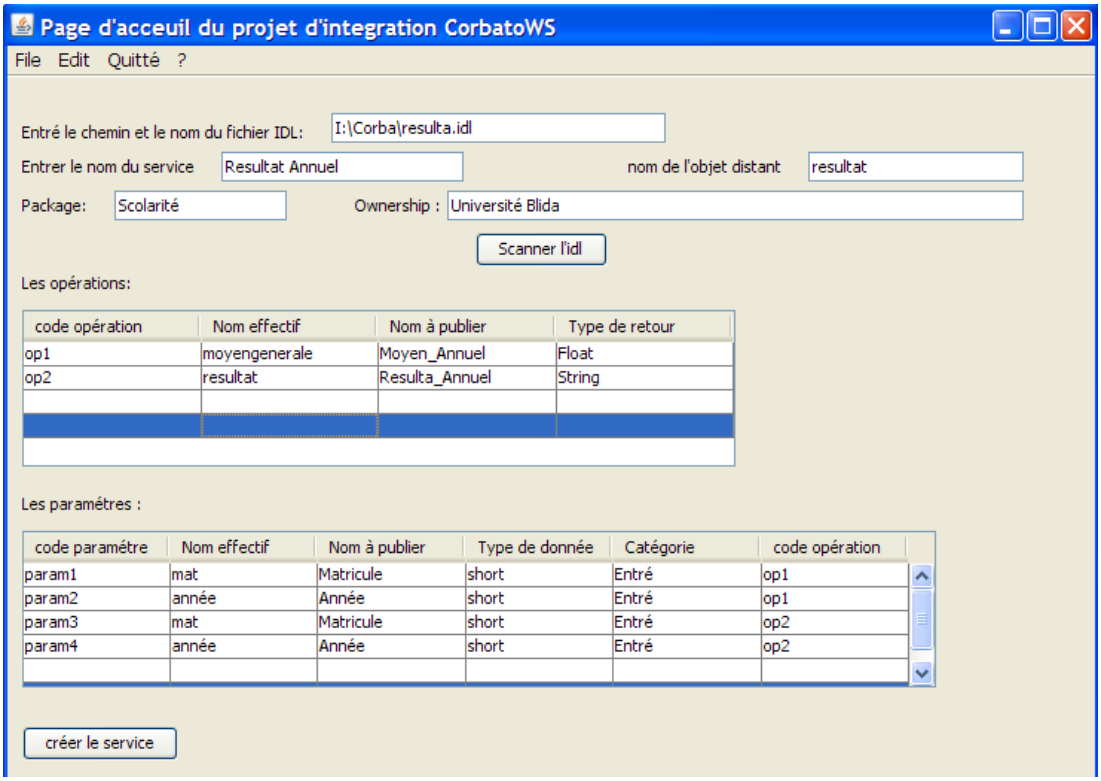


Fig V.2 Interface de création de service simple

Remarque **:**

- l'utilisateur a le choix d'entrer les identificateurs des méthodes et des paramètres manuellement sans passer par le scan automatique du fichier idl.

— La récupération de l'adresse IOR (Interoperable Object Reference) de l'objet serveur se fait à l'aide du service de nommage.

Le code suivant correspond à l'ontologie du service '*resultat annuel*', ce code est généré et compilé automatiquement (ainsi que les classes correspondantes aux actions incluses dans l'ontologie).

```
package Ontology;

import jade.content.onto.*;
import jade.content.schema.AgentActionSchema;
import jade.content.schema.PrimitiveSchema;
/**
 *
 * @author Fayçal
 */
public class resultaOntology extends Ontology {
 public static final String ONTOLOGY_NAME = "resultaOntology";
    public  static Ontology theInstance = new resultaOntology();
    public static Ontology getInstance() {
        return theInstance;
    }
    private static final String moyenne = "Moyen_Annuel";
    public static final String resultat = "Resulta_Annuel";
    private static final String matricule = "matricule";
    public static final String année = "Année Scolaire";
     public resultaOntology(){
        super(ONTOLOGY_NAME,BasicOntology.getInstance());
        try{
        add(new AgentActionSchema(moyenne), moyenne.class);
        AgentActionSchema as1 = (AgentActionSchema) getSchema(moyenne);
            as1.add(matricule, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            as1.add(année, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            as1.setResult((PrimitiveSchema)getSchema(BasicOntology.FLOAT));

        add(new AgentActionSchema(resultat), resultat.class);
        AgentActionSchema as2 = (AgentActionSchema) getSchema(resultat);
            as2.add(matricule, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            as2.add(année, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            as2.setResult((PrimitiveSchema)getSchema(BasicOntology.STRING));
        }catch (OntologyException oe) {
            oe.printStackTrace();
        }
    }
}
```

L'agent service créé s'enregistre au lancement selon le code suivant généré automatiquement

```
public class resultatagent extends Agent {
     static resultat resultaimp;
     private SLCodec codec = new SLCodec();
    private static final String WSIG_FLAG= "wsig";
    @Override

    protected void setup(){
        System.out.println("Hello I'm the resultat scolaire Agent and I'm ready to receive yc
        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(FIPAManagementOntology.getInstance());
        getContentManager().registerOntology(resultaOntology.getInstance());
        // Prepare a DFAgentDescription
        DFAgentDescription dfad = new DFAgentDescription();
        dfad.setName(this.getAID());
        dfad.addLanguages(codec.getName());
        dfad.addProtocols(FIPANames.InteractionProtocol.FIPA_REQUEST);
        ServiceDescription sd;
        sd = new ServiceDescription();
        sd.addLanguages(codec.getName());
        sd.addProtocols(FIPANames.InteractionProtocol.FIPA_REQUEST);
        sd.setType("Scolarité");
        sd.setOwnership("haniche Fayçal");
        sd.addOntologies(resultaOntology.getInstance().getName());
        // WSIG properties
        sd.addProperties(new Property(WSIG_FLAG, "true"));
        // Service name
        String wsigServiceName = "Resultat";
     //   log.info("Service name: "+wsigServiceName);
        sd.setName(wsigServiceName);
        dfad.addServices(sd);
```

L'invocation effective de l'opération « *moyengenerale* » de l'objet distant « *resultat* » se fait à l'aide du code suivant généré automatiquement après la génération du stub

```
 private void serveMoyAction(moyenne moyenne, Action actExpr, ACLMessage msg) {
     String[] args=null;
     short mat= moyenne.getmatricule().shortValue();
     short an= moyenne.getannée().shortValue();
     ACLMessage reply=msg.createReply();
     reply.setPerformative(ACLMessage.INFORM);
      try{
      ORB orb = ORB.init(args,null);
      org.omg.CORBA.Object objRef=orb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      String name = "resultat";
      resultaimp =resultatHelper.narrow(ncRef.resolve_str(name));
      float moy= resultaimp.moyengenerale(mat,an);
      ContentElement ce = null;
      ce = new Result(actExpr,moy);
      getContentManager().fillContent(reply, ce);
      send(reply);
         }
        catch (Exception e){
        System.out.println("ERROR : " + e) ;
        e.printStackTrace(System.out);
        }

 }
```

3.2. Création d'agent service composite

La figure Fig V.4 correspond à la création d'un agent service composite et à sa publication en tant que service web, dans l'exemple une composition par le modèle

séquenceur est présentée ; l'utilisateur choisit les fonctionnalités à inclure dans la composition (dans un ordre logiquement bien défini) par l'indication de l'agent service et de l'opération correspondante à chaque fonctionnalité. Après, en cliquant sur les boutons '*Editer les paramètres'*, les paramètres des opérations choisies ainsi que leur type de retour apparaissent ; l'utilisateur peut alors définir un nouvel ordre pour les paramètres d'appels de la nouvelle fonctionnalité composite ; il doit aussi indiquer les paramètres de la deuxième opération composante qui prendront comme valeur le résultat de la première pendant l'exécution de la fonctionnalité composite, c'est paramètres seront indiqués dans l'interface par la propriété '*Matching'* mise à '*oui'*. Ici une condition doit être vérifiée pour que la composition puisse réussir, et qui est : le type de paramètre de *Matching* doit être égale au type de résultat mis en correspondance ou l'englober. Dans la figure Fig V.3 la relation suivante doit être vérifiée $TypeR1 \subseteq TypeP2.1$

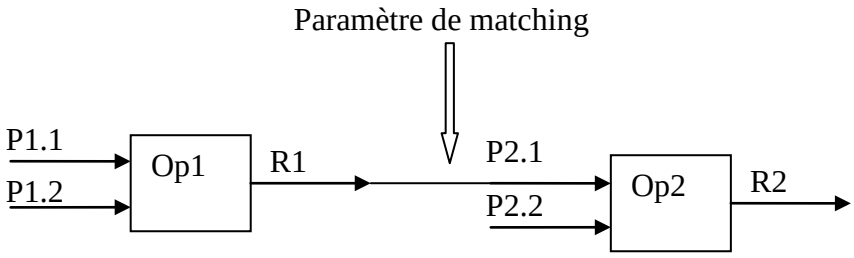


Fig V.3 Assortiment des paramètres dans la composition

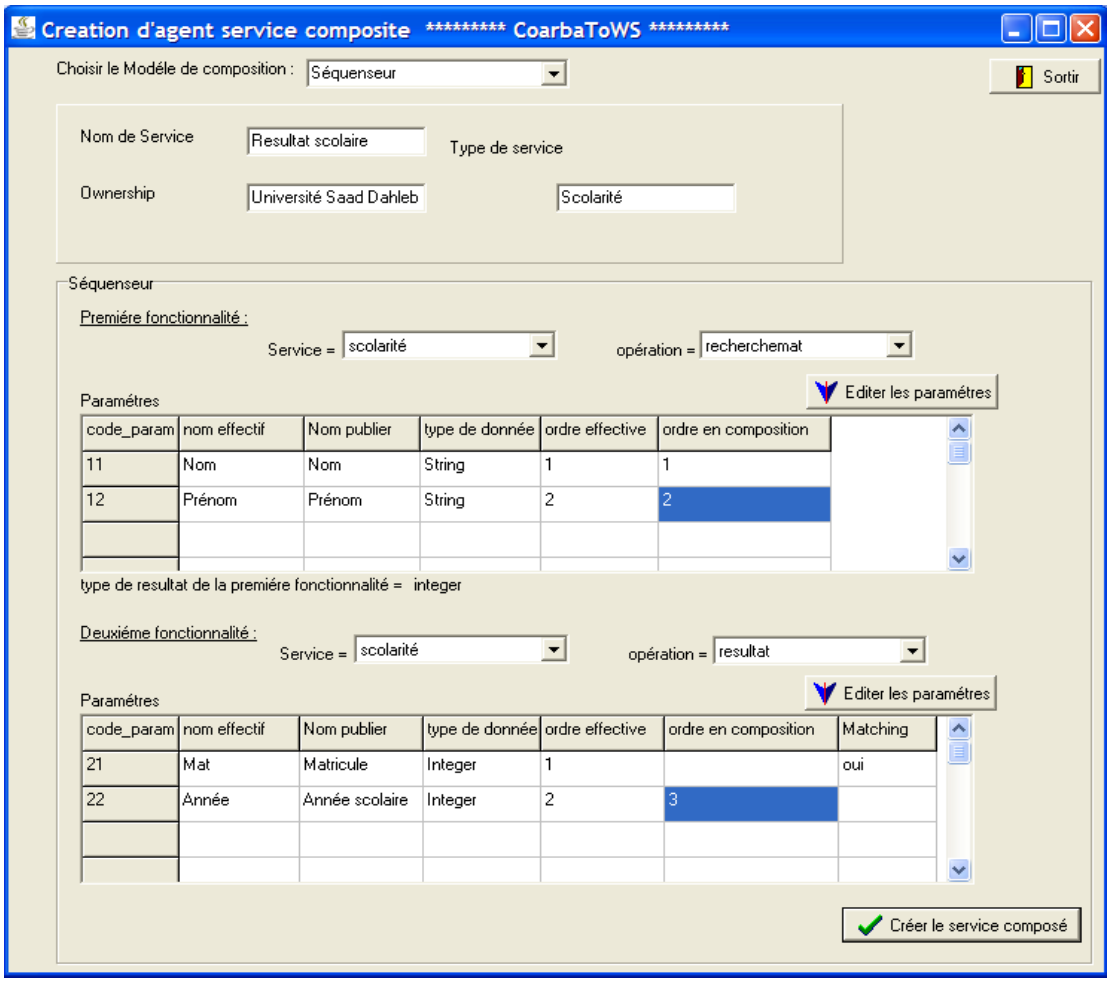


Fig V.4 Interface de création de service composé

Enfin en cliquant sur le bouton « *créer le service composé* »**,** le système effectue les opérations suivantes :

- Générer et compiler l'ontologie (cette ontologie hérite des ontologies des deux services composants avec définition des termes de la nouvelle opération à publier)
- Créer et compiler la classe de l'agent service composé
- Lancer l'agent service et déclencher l'enregistrement en DF et en UDDI

4. Gestion des agents :

Les agents du système ainsi que ceux de la plate forme JADE seront gérés par l'interface graphique fournie par l'agent du système JADE nommé '*Remote Monitoring Agent*' (RMA) (Fig V.5); cette interface offre plusieurs actions sur les agents et sur la plateforme, elle permet de bloquer, tuer, charger, lancer un agent,etc.

Le RMA permet notamment de lancer d'autres agents de débogage, en particulier :

Le *DummyAgent* : qui est un outil simple et très utile pour inspecter les échanges de messages entre les agents de JADE (Fig V.6).

Le *SnifferAgent* : qui est un outil créé pour dépister les messages échangés dans un environnement jade. Le Sniffer est utile au débogage des comportements d'agents (Fig V.7).

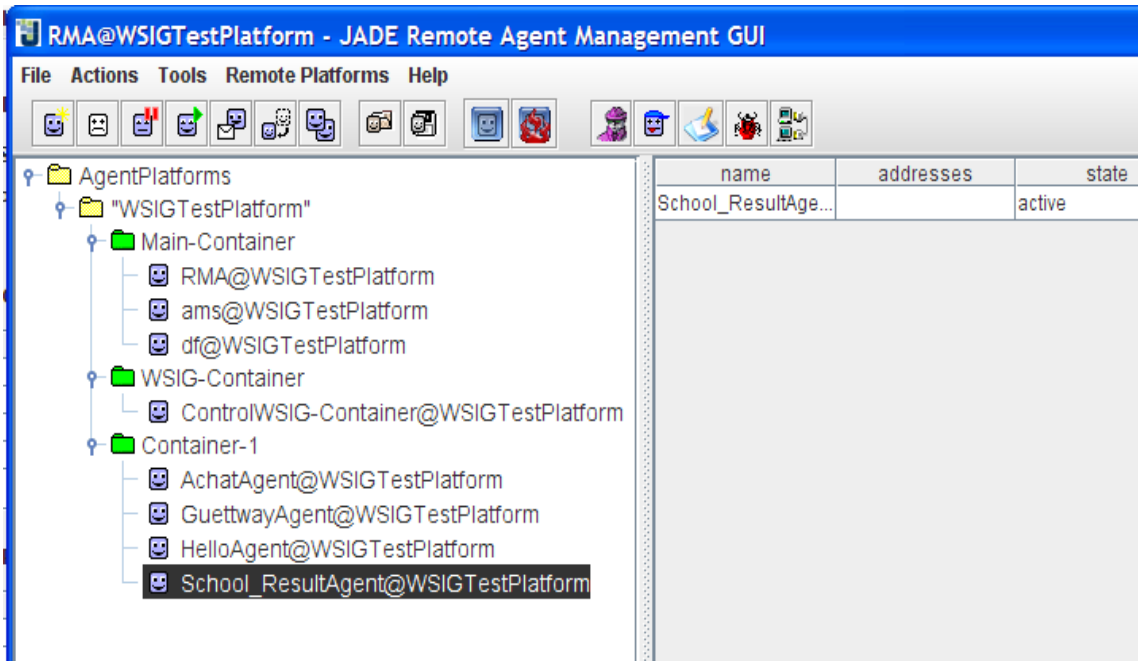


Fig V.5 Gestion des agents par le GUI de l'agent RMA de JADE

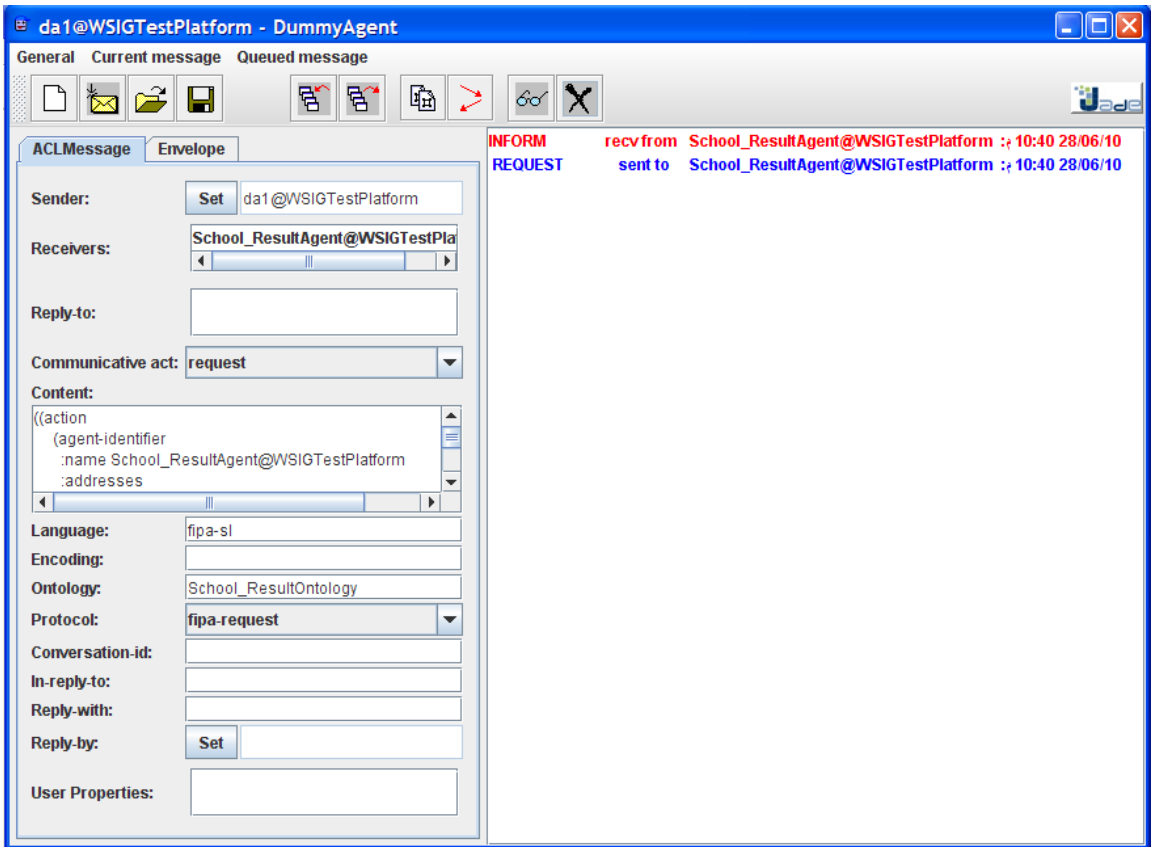


Fig V.6 Interface du DummyAgent

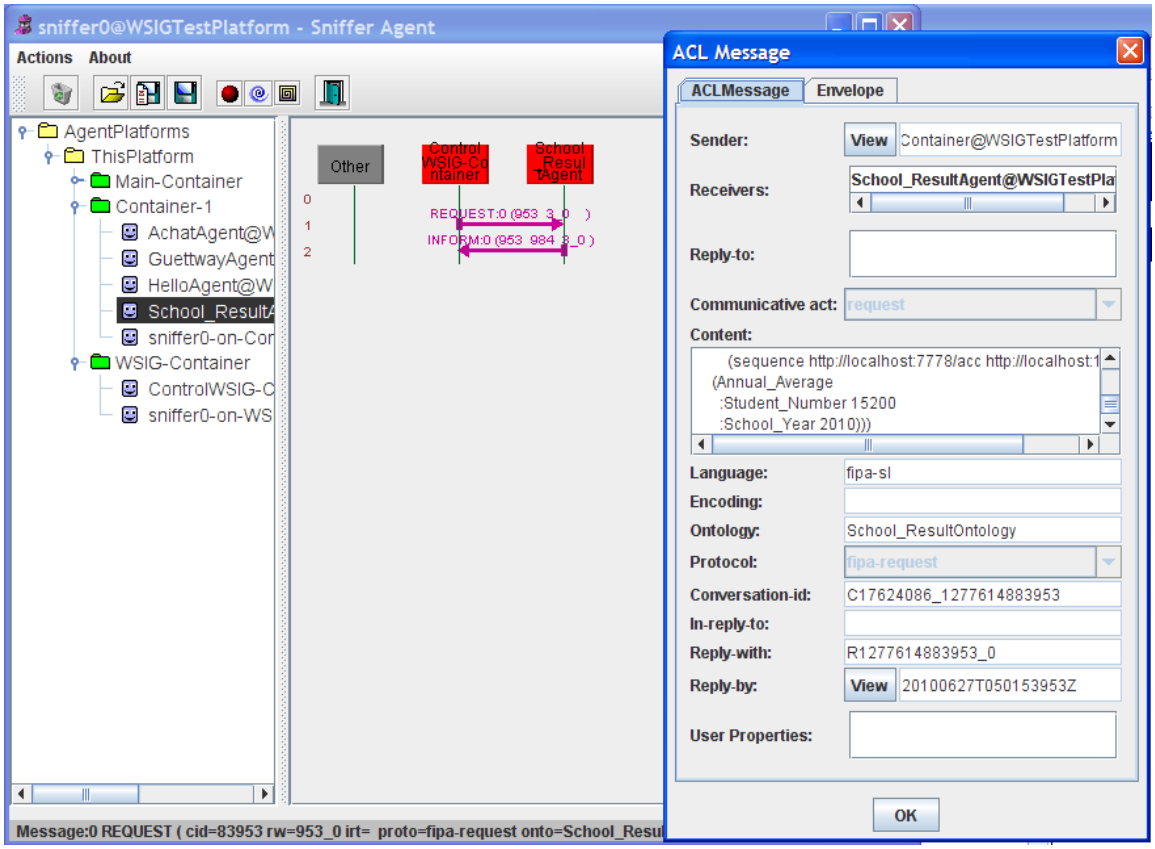


Fig V.7 Interface du SnifferAgent

5. Visualisation et test des services web:

Le WSIG est muni d'une interface graphique d'utilisateur web d'administration à l'aide de laquelle il est possible de vérifier les services web exposés. Si l'installation standard est faite correctement, le GUI d'administration de WSIG peut être atteint à l'URL < http://localhost:8080/wsig >.

Comme présentée dans la figure Fig V.8 la page principale montre la liste des services web exposés et des paramètres de configuration.

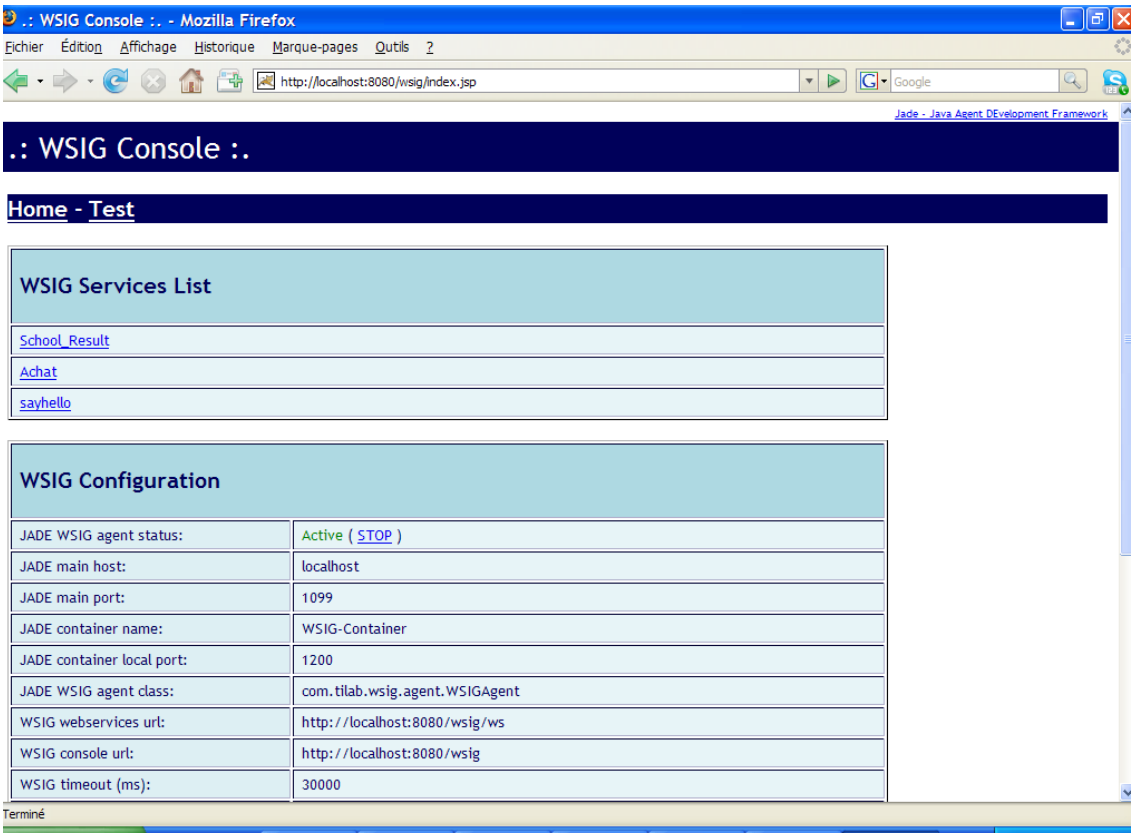


Fig V.8 Page web d'administration de WSIG

En cliquant sur l'un des services de la liste, les informations qui lui sont appropriées seront automatiquement exposées (nom du service, nom de l'agent fournisseur du service, l'ontologie référencée, le préfixe, la liste d'opérations et le lien vers le code WSDL correspondant) (voir Fig V.9).

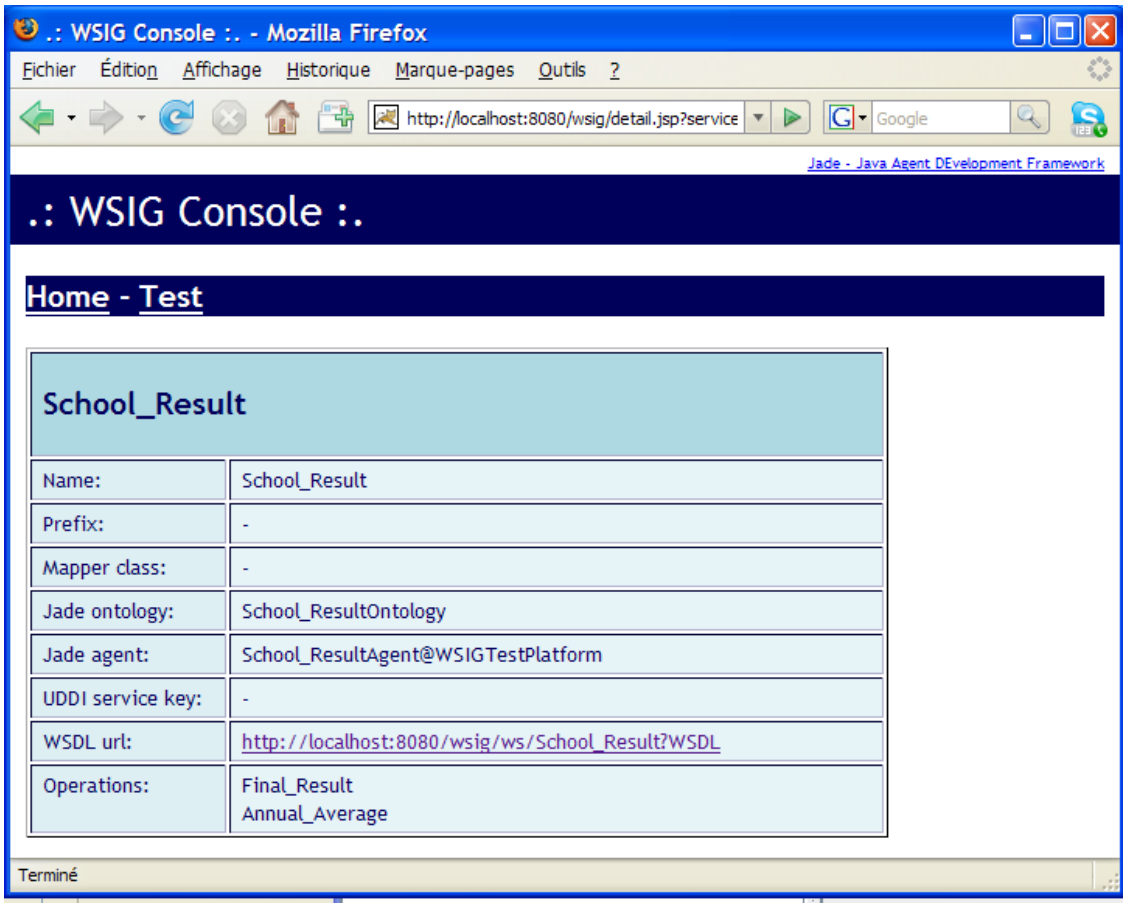


Fig V.9 le GUI d'administration de WSIG : détails du service

Le GUI de WSIG inclut également une page de test au moyen de laquelle il est possible de déclencher des demandes SOAP vers un web service exposé. Ceci est fait en collant un message de requête SOAP dans la région des requêtes puis en cliquant sur le bouton 'Send'. Le message de réponse apparaîtra alors dans la région de réponse SOAP (voir Fig V.10).

L'exemple de message SOAP présenté correspond à l'invocation de l'opération 'Annual_Average' du service 'School_Result', la requête SOAP est transmise via le serveur tomcat vers l'agent WSIG qui la traduit en message ACL en utilisant l'ontologie 'school_ResultOntology', le message Acl d'invocation de la fonctionnalité est envoyé à son tour vers l'agent 'School_ResultAgent' qui fait un appel effectif de la fonctionnalité présentée par le serveur CORBA exécuté dans la même machine ou dans une machine distante, le message de retour suivra le même chemin et sera enfin transformé en réponse soap affichée dans la région SOAP response de la page de test.

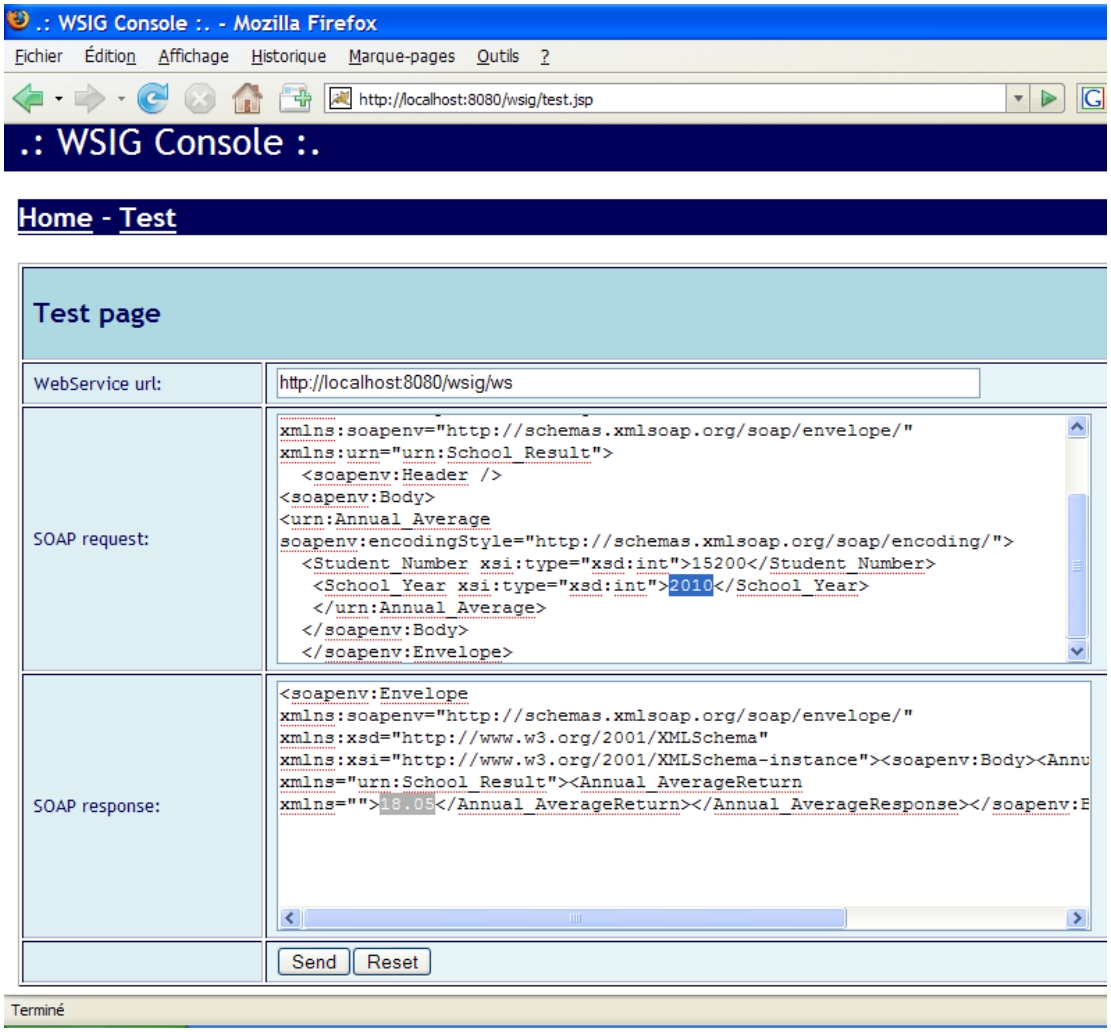


Fig V.10 Page de test des services créés

6. Conclusion

Dans ce chapitre une mise en œuvre de l'architecture du système d'encapsulation a été réalisée en bénéficiant des outils proposés par JADE. Pour la gestion des agents on utilise l'agent RMA de JADE, comme annuaire on utilise l'agent directory facilitator et pour publier les agents service comme services web on utilise le produit additionnel WSIG comme agent passerelle. Il nous a suffi d'implémenter une interface de générations automatiques des agents service simple ou composite, des clients CORBA et des ontologies, et permettant d'initialiser l'exécution et la publication des agents service.

Cette partie du travail nous a permis de s'initier dans plusieurs technologies telles que corba sous java, JADE, WSIG, SGBD Mysql, l'EDI netbeans et le serveur Apache Tomcate.

Conclusion Générale

Conclusion Générale

De nos jours, la définition et la validation des approches pour l'exportation des applications existantes vers de nouvelles architectures orientées service est un sujet de recherche très sensible. Ce travail a traité le problème de migration des fonctionnalités des applications CORBA existantes vers les architectures SOA par une approche black-box à base de système multi-agents. Cette approche considère que les fichiers **IDL** de création des objets CORBA sont disponibles, et permet de créer les clients CORBA, les exposer sous forme de capacité d'agents et enfin les publier en tant que service web. Dans ce rapport sont présentés l'architecture logique du système d'encapsulation, le processus de migration permettant la conception des encapsulateurs ainsi que l'outil de sa mise en œuvre.

Il est à noter que l'architecture de notre wrapper peut être utilisée pour l'encapsulation d'autres types d'applications sauf que, la manière dans laquelle l'agent adaptateur invoque effectivement les fonctionnalités du système legacy diffère d'un type à un autre ; ici la condition essentielle à satisfaire, est que d'une part, le format d'échange pour l'invocation de chaque fonctionnalité à encapsulée soit connue et, que d'autre part, le protocole d'invocation de ces fonctionnalités soit supporté par le langage de programmation utilisé pour le développement du système d'encapsulation.

Enfin, et comme travaux futurs, nous pensons à prolonger l'applicabilité de notre approche pour prendre en charge les fonctionnalités des applications distribuées basées sur des technologies autres que la technologie CORBA.

Annexes

Annexe A: Les services web.

Annexe B: Architecture CORBA.

Annexe C: systèmes Multi-agents.

Annexe D: JADE (Java Agent DEvloppement framework

Annexe A. Les services Web

1. Définition

Un service Web est une application logicielle identifiée par un URI (Universal Resource Identifier), dont les interfaces et associations peuvent être définies, décrites et découvertes par des méthodes XML, et qui peut interagir directement avec d'autres applications en utilisant des messages XML via les protocoles Internet standards (http://www.w3c.org/TR.).

Les services Web sont complètement indépendants des plates-formes d'exécution et des langages de programmation sous-jacents. Ils peuvent ainsi être développés en utilisant n'importe quel langage de programmation, et être déployés sur n'importe quelle plate-forme (du plus petit dispositif au super calculateur). D'autre part les services Web sont des composants logiciels fournis sur Internet, ils utilisent l'infrastructure existante, notamment les protocoles qui ont fait le succès du Web "IP et HTTP", et sont accessibles depuis n'importe où via des URI s. Les services Web sont basés sur le standard XML (eXtensible Markup Language) pour décrire et échanger les données d'une manière uniforme. Ils séparent la description des services de leurs implémentations, en définissant des interfaces dans le langage WSDL (*Web Services Description* L*anguage*) et communiquent en utilisant des protocoles standards tels que SOAP (*Simple Object Access Protocol*).

Utilisés dans un premier temps dans le B2B *Business-to-Business* pour permettre aux entreprises de communiquer entre-elles et avec leurs clients, les services Web se sont désormais ouverts à une utilisation plus générique. N'importe quel composant matériel ou application logicielle peut donc s'exposer comme un service Web afin de faciliter son déploiement, son utilisation et son intégration par d'autres applications.

2. Architecture de référence

Les efforts de recherche et de développement récents autour des services Web ont conduit à un certain nombre de spécifications qui définissent aujourd'hui l'architecture

de référence des services Web. L'architecture de référence des services Web telle que la montre la figure Fig.A 1 s'articule autour des trois éléments suivants :

— Le fournisseur de services qui correspond au propriétaire du service. Il est constitué de la plate-forme d'hébergement du service.

— Le client qui correspond au demandeur de services. Il est constitué de l'application qui va rechercher et invoquer un service. L'application cliente peut être elle même un service Web.

— L'annuaire de services qui correspond à un registre de description offrant des facilités de publication de services à l'intention des fournisseurs, et des facilités de recherche de services à l'intention des clients.

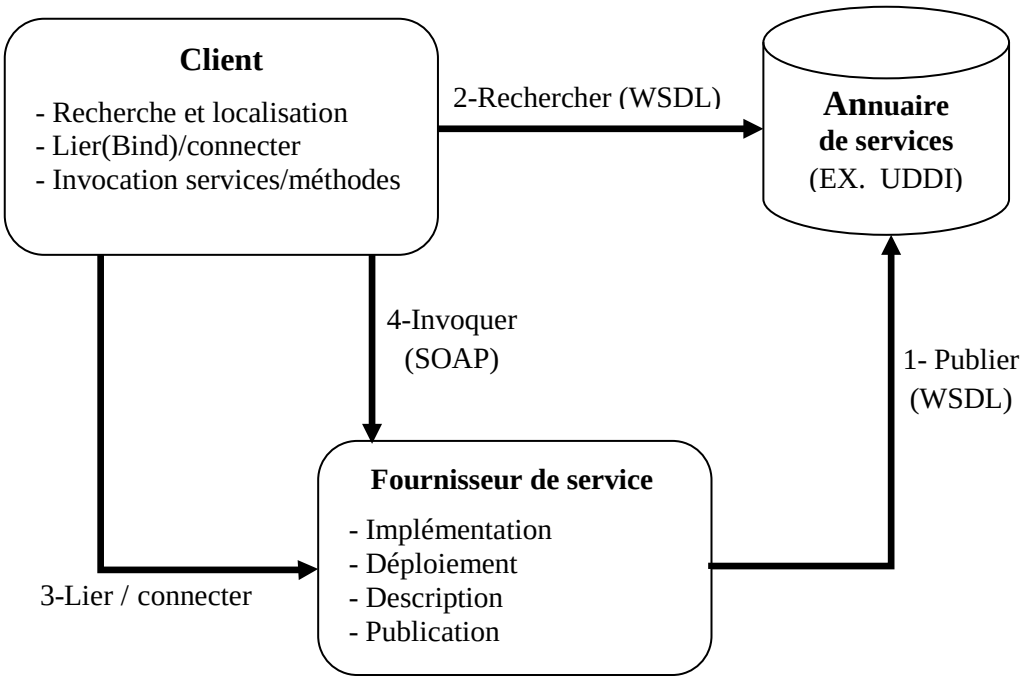


Fig.A 1 Architecture des Services Web

Pour garantir l'interopérabilité des trois opérations précédentes (publication, recherche et liaison/ invocation), des propositions de standards ont été élaborées pour chaque type d'interaction (Toumani).

2.1. WSDL : Web Services Description Language

WSDL est un langage qui permet de décrire les services web, et en particulier, les interfaces des services web. Ces descriptions sont des documents XML. WSDL décrit un service web en deux étapes fondamentales : une abstraite et une concrète. Dans chaque étape, la description utilise un nombre de constructions pour favoriser la réutilisation de la description et pour séparer les préoccupations de conception indépendantes (Fig.A 2) (G.alonso, et al., 2003)**.**

Au niveau abstrait, WSDL décrit un service Web en termes des *messages* qu'il envoie et reçoit; les *messages* sont décrits de façon indépendante d'un format spécifique en utilisant un système de *types*, typiquement un schéma XML. La partie abstraite est composée de définitions de *"port type"* qui sont analogues aux interfaces dans les IDLs (Interface description language) d'un "middleware" traditionnel. Chaque *"port type"* est une collection logique d'opérations. Une *"operation"* associe un modèle d'échange de message à un ou plusieurs messages. Un *message* est une unité de communication avec un service web. Il représente les données échangées dans une unique transmission logique.

Un modèle d'échange de messages identifie l'ordre et la cardinalité des messages envoyés et/ou reçus. Une interface groupe un ensemble d'opérations.

Au niveau concret, un *"binding"* indique des détails de format de transport pour une ou plusieurs interfaces. Un *"endpoint"* (i.e. point final) associe une adresse de réseau à un "*binding*". Finalement, un service groupe un ensemble d'*endpoints* qui implémente une interface commune.

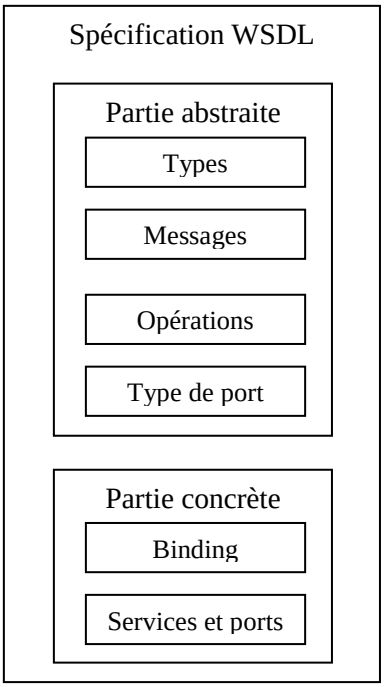


Fig.A 2 La spécification d'un service Web avec WSDL

2.2. UDDI: Universal Description Discovery and Integration

L'objectif primaire d'UDDI est la spécification d'un canevas pour décrire et découvrir des services Web. Le noyau d'UDDI travaille avec la notion de "business registry", qui est un service sophistiqué de noms et répertoires. Plus précisément, l'UDDI définit des structures de données et APIs pour publier les descriptions des services dans le registre et pour interroger le registre afin de chercher des descriptions publiées. Parce que les APIs de l'UDDI sont aussi spécifiées en WSDL avec une attache SOAP, le registre peut être invoqué comme un service Web (en conséquence, ses caractéristiques peuvent être décrites dans le même registre, comme un autre service).

Les spécifications du « registre » UDDI ont deux buts principaux en ce qui concerne la découverte d'un service: Le premier est de soutenir les développeurs dans la découverte d'informations sur les services et le deuxième est de permettre la liaison dynamique et en conséquence habilite les clients pour interroger le registre et obtenir des références aux services d'intérêt.

De plus, pour définir un registre de services Web, l'UDDI adresse aussi le concept d'Universal *Business Registry (UBR)*. En fait, le but initial du consortium a été de

soutenir des répertoires mondiaux où chacun pourrait publier des descriptions de service et où chacun pourrait interroger ces répertoires pour trouver les services d'intérêt (Rosa-Roserio, 2004).

2.3. SOAP : Simple Object Access Protocol

Actuellement dans sa version 1.2 le SOAP est un protocole léger, destiné à l'échange d'informations structurées dans un environnement distribué. Il utilise des technologies XML pour définir une structure de messages pouvant être échangés sur divers protocoles sous-jacents (ex. HTTP, SMTP). Cette structure a été conçue pour être indépendante de tout modèle de programmation et autre sémantique spécifique d'implémentation.

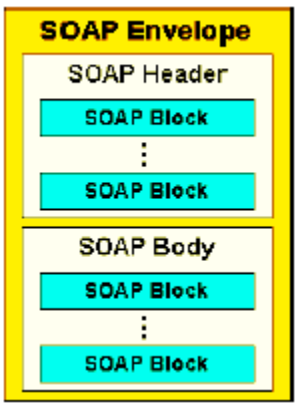


Fig.A 3 Structures d'un message SOAP

Un message SOAP est une transmission unidirectionnelle entre des noeuds SOAP (d'un émetteur vers un récepteur SOAP), mais les messages SOAP peuvent être combinés par les applications pour implémenter les séquences plus complexes d'interactions (ex. requête-réponse, échanges multiples "conversationnels"). Un message SOAP contient deux sous éléments à l'intérieur de l'élément Enveloppe externe: un élément Header (en-tête) et un élément Body (corps) (voir Fig.A 3). Les contenus de ces éléments sont définis par l'application et ne font pas partie de la spécification de SOAP (Benmokhtar, 2003-2004).

3. Architecture étendue

Différentes extensions de l'architecture de référence ont été proposées dans la littérature. Le groupe architecture du W3C travaille activement à l'élaboration d'une architecture étendue standard.

Une architecture étendue est constituée de plusieurs couches se superposant les unes sur les autres, d'où le nom de pile des Web services. La figure Fig.A 4 décrit un exemple d'une telle pile (Toumani, et al.).

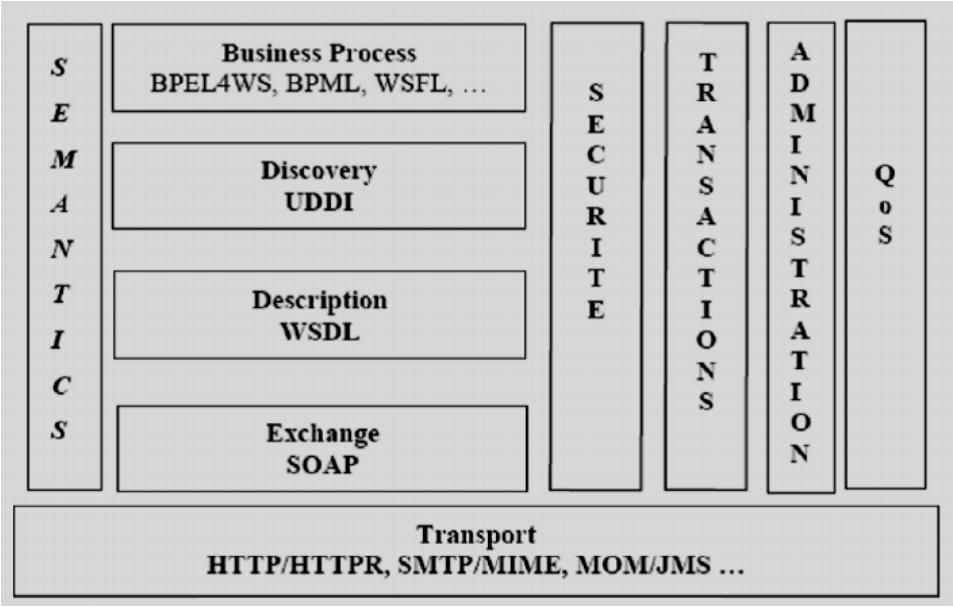


Fig.A 4 Pile des web services sémantiques

La pile est constituée de plusieurs couches, chaque couche s'appuyant sur un standard particulier. On retrouve, au-dessus de la couche de transport, les trois couches formant l'infrastructure de base décrite précédemment. Ces couches s'appuient sur les standards émergents SOAP, WSDL et UDDI.

Comme mentionné précédemment, l'infrastructure de base définit les fondements techniques permettant de rendre les processus métier (business-process) accessibles à l'intérieur d'une entreprise et au-delà même des frontières d'une entreprise. Dans ce contexte deux types de couches permettent de la compléter : (i) les couches dites

transversales (exple : sécurité, administration, transactions et qualité de services (QoS)) rendent viable l'utilisation effective des Web services dans le monde industriel ;

(ii) une couche Business-process permettant l'utilisation effective des Web services dans le domaine du e-business. Concernant la Business-process certaines préoccupations ont été relevées :

• Comment les processus métier (business-process) peuvent-ils être représentés comme des Web services ?

• Nécessité de décrire comment les Web services sont utilisés pour implanter les activités d'un business-process.

• Les problèmes de composition de service, *i.e.* quel(s) partenaire(s) va (vont) exécuter quelle(s) partie(s) d'un business-process ?

Différents auteurs de la communauté de recherche s'accordent sur la nécessité de spécifier le comportement externe de chaque partie impliquée dans le protocole d'intégration de processus (partie publique) sans pour autant révéler leurs implémentations internes (partie privée). Deux raisons justifient cette séparation :

• Les entreprises ne tiennent pas forcément à révéler leurs prises de décisions internes et souhaitent préserver la confidentialité de leurs données.

• La séparation publique-privée permet de modifier la partie privée indépendamment de la partie publique.

A cet effet, différents langages ont été proposés dans le but de décrire le processus public d'un service (*e.g*, WSCL) ou la spécification, de manière procédurale, de la composition de services (*e.g*, BPML (http://www.bpmi.org/), BPEL4WS (http://www106.ibm.com/¬ developerworks/library/ws-bpel/)).

Annexe B: Architecture CORBA

1. Introduction

CORBA est une spécification, c'est-à-dire un standard qui décrit une architecture permettant de faire collaborer des applications disposées sur des machines, des environnements, des langages différents. Elle a été spécifiée pour la première fois en 1992 et revue en 1995 pour sa version 2. CORBA, est une architecture basée sur un bus à requêtes lequel se nomme dans la terminologie CORBA : ORB (Object Request Broker, bus à requêtes objet), c'est l'élément clé de cette architecture qui est en charge d'assurer les collaborations entre applications.

A la figure Fig.A 5, nous présentons l'architecture générale de CORBA dans laquelle se dégagent plusieurs éléments en gravitation autour de l'ORB (Rousset, et al., 1999).

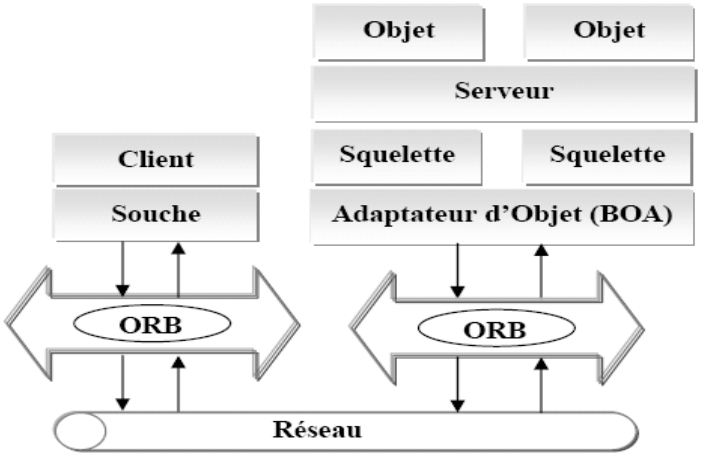


Fig.A 5 Architecture générale de CORBA

2. L'ORB (Object Request Broker)

L'ORB est le coeur de l'architecture CORBA. Il est responsable de la mise en relation et de la communication entre tous les éléments présents dans cette architecture distribuée. Il prend en charge le dialogue entre les objets serveurs et les différents clients qui s'y connectent. Un ORB est en fait réalisée sous forme d'une bibliothèque de fonctions qui permettent de mettre en oeuvre les coopérations entre clients et serveurs. Pour utiliser un

environnement CORBA, il faut acheter auprès d'un fournisseur un ORB qui sera fourni avec tout un ensemble d'outils nécessaires pour mettre en œuvre une application répartie. L'OMG spécifie également un ensemble d'applications se branchant sur l'ORB, qui remplissent des fonctions très utiles. Ces applications sont appelées « services » et ont pour objectifs de fournir toutes les fonctionnalités nécessaires au développement d'applications réparties.

3. Les services CORBA

Les services CORBA ont été définis dans le but de simplifier les développements puisqu'ils normalisent des sous ensembles répétitifs que l'on retrouve dans la plupart des applications. Les services spécifiés par l'OMG sont actuellement au nombre de dix-sept dont quatre sont liés à la technologie des agents mobiles et que nous allons présenter ci-après :

3.1. Service de noms (Naming Service)

Appelé aussi COSnaming dans la terminologie CORBA, ce service propose l'association d'un ou plusieurs noms logiques aux objets CORBA d'une application et permet de constituer un véritable annuaire ou catalogue d'objets distribués. Il est constitué d'un graphe de noms généralement alimenté par les programmes serveurs, dans lequel les clients peuvent naviguer et retrouver une ou plusieurs instances d'objets CORBA.

3.2. Service de cycle de vie (Life Cycle Service)

Ensemble d'outils pour la gestion d'un objet durant toute sa durée de fonctionnement. Il s'agit d'opérations destinées à créer, à copier, à déplacer ou à supprimer des objets.

3.3. Externalisation (Externalization)

Ce service repose sur la gestion de flux d'objets. Lorsque les objets sont placés dans un flux, on parle d'externalisation. Quand ils sont chargés à partir de ce flux, cette opération est appelée internalisation. Ce service est une solution qui favorise l'interopérabilité.

3.4. Service de sécurité (Security service)

Il fournit tous les éléments pour sécuriser l'environnement CORBA en prenant par exemple l'authentification, l'intégrité et la confidentialité.

4. Principe de fonctionnement

Nous allons maintenant détailler les différents éléments au cœur de la norme CORBA :

4.1. IDL (Interface Definition Language)

Développer des applications distribuées flexibles sur des plates-formes hétérogènes nécessite une séparation interface/implémentation(s). L'IDL (Interface Definition Language) aide à accomplir cette séparation. En effet, il est à la base de l'interopérabilité de CORBA. Il s'agit d'un langage purement descriptif qui permet, lors de la modélisation et de l'analyse de l'application, de définir les échanges et les interactions entre le client et le serveur. L'IDL introduit un nouveau concept au sein des langages objets : l'interface. Une interface est constituée d'un ensemble de prototypes de méthodes permettant de définir une vue fonctionnelle sur un objet. Cette vue sera exposée par l'objet serveur afin que le client puisse interagir avec lui. Un client ne manipule un objet serveur que par le biais de son interface. La description d'une interface CORBA permet de définir un comportement, un contrat entre les clients et les serveurs sans pour autant préjuger de la façon dont ils l'implémentent. La règle étant : un serveur doit implémenter une interface, un client ne peut utiliser l'objet qu'au travers de son interface.

4.2. Les module Souche et Squelette

Dans l'architecture CORBA, les communications entre un client et un serveur sont prises en charge par l'ORB avec deux modules spécifiques, appelés stub (souche) pour la partie cliente et Skelton (squelette) au niveau serveur. Ces deux modules sont générés par le pré-compilateur IDL. Les souches IDL construisent des requêtes, qui vont être transportées par le bus, puis délivrées par celui-ci aux squelettes IDL qui les délégueront aux objets.

4.3. BOA (Basic Object Adapter)

Le BOA *(Basic Object Adapter)* ou « adaptateur d'objet de base », est l'outil qui permet l'interaction entre l'ORB et les différents objets CORBA implémentés du côté serveur. Le BOA va donc offrir des fonctionnalités pour la gestion des implémentations d'objets et permet aussi de spécifier la stratégie du serveur pour le traitement des requêtes des clients. Il est défini comme un pseudo objet par la norme.

4.4. Identification des objets IOR

Une IOR *(Interoperable Object Reference)*, est une représentation binaire de la référence d'un objet. Elle identifie cet objet de manière unique. L'intérêt des IOR vient du fait qu'elles peuvent être interprétées par les différents ORB, et ce moyen d'identification universel constitue l'un des piliers de l'interopérabilité de CORBA. L'IOR est placée dans le stub du Client sans aucune interaction avec la partie applicative. Cependant elle est utilisée ensuite pour accéder directement à l'objet serveur.

4.5. Une famille de protocoles communs xIOP

A la base de CORBA se trouve un protocole de communication commun : IIOP. Pour être précis, signalons que ce dernier est la spécialisation d'un protocole plus général, baptisé GIOP (General Inter-ORB Protocol). Chargé de faire communiquer des ORB, ce dernier est conçu pour fonctionner avec tous les protocoles réseau existants par le biais de spécialisations. Par exemple, IIOP est la spécialisation de GIOP adapté au protocole TCP/IP ; c'est donc lui qui prend en charge toutes les communications entre les ORB dans le monde Internet.

4.6. Interfaçage dynamique (DII/DSI)

La norme CORBA spécifie un moyen d'accéder à des objets lors de l'exécution d'une application sans pour autant, avant cette exécution, avoir déterminé quels seront ces objets. En utilisant DII (Dynamic Invocation Interface), on construit dynamiquement des requêtes vers des objets CORBA pour lesquels on ne possède pas de stub. Avec DSI (Dynamic Skeleton Interface), on intercepte les requêtes en provenance de clients sans avoir pour cela généré de Skelton. DSI est la réciproque de DII, côté serveur. Ces deux

techniques servent lors de la réalisation d'applications qui doivent s'adapter automatiquement à l'évolution dynamique des interfaces des objets.

Annexe C : Les systèmes Multi Agents

1. Introduction

Les systèmes multi-agents (SMA) sont une approche privilégiée pour s'intéresser aux systèmes complexes grâce à leur démarche totalement décentralisée. Les modèles multi-agents modélisent des phénomènes dont le comportement global émerge en partie des interactions locales des entités les plus fines du système étudié. Ces entités à caractère autonome peuvent percevoir, agir et interagir dans leur environnement (Fournier, 2005).
La complexité des réseaux de capteurs et leurs exigences en term de performance et d'efficacité en consommation d'énergie ont conduit à exploiter les évolutions faites dans le domaine des systèmes multi-agents.

2. Les Systèmes multi-agents

2.1. Définitions

Les concepts des systèmes multi-agents (SMA) sont essentiellement issus de deux courants en informatique: l'intelligence artificielle et le génie informatique.
Comme tous les concepts de l'IA, la grande diversité des applications possibles réalisées à l'aide de la modélisation agent à donné plusieurs définitions qui dépendent le plus souvent du domaine applicatif.
Pour (Cardon, 2002), Un ***Système Multi-Agent (SMA)*** est constitué d'un ensemble d'agents ou d'organisations d'agents situés dans un environnement composé d'objets qui ne sont pas les agents du système. Ce système communique avec son environnement par l'action d'agents spécifiques dits agents d'interface. Un système multi-agents peut être vu selon quatre axes AEIO comme le montre la formule suivante (Jamont, 2005):

SMA = Agent + Environnement + Interaction + Organisation

De plus la fonction d'un système multi-agents, n'est pas seulement la somme des fonctions des agents, elle doit prendre en compte la notion de coopération entre les agents.

Fonction (SMA) = Σ fonction (Agent) + Fonction collective

Plusieurs définitions ont été données pour les agents, la définition adoptée par la plupart des documentations est celle donnée par Russell and Norvig (Russell, et al., 1995). D'après eux, *"**un agent** est toute chose qui perçoit des données de son environnement à l'aide des capteurs et agit sur cet environnement à l'aide des membres"*.

Nous pouvons tirer les propriétés suivantes d'un agent:

• *Autonomie:* les agents agissent sans intervention directe des humains, et possèdent leurs propres ressources et un moyen de contrôle de leurs actions et de leur état interne.

• *Réactivité:* les agents perçoivent, de manière limitée, leur environnement, et répondent de manière opportune à ses changements.

• *Capacité social:* les agents interagissent avec les autres agents (et éventuellement avec les humains) via un langage de communication agent.

• *Proactivité:* les agents n'agissent pas seulement en réponse à leur environnement, ils sont capables d'avoir des comportements dirigés par des buts en prenant l'initiative.

Dans un système multi-agents, on appelle ***environnement*** l'espace commun aux agents du système. D'après (Russell, et al., 1995), l'environnement à les propriétés suivantes: *Accessibilité, Déterministe ou non, Statique vs dynamique, Continu vs discret*.

Les ***interactions*** proviennent de la mise en relation dynamique de plusieurs agents par le biais d'un ensemble d'actions réciproques.

D'après Jacques Ferber (Ferber, 1995), les ***organisations*** constituent à la fois le support et la manière dont se passent les interrelations entre les agents, c'est-à-dire dont sont réparties les tâches, les informations, les ressources et la coordination d'actions.

2.2. Taxonomie des agents

Il existe plusieurs critères de classification des agents. Dans ce qui suit, nous allons citer les types d'agents les plus utilisé dans la littérature.

2.2.1. Agents communicants ou situés

• Un agent purement communicant est un système ouvert qui détient ses propres ressources. Il est conduit par l'ensemble de ses propres objectifs, et il peut communiquer avec d'autres agents. Son comportement est lié aux communications qu'il reçoit (Quinqueton, 2003).

• Un agent purement situé ne communique pas directement avec d'autres agents. Les interactions se font via l'environnement. Sa position dans l'environnement est celle qui détermine son comportement. Il peut se déplacer et se reproduire (Quinqueton, 2003).

2.2.2. Agents à capacités cognitives ou réactives

• Les agents à capacités cognitives s'inspirent du modèle humain. Ces agents possèdent une représentation explicite de leur environnement, des autres agents, et d'eux-mêmes. Ils sont aussi dotés de capacités de raisonnement et de planification ainsi que de communication.

• Les agents à capacités réactives ne possèdent pas de moyen de mémorisation et n'ont pas de représentation explicite de leur environnement: ils fonctionnent selon un modèle stimuli/réponse. En effet, dès qu'ils perçoivent une modification de leur environnement, ils répondent par une action programmée.

2.2.3. Classification basée sur la structure de l'agent

D'après (Russell, et al., 1995) chaque agent est décomposé en architecture et programme. Selon la structure de l'agent on peut trouver distinguer les types suivant:

• ***Agent à réflexe simple (Simple reflex agent):*** caractérisé par les règles conditionaction: ***if (condition) then action.***

• ***Agent à réflexe basé sur un modèle d'état (Model-based reflex agent)***: l'agent combine entre les perceptions courantes et l'historique des états internes (dit modèle interne) afin de construire une description de l'état courant.

• ***Agent basé sur le but (Goal-based agent)***: L'agent utilise les connaissances sur l'état courant et des informations décrivant la situation désirable afin d'agir sur l'environnement.

• ***Agent basé sur l'utilité (Utility-based agent)***: Durant le chemin vers le but destination on peut assigner à chaque état d'une configuration particulière de l'environnement une valeur d'utilité afin d'aboutir à un comportement de bonne qualité.

• ***Agent d'apprentissage (Learning agent)***: Afin d'améliorer la décision de l'agent, on lui donne le pouvoir d'apprendre au fur et à mesure ce dont il a besoin.

2.2.4. Agents à base de connaissances (agents logiques)

Ce sont des agents basés sur les connaissances disponibles concernant l'environnement et un raisonnement (logique) portant sur les actions possibles dans cet environnement.

2.2.5. Agents mobiles (MA: Mobile Agent)

Les agents mobiles sont des agents capables de se déplacer (programme et état) à travers un réseau, et recommencer leurs exécutions au site distant. La motivation principale derrière les agents mobiles est de remplacer les RPC (remote procedure calls). Quelques problèmes techniques doivent être pris en considération (Wooldridge, 2002):

- ***Sérialisation:*** comment coder l'agent dans un forme convenable pour être transmis dans le réseau, et quel aspect de l'agent va être sérialisé (programme et données).
- ***Exécution locale et distante:*** comment est exécuté un agent arrivé à sa destination.
- ***Sécurité:*** quand un agent est envoyé d'un hôte à un autre, il peut perturber l'exécution des processus de l'hôte distant.

2.3. Méthodes de Raisonnement

Le raisonnement est la manière de pensé d'un agent pour résoudre le problème en démarrant des hypothèses pour aboutir aux résultats. Plusieurs méthodes existent :

2.3.1. Raisonnement déductif

Les connaissances de l'agent sur l'environnement sont traduites en une description symbolique adéquate et précise appelé base de connaissances. Puis un processus de déduction est y appliqué afin d'aboutir au résultat.

2.3.2. Raisonnement Means-Ends

Cette méthode est connue sous le nom "Planning raisonnement" dans laquelle l'agent suit un plan (séquence d'actions) généré par un système de planification (Wooldridge, 2002). *Plan = P ("But ou intention", "état courant de l'environnement", "actions possibles»).*

2.3.3. Système de raisonnement procédural (PRS)

Le système de raisonnement procédural (SRP) est peut être la première architecture d'agent utilisant le paradigme BDI (belief-desire-intention). Dans un SRP, l'agent est équipé d'une bibliothèque précompilée de plans construits manuellement par le programmeur.

2.4. Communications entre agents

La communication est toujours considérée comme un sujet très important pour un système multi-agent. Elle permet aux agents d'échanger les différentes informations et de coopérer afin d'atteindre un but commun. Comme exemples d'ACLs (Agent Communication Language) nous pouvons citer (Nodine, et al., 1998): KIF (Knowledge Interchange Format), KQML (Knowledge Query and Manipulation Language) et le langage de communication d'agent de FIPA.

2.5. Coordination

L'un des problèmes les plus importants dans le travail coopératif est la coordination. Elle concerne la gestion des interdépendances entre les activités des agents. Plusieurs approches sont développées pour des activités de coordinations dynamiques (Wooldridge, 2002).

2.5.1. Coordination à travers une planification globale et partielle

Le principe est d'échanger les informations entre les agents coopérants afin d'atteindre des conclusions communes sur le processus de résolution du problème. La planification est partielle parce que le système ne génère pas un plan pour le problème entièrement. Elle est globale parce que les agents échangent des plans locaux et coopèrent afin d'atteindre une vue globale.

2.5.2. Coordination à travers les buts communs

Quand un groupe d'agent sont engagés dans une activité de coopération, ils ont un but commun en plus de leurs buts individuels. Levesque et al (Levesque, et al., 1990) ont défini la notion de but persistant commun (JPG: Joint Persistent Goal).

2.5.3. Coordination par modélisation mutuelle

Chaque agent se met à la place de l'autre et essaye de savoir ses croyances et ses intensions afin de prédire les actions à faire. L'un des systèmes d'agents qui implémente cette idée est le système *MACE*. Dans MACE chaque agent maintient un modèle de relations contenant une représentation des autres agents: leurs capacités, intérêts, leurs compétences...

2.5.4. Coordination basée sur les normes et les lois sociales

Les agents utilisent des normes, comparables à nos normes sociales de coordination, qui sont soit prédéfinis (inchangeable) ou bien change dans la vie du système (dans ce cas les nouvelles normes seront émergés vers tous les agents).

2.6. Apprentissage

L'apprentissage induit des changements de contenu et d'organisation de la base de connaissances d'un système susceptibles d'améliorer ses performances pour une tâche particulière ou pour un ensemble de tâches. Il y a apprentissage lorsque le système acquiert de nouvelles connaissances ou lorsqu'il réorganise ses connaissances actuelles pour en faire un meilleur usage. Il existe trois grandes familles d'apprentissage (Russell, et al., 1995):

2.6.1. Apprentissage supervisé

Processus dans lequel l'apprenant reçoit des exemples d'apprentissage comprenant à la fois les valeurs d'entrée et de sortie.

2.6.2. Apprentissage non-supervisé

Contrairement à l'apprentissage supervisé, on ne fournit ici qu'une base d'entrées, et c'est le système qui doit déterminer ses sorties en fonction des similarités détectées entres les différentes entrées (règle d'auto organisation).

2.6.3. Apprentissage par renforcement

Dans cette méthode, l'agent apprend sans supervision et tout le temps. L'apprentissage est basé sur le principe de rétro-action: lorsque l'agent est confronté à une certaine situation, il choisit une action. Sa réaction est alors jugée par rapport à un objectif

prédéfini, et l'agent reçoit une récompense déterminée par la qualité de sa réponse par rapport à la réponse optimale.

3. Plateformes pour les systèmes multi-agents

Une plate-forme de développement des systèmes multi-agents est une infrastructure de logiciels utilisée comme environnement pour le déploiement et l'exécution d'un ensemble d'agents. Elle devrait fournir des manières confortables pour créer et tester des agents et agir en tant qu'un médiateur entre le système d'exploitation et les applications (agents) tournant dessus.

Plusieurs plates-formes multi-agents existent: les plates-formes de simulation, les plates-formes de développement et les plates-formes d'exécution. Elles ont été un courant notable d'influence pour les méthodes orientées agent.

3.1. AgentBuilder

AgentBuilder (Age) est une suite intégrée d'outils permettant de construire des agents intelligents, il a été développé en JAVA par Reticular Systems Inc. L'élaboration du comportement des agents se fait à partir du modèle BDI et du langage AGENT-0. KQML est utilisé comme langage de communication entre les agents. Par contre on peut créer des fichiers «.class » et les exécuter sur une JVM standard.

AgentBuilder est composé d'une interface graphique et d'un langage orienté agent permettant de définir des croyances, des engagements et des actions. Il permet également de définir des ontologies et des protocoles de communications inter-agents. Un agent crée avec cet outil est typiquement un agent d'interface, chargé de faciliter la recherche d'information ou la réalisation de certaines tâches à la place de son utilisateur. Un tel agent sera capable de filtrer l'information, de négocier des services avec d'autres agents et dialoguer avec son utilisateur.

3.2. Jack

Jack (http://www.agent-software.com.au) est décrit comme étant un environnement pour construire, exécuter et intégrer des systèmes multi-agents commerciaux, écrits en Java et utilisant une approche orientée composants. Il est développé par Agent Oriented

Software Pty. Ltd., une société australienne. Il est basé sur le modèle BDI dMARS développé à l'Australian Artificial Intelligence Institute. Jack s'intéresse principalement sur l'étape de développement. Les outils logiciels sont le JDE (Jack Development Environment), un environnement de programmation graphique, le compilateur du *Jack Agent Language* (JAL), qui traduit les programmes écrits en JAL en Java pur, et la librairie de classes permettant l'exécution des agents, appelée *Jack Agent Kernel*.

3.3. MadKit

MadKit (Multi-Agents Developpement Kit) (Gutknecht, et al., 2000) est une plateforme multi-agents modulaire et scalable écrite en Java et conçue selon le modèle d'organisation Alaadin AGR (Agent/Group/Role): des agents sont situés dans les groupes et jouent des rôles. MadKit est développée en 1996 par Olivier GUTKNECHT et Michel FERBER au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) de l'Université Montpellier II. Il s'agit d'une plate-forme libre pour l'utilisation dans l'éducation. Un moteur d'exécution est utilisé dans MadKit où chaque agent est construit en partant d'un micro-noyau. Chaque agent a un rôle et peut appartenir à un groupe. MadKit est doté d'un environnement de développement graphique qui permet facilement la construction des applications.

3.4. MASK

MASK (Multi-Agent System Kernel) (Michel, et al., 2004) est un environnement de développement de systèmes multi-agents expérimental développé depuis 1993 dans l'équipe MAGMA. Il est fondée sur l'approche d'analyse et de conception Voyelles a constitué le premier support logiciel de cette méthode. L'objectif est de fournir des bibliothèques d'Agent, de manipulation d'Environnements, d'Interactions, et d'Organisations ainsi que des outils d'aide à la programmation. Les modèles implémentés sont des architectures d'agents hybrides (ASTRO) et réactifs (PACORG), des modèles d'interaction à protocoles (IL) et à forces (PACO), et un modèle d'organisation statique (RESO). Ces modèles intègrent des éditeurs permettant d'éditer les instances de ces modèles.

3.5. Zeus

Zeus est un environnement intégré pour la construction rapide d'applications à base d'agents collaboratifs. Il est développé par l'Agent Research Program du British Telecom Intelligent System Research Laboratory. La documentation de Zeus est abondante, et insiste particulièrement sur l'importance des aspects méthodologiques de Zeus « The agent creation methodology is vital to the use of the Zeus toolkit ». La méthodologie de Zeus utilise décomposition en quatre étapes pour la construction d'agents nommées respectivement: analyse du domaine, conception, réalisation et support d'exécution.

IV.3.6 Dima

DIMA (Développement et Implémentation de Systèmes Multi-Agents) (Z.Guessoum, et al., 2001) est un environnement de développement de systèmes multi-agents dont le développement a débuté en 1993. Il se caractérise par une approche résolument orientée Objets, l'objectif étant d'étendre les Objets pour en faire des Agents, en leur donnant les propriétés manquantes, en particulier l'autonomie, la proactivité, l'adaptation et la sociabilité. L'architecture proposée est modulaire. Un Agent est composé de différents composants, chacun implémentant un comportement de l'agent. Ces comportements peuvent être réactifs (passifs) ou cognitifs (proactifs). L'orientation objet permet la composition et l'héritage des comportements. Les agents sont adaptatifs, grâce à un mécanisme de méta-comportement qui leur permet de choisir le comportement le plus adapté.

3.7. JADE (voir l'Annexe D)

Annexe D: Java Agent DEvloppement framework (JADE)

1. Introduction

Notre application est développée avec des agents JADE, ceci nous conduit à donner une présentation sommaire de cette plateforme. Nous présentons dans ce chapitre une vue globale de l'environnement des agents JADE.

JADE est une marque déposée enregistrée par CSELT et résulte principalement d'une activité de recherche. Il a avec succès passé les tests d'interopérabilité de FIPA à Séoul en janvier 1999 et à Londres avril 2001 et a été intensivement employé dans le cadre de plusieurs projets. Le développement de Jade continue toujours. D'autres améliorations, perfectionnements, et réalisations ont été déjà projetés, la plupart d'entre eux en collaboration avec les utilisateurs intéressés de la communauté de Jade.

JADE (Java Agent Development Framework - Bellifemine, Poggi, Rimassa, 1999) est une plate-forme multi-agents développée en Java par CSELT (Groupe de recherche de Gruppo Telecom, Italie) qui a comme but la construction des systèmes multi-agents et la réalisation d'applications conformes aux spécifications FIPA (**Foundation for Intelligent Physical Agents)** (FIPA, 1997). JADE comprend deux composantes de base : une plate-forme agents compatible FIPA et un paquet logiciel pour le développement des agents Java.

Le JADE est entièrement développée dans Java et est basé sur les principes fondamentaux suivants :

Interopérabilité. Le JADE est conforme avec les spécifications de FIPA .Par conséquence un agent JADE peut interagir avec d'autres agents d'autres plateformes (à condition qu'ils se conforment à la même norme).

Uniformité et portabilité. Le JADE fournit aux applications un ensemble homogène d'APIs qui sont indépendantes du réseau et de la version de Java. Plus en détails, l'environnement d'exécution de JADE fournit les même APIs que celles de l'environnement de J2EE, de J2SE et de J2ME. Dans la théorie, les développeurs pourraient décider l'environnement d'exécution de Java au temps de déploiement.

Facilité d'utilisation. La complexité du middleware est cachée derrière un ensemble d'APIs simples et intuitives.

Respect la philosophie pay-as-you-go. Les programmeurs n'ont pas besoin d'employer tous les outils fournis par le middleware. ça n'exige pas au développeur de maitrisé ni de connaitre les outils qui ne luis concernes pas.

2. Architecture de jade

La figure Fig.A 6 (Fabiob-Bellifemine, et al., 2007) montre les éléments architecturaux principaux d'une plateforme JADE. Une plateforme JADE se compose de conteneurs d'agent qui peuvent être répartis sur le réseau. Les agents vivent dans des conteneurs qui sont les processus Java qui fournit l'environnement d'exécution et tous les services requis pour accueillir et exécuter des agents. Il y a un conteneur spécial, appelé *'main container'*, qui représente le point centrale d'une plateforme : c'est le premier conteneur à lancer et tous autres conteneurs doivent se joindre à lui par l'inscription.

Le programmeur identifie les conteneurs en employant simplement un nom logique ; par défaut le conteneur principal est appelé 'Main container' tandis que les autres sont appelés `Container-1 ', ` Container-2 ', etc.… ; des options de ligne de commande sont disponible pour dépasser les noms par défaut.

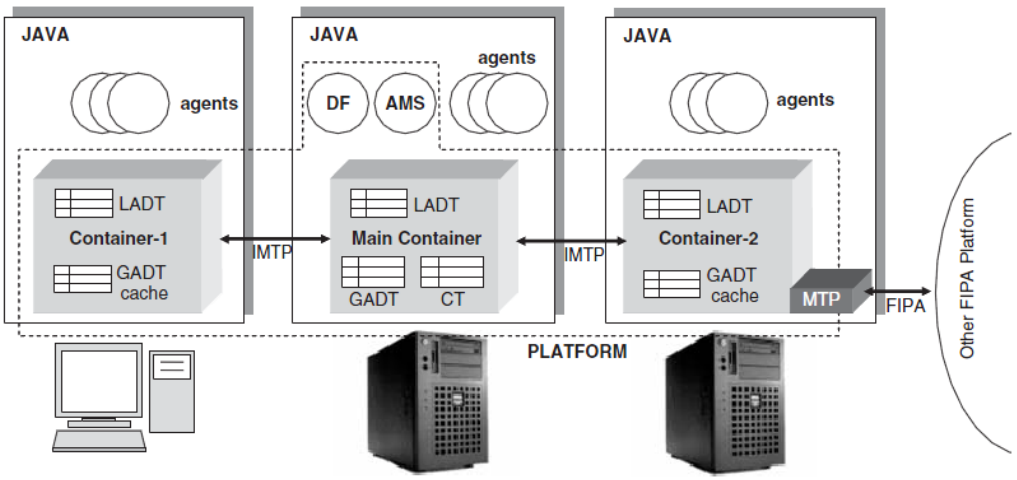


Fig.A 6 Relation entre les principaux éléments architecturaux

Le conteneur principal a les responsabilités spéciales suivantes :
Gestion de la table de conteneur (CT), c'est l'enregistrement des références d'objet et des adresses de transport de tous les nœuds de conteneurs composant la plateforme ;
Gestion de la table de descripteur globale d'agent (GADT), c'est l'enregistrement de tous les agents actuels dans la plateforme, y compris leur état actuel et leur endroit ;
Accueille des deux agents AMS (Agent Managment System) et DF (Directory Facilitator), les deux agents spéciaux qui fournissent respectivement, le service de gestion d'agent et de page blanche, et le service de page jaune de la plateforme.

La plate-forme d'agent Jade inclut tous les composants obligatoires qui contrôlent un SMA. Elle peut être répartie sur plusieurs serveurs. Une seule application Java, et donc une seule machine virtuelle de Java (JVM), est exécutée sur chaque serveur. Chaque JVM est un conteneur d'agents qui fournit un environnement complet pour l'exécution d'agent et permet à plusieurs agents de s'exécuter en parallèle sur le même serveur. Chaque conteneur d'agents est un environnement multi- threads composé d'un thread d'exécution pour chaque agent. L'architecture de communication offre la transmission de messages flexibles et efficaces. JADE crée et contrôle une file d'attente des messages entrants pour chaque agent. Le modèle global de communication FIPA a été mis en application. Ses composants ont été distingués clairement et ont été entièrement intégrés: protocoles d'interaction, ACL, langues, schémas de codage, protocoles de transport...
Une plate-forme multi-agents JADE est alors composée de plusieurs conteneurs d'agents. La distribution de ces conteneurs à travers un réseau d'ordinateurs est permise, à condition que la communication RMI entre leurs hôtes soit conservée. Chaque conteneur d'agents est un objet serveur RMI qui gère localement un ensemble d'agents. Il règle le cycle de vie des agents (création, suspension, attente et destruction). Il assure en plus, le traitement de tous les aspects de la communication : répartition des messages ACL reçus, routage des messages selon l'agent destinataire (receiver) et dépôt des messages dans les files de messages privées des agents. Pour les messages vers l'extérieur, le conteneur d'agents maintient assez d'information pour chercher l'emplacement de l'agent récepteur et pour choisir une méthode de transport convenable pour expédier le message.

La plate-forme d'agent fournit une interface graphique utilisateur (GUI) pour la gestion à distance des agents RMA (Remote Management Agent), surveillant le statut des agents, permettant par exemple d'arrêter et de remettre en marche des agents. Le GUI permet également de créer et de lancer un agent sur un serveur à distance, à condition qu'un conteneur d'agent fonctionne déjà. Le GUI permet de commander d'autres plates-formes d'agent à distance.

3. Comportement d'un agent (Behaviour)

Un comportement représente une tache qu'un agent doit effectuer. Chaque comportement possède deux méthodes ; la méthode action() qui définit les opérations à exécuter par l'agent, et la méthode done() qui renvoie un booléen spécifiant si le comportement a terminé et dans ce cas il sera supprimé de la file des comportement active(Fig.A 7).
On distingue plusieurs types de comportements :

Comportements à mono coup (**One-shot behaviors)**

ils sont conçu pour être accomplies dans une seul phase d'exécution, leur méthode *action()* est ainsi exécuté qu'une seul fois. La classe `jade.core.behaviours.OneShotBehaviour` implémente la méthode *done()* pour retourner *true* et peut être étendue pour implémenté de nouveaux comportements de type one-shot.

```
public class MyOneShotBehaviour extends OneShotBehaviour {
   public void action() {
     // perform operation X
   }
}
```

Dans cette exemple l'opération X est exécuté une seul fois.

Comportement Cyclique (Cyclic behaviors):

comportements sont conçus pour ne jamais accomplir ; leur méthode *action()* exécute les mêmes opérations à chaque fois qu'elle est appelée. La classe `jade.core.behaviours.CyclicBehaviour` implémente la méthode *done()*

pour retourner *false* et peut être étendue pour implémenté de nouveaux comportements de type Cyclique

```
public class MyCyclicBehaviour extends CyclicBehaviour {
    public void action() {
      // perform operation Y
    }
}
```

Dans cet exemple, l'opération Y est effectuée répétitivement jusqu'à ce que l'agent exécutant le comportement se termine.

Comportement Générique (Generic behaviours)

Elle exécute différentes opérations qui dépendent d'un état, elle termine quand une certaine condition est satisfaite.

```
public class ThreeStepBehaviour extends Behaviour {
  private int step = 0;
  public void action() {
    switch (step) {
    case 0:   // perform operation X
              step++;
              break;
    case 1:  // perform operation Y
              step++;
              break;
    case 2:  // perform operation Z
              step++;
              break;
    }
  }
  public boolean done() {
   return step == 3;
  }
}
```

Dans cette exemple la variable *step* représente l'état du comportement ; les opération X, Y et Z serons exécutées séquentiellement dans trois occasion de lancement du comportement pour enfin il s'accomplit

Comportements composés

Le JADE fournit également la possibilité de composer des comportements pour créer des comportements complexes.

Ce dispositif, est particulièrement commode en mettant en application des tâches plus complexes

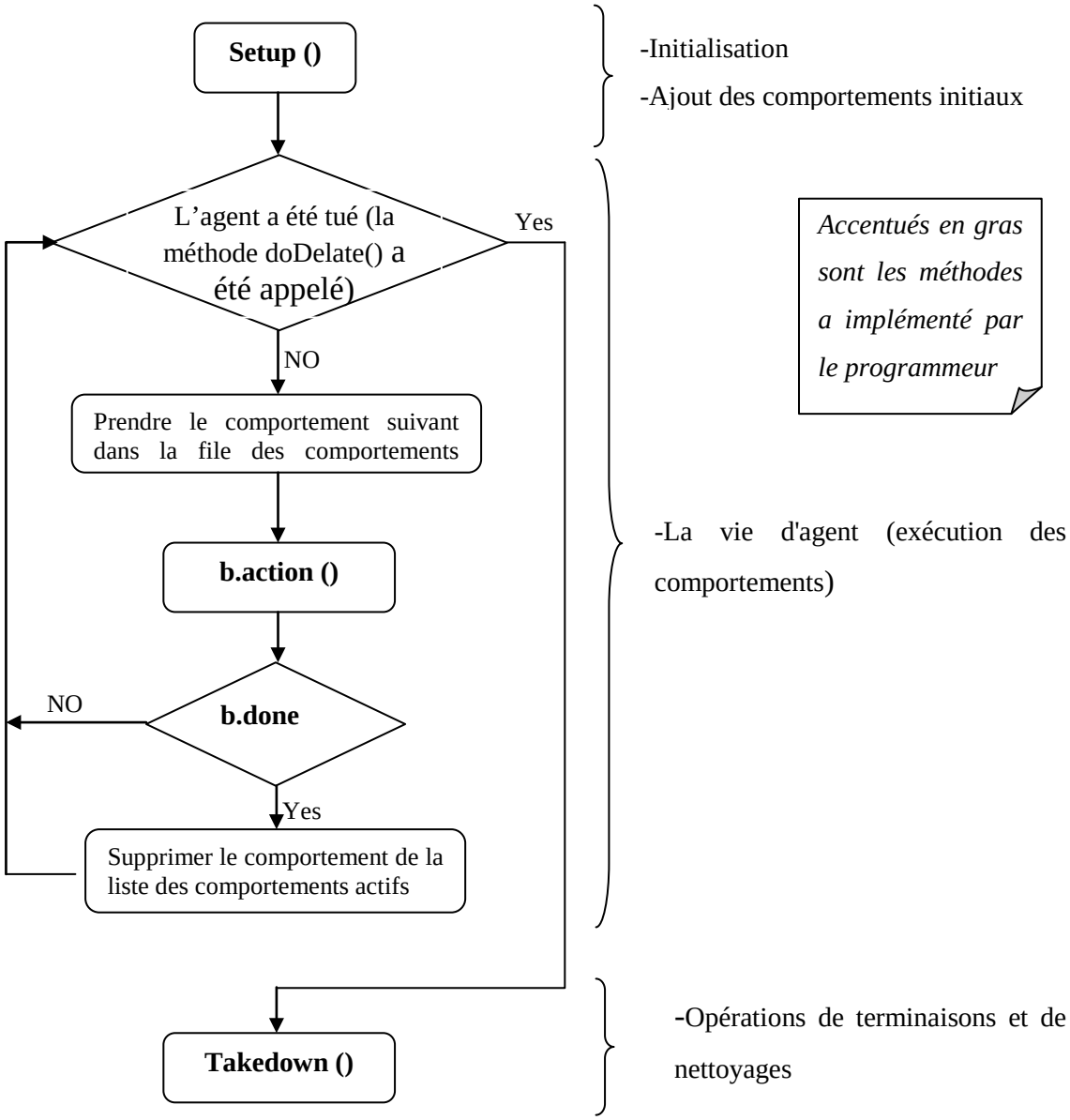


Fig.A 7 Exécution des comportement d'un agent JADE

4. Communication entre agents

Un des dispositifs les plus importants que les agents JADE fournissent est la capacité de communiquer. Le paradigme de communication adopté est **le mode asynchrone**. Chaque agent a une sorte de boîte aux lettres (la file d'attente de message d'agent). A chaque fois qu'un message arrive, il est ajouté à la queue de la file et l'agent commence par traiter le premier message arrivé(voirFig.A 8) (Fabiob-Bellifemine, et al., 2007).

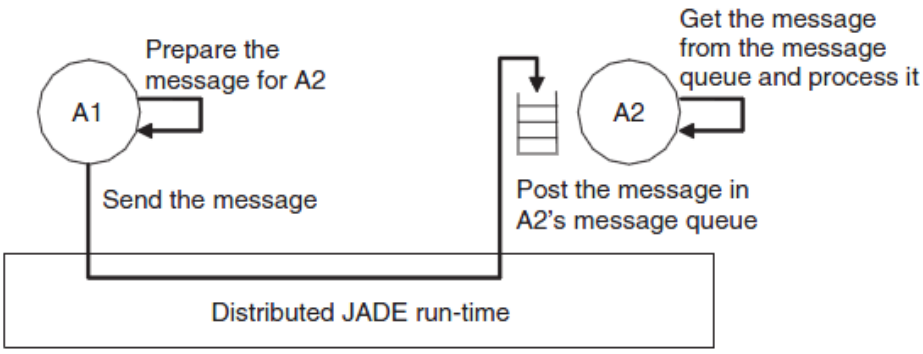


Fig.A 8 Principe de communication asynchrone des agents

Les messages échangés par les agents JADE obéissent au format indiqué par le langage ACL qui est défini par la norme FIPA pour l'interopérabilité des agents. Ce format comporte un certain nombre de champs, en particulier :

• L'***expéditeur*** du message (sender)

• La liste des destinataires (receiver)

• Le ***contenu du message,*** c'est l'information réelle incluse dans le message (le contenu du message qui peut être une chaîne de caractères ou un objet correspondant a l'ontologie utilisée).

• La ***langue,*** représente la syntaxe exprimée dans le contenu (l'expéditeur et le récepteur doivent pouvoir être en mesure de coder /analyser des expressions conformément à cette syntaxe pour que la communication soit efficace).

• L'***ontologie,*** c'est le vocabulaire des symboles utilisés dans le contenu du message (l'expéditeur et le récepteur doivent attribuer la même signification aux symboles pour que la communication soit efficace).

• L'intention communicative (appelée "***performative*** ") indiquant ce que l'expéditeur prévoit en envoyant le message.

La ***performative*** peut prendre les valeurs suivantes :

• REQUEST (DEMANDE), si l'expéditeur veut que le récepteur effectue une action.

• INFORME, si l'expéditeur veut que le récepteur se rende compte d'un fait.

• QUERY_IF, si l'expéditeur désire savoir si une certaine condition est satisfaite.

• CFP (appel pour la proposition), si l'expéditeur désire une offre de proposition du receveur.

• PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL, si l'expéditeur et le récepteur sont engagés dans une négociation.

Bibliographie

Bibliographie

A.Erradi, S.Anand et N.Kulkarni Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture [Revue] // IEEE International Conference on Services Computing (SCC'06). - 2006.

Agentbuilde [En ligne] // http://www.agentbuilder.com/.

Audrey Occello Introduction à l'architecture orientée service // cour SAR O3 – SI3 SOA. - 2006.

Balis.B, Bubak.M et Wegiel.M "A solution for adapting legacy code as web services" Workshop on Component Models and Systems for Grid applications. 18th Annual ACM International Conference on Supercomputing, Saint-Malo France. - 2004. - pp. 57-75.

Benmokhtar Sonia "Synthése ad hoc de services Web dans les environnements de l'informatique diffuse" Université Pierre et Marie Curie, 2003-2004.

Bo Zhang, Liang Bao, Rumin Zhou, Shengming Hu et Ping Chenet "A Black-Box Strategy to Migrate GUI-Based Legacy Systems to Web Services" IEEE International Symposium on Service-Oriented System Engineering. - 2008. - pp. PP 25-31.

Cardon Alain "Etude de la conception et du contrôle comportemental d'une organisation massive d'agents" LIH, Faculté des Sciences Université du Havre., 2002.

Cetin Semih, Altintas N. Ilker, Oguztuzun Halit, Dogru Ali H, Tufekci Ozgur et Suloglu Selma "Legacy Migration to Service-Oriented Computing with

Mashups" IEEE International Conference on Software Engineering Advances(ICSEA). - 2007.

Cigref "Livre blanc, Accroitre l'agilité du systéme d'information" Cigref :Club informatique des grandes entreprises françaises, 2003.

Dietmar Kuebler et Wolfgang Eibach "Adapting legacy applications as Web services" www.ibm.com. - 2002.

F.Haniche, H.Mellah et H.Drias "Integrating legacy systems in a SOA using an agent based approach for information system agility" IEEE, International Conference on Machine and Web Intelligence (ICMWI 2010). pp. 317-322.

Fabiob-Bellifemine, Giovanni-Caire et Dominic-Greenwood "Developing Multi Agent Systems with JADE" Wiley, 2007.

Ferber "Les Systèmes Multi-Agents: vers une intelligence collective" InterEdition, 1995.

Forrester Research Inc "Transformation des applications existantes : transition accélérée vers SOA" 2007.

Fournier Sébastien "Intégration de la dimension spatiale au sein d'un modèle multi-agents à base de rôles pour la simulation: Application à la navigation maritime" Thèse de doctorat. École doctorale: (MATISSE). L'Université de Rennes 1, 2005.

Frank Jennings, Matjaz B.Juric, Poornachandra Sarang, Ramesh Loganathan "SOA Approach to Integration" Packt Publishing, 2007.

G.alonso, f.casati, h.kuno, v.machiraju "Web Services Concepts, Architectures and Applications" Springer-Verlag Berlin -, 2003.

G.Lewis, D.Smith et E.Morris "SMART: The Service-Oriented Migration and Reuse Technique" Software Engineering Institute, 2005.

Gerardo Canfora [et al.] "Migrating Interactive Legacy Systems to Web Services" IEEE Conference on Software Maintenance and Reengineering (CSMR'06). - 2006.

Gilles Laborderie "SOA par la pratique" http://blog.octo.com/. - 2008.

Giusy Di-Lorenzo [et al.] "Turning Web Applications into Web Services by Wrapping Techniques" IEEE Reverse Engineering, Working Conference. 2007. pp. 199-208.

Grace Lewis, Edwin Morris et Dennis Smith "Analyzing the Reuse Potential of Migrating Legacy Components to SOA" IEEE Conference on Software Maintenance and Reengineering (CSMR'06). 2006.

Gutknecht O., Ferber J. et Michel F. "MADKIT:Une plate-forme multi-agents générique" Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier, 2000.

H.Mellah, S.Hassas et H.Drias "Modelling information in a three dimensional system for a contribution to the agility of an enterprise information system" International Research journal System and Information Sciences. 2008. pp. 84-89.

Hamadi Rachid et Benatallah Boualem "A Petri Net-based Model for Web Service Composition"Australian Computer Society, Inc. the Fourteenth Australasian Database Conference (ADC2003). - 2003.

Harr M.Sneed et AneCon GmbH Wien "Wrapping Legacy Software for Reuse in a SOA" CiteSeerX, 2008.

He Guo et Chunyan Guo "Wrapping Client-Server Application to Web Services for Internet Computing" the Sixth IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05). 2005. pp. 1-5.

Jamont Jean-Paul "DIAMOND: une approche pour la conception de systèmes multi-agents embarqués" Thèse de doctorat, Ecole Doctorale Mathématiques Sciences et Technologies de l'Information. Institut National Polytechnique de Grenoble, 2005.

Joris Van et Geet "Reverse Engineering in the World of Enterprise SOA" 15th IEEE Working Conference on Reverse Engineering. 2008. pp 311-314.

K.Bennett "Legacy systems: Coping with success" IEEE Computer Society Press Los Alamitos, CA, USA, 1995.

Kanchanavally, Kunal Mittal et Srinivas "Pro Apache Beehive Book chapter2: Introducing web services and SOA fundamentals" Appress, 2005. - pp. 15-25.

L.O'Brien, C.Verhoef et C.Stoermer "Software Architecture Reconstruction: Practice Needs and Current Approaches" Software Engineering Institute, 2002.

Levesque H. J., Cohen P. R. et Nunes. J. H. T. "On acting together" the 8th National Conference on Artificial Intelligence (AAAI- 90). - Boston 1990. - pp. 94-99.

M.Eveno et C.Heubès "Comprendre et savoir utiliser un ESB dans une SOA" livre blanc : Xebia, 2007.

McCoy DW et Plummer DC "Defining cultivating and measuring enterprise agility" Gartner research, 2006.

Michel Occello [et al.] "MASK: An AEIO Toolbox to Design and Build Multi-Agent Systems" Amsterdam, Netherlands. In Knowledge Engineering and Agent Technology, 2004.

Nodine Marian et Chandrasekara Damith "Agent Communication Languages for Information-Centric Agent Communities [Livre]. - [s.l.] : MCC Technical Report MCCINSL- 096-98, 1998.

Quinqueton Joël "Cours: Fondements des Systèmes Multi-Agents" LIRMM, 2003.

Rosa-Roserio Ricardo "Découverte et Sélection de Services Web pour une application Mélusine" Institut d'Informatique et de Mathématiques Appliquées de Grenoble, 2004.

Rousset Laurent, Acremann David et Moujeard Gilles "Développer avec CORBA en Java et C++ " Edition Compus Press, 1999.

Russell Stuart.J et Norvig Peter "Artificial Intelligence:A Modern Approach" New Jersey : Prentice-Hall, 1995.

Sucharov Leon "Legacy modernisation" ITadviser, Issue 47, 2007.

Syntec informatique "Moteur de l'innovation, Les Architectures orientées service" Collection thématique N°10, 2007.

Toumani Farouk et Kellert Patrick "Les Web services sémantiques" Laboratoire LIMOS - UMR (6158) du CNRS.

Toumani Patrick Kellert et Farouk "Les Web services sémantiques " Laboratoire LIMOS - UMR (6158) du CNRS.

Willem-Jan van den Heuvel "Aligning Modern Business Processes and Legacy Systems" Massachusetts Institute of Technology, 2007.

Wooldridge Michael "An Introduction to Multiagent Systems " JOHN WILEY & SONS, LTD. England, 2002.

Y.Huang [et al.] "Wrapping Legacy Codes for Grid-Based Applications" the 17th International Parallel and Distributed Processing Symposium (Workshop on Java for HPC), 2003. pp. 1-7.

Z.Guessoum, T.Meurisse et J-P.Briot "Construction modulaire d'agents et de systèmes multi-agents adaptatifs en DIMA " Techniques et Sciences Informatiques, 2001. - Vol. 1.

Zhuopeng.Zhang et Hongji.Yang "Incubating services in legacy systems for architectural migration" IEEE, the 11th Asia-Pacific Software Engineering Conference (APSEC'04). 2004. pp. 196 – 203.