UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département d'Informatique

MEMOIRE DE MAGISTER

Option : Systèmes d'information et de connaissances

UNE APPROCHE ARCHITECTURE LOGICIELLE POUR LA COMPOSITION DE SERVICES WEB

Par

Khadidja BENTLEMSAN

Devant le jury composé de :

H. ABED	Professeur, USDB	Présidente
N. BENBLIDIA	Maitre de conférences (A), USDB	Examinatrice
K.W. HIDOUCI	Maitre de conférences (A), ESI.	Examinateur
A. HENNI	Maitre de conférences (A), ESI	Promoteur
D. BENNOUAR	Maitre de conférences (B), USDB.	Co-Promoteur

Blida, juin 2010

RESUME

La composition de Services Web (SW) est un problème sur lequel se concentre un nombre important de travaux de recherches. Les approches actuelles de composition de SW se basent essentiellement sur la mise en œuvre des langages orientés spécifiquement vers la résolution du problème de composition de SW.

Les Langages de Composition de Services Web (LCSW) nécessitent un travail pénible et de longue haleine pour la maitrise des divers concepts, mécanismes et artefacts liés d'une part aux SW eux même et d'autre part aux approches de composition de SW. De plus, les LCSW sont limités et ne permettent pas la spécification d'une composition hétérogène, dans laquelle nous trouvons des services Web et des services non Web.

Le travail présenté dans ce mémoire propose une nouvelle approche qui vise d'une part à simplifier la composition des SW et d'autre part à permettre la composition hétérogène. Cette approche consiste à considérer le problème de composition de SW dans l'espace Architecture Logicielle et utiliser les langages de description d'architecture logicielle comme support de spécification de la composition de SW et des compositions des services hétérogènes.

Mots clés : Service Web, Architecture Logicielle, Composant, Connecteur

الملخص

تركيب خدمات الويب هو مشكل تتركز عليه عدة أعمال بحث .وتستند الطرق الحالية على إعداد لغات موجهة تحديدا نحو حل مشكلة تركيب خدمات الويب.

لغات تركيب خدمات الويب تتطلب جهدا لا بأس به من المصمم للتأقلم مع المفاهيم المختلفة و الآليات المرتبطة من جهة بخدمات الويب، ومن جهة أخرى بطرق تركيب هذه الخدمات، بالإضافة إلى أن مهمة هذه اللغات محدودة، ولا تمكننا من التركيب الغير متجانس ، أي خدمات ويب مع خدمات غير الويب.

العمل المعروض في هذه المذكرة يتمثل في اقتراح منهاج جديد يهدف إلى تبسيط التركيب وتوفير التركيب غير المتجانس على مستوى الهندسة، واستعمال لغات وصف الهندسة البرمجية لتطبيق التركيب الغير متجانس للخدمات.

الكلمات الرئيسية : خدمات الويب، الهندسة البرمجية ، مكون، موصل

ABSTRACT

Web Service (WS) composition is a problem on which focus a great number of research works. Current approaches are based primarily on the implementation of language directed specifically to solve the composition problem of WS.

The Web Service Composition Languages (WSCL) require a laborious and time-consuming for the mastery of various concepts, mechanisms and artifacts related to the WS and other approaches of WS composition. In addition, WSCL are limited and do not allow the specification of a heterogeneous composition, in which we find Web services and non-Web services.

The work presented in this thesis proposes a new approach which aims at one hand to simplify the composition of WS and secondly to enable the heterogeneous composition. This approach considers the composition of WS problem in software architecture space and use the Software Architecture Description Languages as a support for specifying the WS composition and the composition of heterogeneous services.

Keywords: Web service, Software architecture, Component, Connector

REMERCIEMENTS

Mes meilleurs remerciements vont tout d'abord à Mr Djamel BENNOUAR, sa patience, sa disponibilité permanente et ses conseils judicieux m'ont permis de réaliser ce modeste travail.

Je remercie vivement les membres de jury de m'avoir fait l'honneur d'être rapporteurs et examinateurs tout en apportant leurs remarques et leurs contributions à l'élaboration de ce mémoire.

Je souhaite exprimer mes reconnaissances les plus sincères aux enseignants qui ont contribué à ma formation de magister.

Nombreux sont ceux qui m'ont encouragé et soutenu, je ne saurais leur exprimer mes remerciements autant que je le souhaiterais ; ma profonde reconnaissance à tous ceux qui ont contribué de prés ou de loin à l'élaboration de ce mémoire.

DEDICACE

A La Mémoire De Mon Grand Père

A Tous Ceux Qui Me Sont Chers

Je dédie ce travail

TABLE DES MATIERES

LISTE DES ILLUSTRATIONS GRAPHIQUES ET TABLEAUX

INTRODUCTION

Depuis son apparition, le Web ne cesse de connaître des succès extraordinaires et une utilisation de plus en plus croissante. Il constitue aujourd'hui le support le plus préféré pour les interactions simples de type B2C[1] (Business to Customer).

B2C s'est rapidement étendu à ce qu'on appelle les échanges B2B [2](Business to Business) permettant la communication entre des applications de diverses organisations. Ces dernières ont ainsi commencé à mettre sur Internet un certain nombre de services pouvant être exploitées individuellement ou dans le contexte de la définition de systèmes plus complexes. B2B a rendu le Web d'aujourd'hui comme une machine parallèle énorme faisant tourner des applications dont les composants sont distribués un peu partout sur les sites formant la web-machine.

Parmi les diverses approches introduites pour la mise en place de systèmes distribués sur le Web, tel que les bus CORBA, et les RMI de Java, les services Web représentent la solution la plus prometteuse. Le choix de Microsoft du protocole SOAP, qui est le protocole de communication des services Web, comme protocole officiel de son système « .Net » n'est qu'un des plus grands témoignages de l'importance des services Web dans la conception de systèmes B2B.

Les services Web ont été proposés initialement par IBM et Microsoft, puis en partie standardisés par le consortium W3C. Un service Web est un composant logiciel qui opère dans le contexte des éléments fondamentaux suivant :

1- Un protocole standard de communication : Les services Web utilisent le protocole SOAP (Simple Object Access Protocol) pour réaliser les divers

[1] Logiciels informatiques permettant de mettre en relation des entreprises avec leurs clients (consommateurs)

[2] Logiciels informatiques permettant de mettre en relation des entreprises, dans un cadre de relations clients/fournisseurs

échanges entre eux. Il est principalement utilisé pour faire des appels de méthodes distantes. Le protocole SOAP utilise le langage XML pour la spécification de toute information échangée. Le protocole utilise une enveloppe pouvant être signée et contenant les diverses actions et données échangées. Le protocole SOAP est souvent transporté par le protocole http. il peut aussi être transporté par d'autres protocoles tels que FTP et SMTP) SOAP transporté par HTTP représente une puissante alternative aux autres techniques d'appel de procédure distantes tels que les RPC et RMI, vu que ces derniers sont souvent filtrés par les divers pare feu installé un peu partout sur Internet.

2- Une interface à travers laquelle le service Web expose les fonctionnalités qu'il est capable de réaliser. L'interface d'un service Web est décrite dans langage de description de service web ou WSDL (Web Service Description Language). L'interface d'un service Web est spécifiée en XML et est identifiée par une URL.
3- Un mécanisme de découverte : UDDI (Universal Description, Discovery and Integration) est un mécanisme de découverte des services Web, standardisé par l'OASIS depuis 2005. UDDI permet aux entreprises de s'inscrire et de publier leurs services Web. il se comporte lui-même comme un service Web dont les méthodes sont appelées via le protocole SOAP. Les opérations pouvant être effectuées concernent la recherche, la navigation, l'ajout, et la suppression de services .UDDI permet aux développeurs de découvrir les détails techniques d'un service Web (le WSDL) ainsi que les informations orientées métier.

La mise en place d'un service Web et son exploitation peuvent être résumées par les étapes suivantes :

- Le fournisseur d'un service Web déploie ce dernier sur Internet et publie sa description (WSDL) dans un registre UDDI.
- Un client définit des critères et lance à l'aide du registre UDDI la recherche d'un service correspondant critères spécifiées.
- Une fois le service trouvé, le client récupère sa description WSDL
- le client invoque ensuite le service

Les efforts de recherche/développement considérables et sans cesse croissant des grandes entreprises dans le domaine des services Web laisse entendre que dans un avenir très proche il sera question de réaliser des applications basées principalement sur les services Web. Le web sera ainsi le lieu ou seront exposé un très grand nombre de service Web, prêt à être exploité.

Dans la conception d'application basée principalement sur les services Web, nous pouvons distinguer deux sortes d'application : La première est représentée par une application ordinaire qui fait appel à des services Web. La deuxième est représentée par une application qui elle-même devient un service web. Nous disons qu'une telle application a été réalisée par la composition de service Web. La composition de service Web vise la mise en place de service Web qui accomplissent des tâches pour résoudre des problèmes qu'un simple service web ne saurait résoudre.

La composition des services Web a pour objectif de déterminer une combinaison de services en fonction des besoins d'un client. Du coté du client, cette composition semblera un unique service. La composition sera transparente au client, même si cette composition sera la combinaison de plusieurs services web. Par ailleurs, une composition peut être composée de services atomiques ou composites. Les services atomiques sont caractérisés pour avoir une unique fonctionnalité. Contrairement aux services composites qui peuvent regrouper plusieurs services atomiques. En conséquence, une composition peut être soit composée de services composites, soit de services atomiques mélangés. Deux grandes techniques, l'*orchestration* et la *chrographie*, ont été proposées pour la construction de services web complexes par composition de services web plus simple.

Une orchestration est basée sur un élément central responsable de la composition dans son ensemble. Le processus réalisée par le service web composite devient alors la somme de ses sous processus et l'orchestrateur gère, seul, les échanges de messages.

Une chorégraphie décrit les séquences de messages échangés par un service web lors de son interaction avec d'autres services (typiquement l'échange public de

messages entre des services web) plutôt qu'un processus spécifique exécuté par un acteur particulier. Il s'agit donc d'une manière décentralisée de gérer une composition puisque chaque service web est responsable d'une partie du processus

1. Problématique

Pour la mise en œuvre d'une de ces techniques de composition, un nombre de plus en plus important de langages de description de services web composite ont été proposés depuis 2001. Le nombre important de ces langages revient principalement aux défauts que présente chaque langage et surtout à la difficulté de leur mise en œuvre, même dans le contexte d'Environnements Intégrés de Développement Rapide d'Applications. En effet, Les solutions actuelles de composition de services Web, représentées par la multitude de langages de composition de services web, sont fortement liées aux technologies d'implémentation. Cette situation a engendré les inconvénients suivants :

- Difficulté de compréhension et mise en œuvre : Il est très difficile de comprendre une composition de services web sans connaître tous les détails du langage avec lequel elle a été spécifiée. Une connaissance préalable des structures du langage est requise pour comprendre l'enchaînement du processus métier ;

- Les langages de composition de services web ne sont pas des langages de haut niveau, dans lesquels nous pouvons exprimer à un haut niveau d'abstraction c'est-à-dire un processus en termes des tâches à accomplir et la coordination entre ces tâches. Bien au contraire, ces langages sont de bas niveau, trop proche du niveau implémentation et langage de programmation et dans lesquels existent plusieurs détails que le concepteur doit maîtriser. De plus la spécification des propriétés non fonctionnelles est trop limitée lorsque celle ci est de niveau implémentation. De ce fait l'étude et l'évaluation du système tôt dans un processus de conception deviennent difficiles.

- Les techniques actuelles de spécification de la composition de services Web sont spécifiques aux services Web. Il n'est pas possible de les utiliser pour la

conception d'un système hétérogène composé de services web et d'autres services non web.

2. Objectifs et éléments fondamentaux de notre approche

L'objectif principal de notre travail est de rechercher une technique de composition de service web qui nous permettrait de surmonter les problèmes que nous venons de mettre en évidence. Cette technique devrait permettre

- La spécification facile et aisée de services web composite. Le concepteur n'aura pas à se soucier des divers détails de niveau implémentation. Ces détails seront en fait générés automatiquement par la technique recherchée en se basant sur des propriétés globales du système à réaliser.
- La spécification de propriété non fonctionnelle qui permettrait d'étudier tôt dans un processus de conception le service web composite à réaliser.
- La spécification de systèmes hétérogènes dans lesquels un service web pourrait être réalisé à la base de service Web et d'autres services non Web.

Afin d'atteindre les objectifs cités, nous nous sommes basés sur une idée principale qui consiste à ramener le problème de composition de Web service à un niveau d'abstraction très élevé qui est le niveau architecture. Il y a quelques années cette idée ne nous aurait mené nulle part car le niveau architecture n'existait que pour la décomposition informelle d'un système sous forme de sous systèmes moins complexe. Aujourd'hui, la phase architecture dans un processus de conception s'est dotée de concepts, outils et méthodologies qui ont donné naissance à une nouvelle discipline du génie logiciel connu sous le nom d'Architecture Logicielle. Le choix de l'Architecture Logicielle comme approche pour la composition de service Web repose sur une observation très simple. La conception de système complexe pris en charge par l'Architecture Logicielle et la conception de service web composite utilise le même principe général : c'est la conception par composition, appelée souvent par assemblage de composant en architecture logicielle. De plus les services web et l'architecture logicielle utilise un même concept concernant les brique de base : C'est le concept de composant.

Si à sa naissance et durant plusieurs années l'architecture logicielle se basait sur des composants abstrait très complexes, ce qui est souvent appelée la programmation dans le large (programming in the large), aujourd'hui plusieurs approches existent pour permettre le raffinement d'un architecture pour arriver ensuite à une spécification architecturale très proche du niveau implémentation. Ainsi la spécification d'une architecture peut aujourd'hui être exprimée à base de composants très complexes et abstraits ou à base de composants très fins complètement définis au niveau implémentation. C'est dans ce dernier contexte que les web services primitifs peuvent se positionner.

Le choix de l'approche architecture logicielle pour la résolution du problème de la composition de web services est dû aussi en grande partie au fait que l'approche architecture logicielle parait aujourd'hui unanimement acceptée comme voie prometteuse pour la production de logiciels de haute qualité. La dernière version d'UML en l'occurrence UML2.0 [1] témoigne de l'importance grandissante de l'approche architecture logicielle et de son intérêt. UML2.0 représente un pas important dans la prise en considération des concepts fondamentaux de l'architecture logicielle. C'est face à la faiblesse d'UML1.4 et UML1.5 à représenter une architecture logicielle [2] [3] et la nécessité de rendre plus efficace les approches MDA (Model Driven Architecture) [4] qu'UML2.0 fut défini. A travers UML 2.0 l'OMG a introduit un ensemble de spécifications concernant notamment les concepts de composants, ports et connecteurs. Ces derniers représentent les concepts fondamentaux sur lesquels, repose la spécification d'architecture logicielle. Les composants et les connecteurs sont décrits par leur interface, leur type, leur sémantique ou comportement, les contraintes d'exploitation, l'évolution et les propriétés non fonctionnelles [5].

La spécification d'architecture logicielle repose principalement sur deux concepts important : le concept de composant et le concept de connecteur. Un composant exprime via des interfaces les ressources dont il a besoin (appelé généralement les services requis) et les ressources qu'il est capable de produire (appelé généralement les services fournis). Une interface est le lieu où sont spécifiées les conditions d'exploitation d'un service. Ces conditions sont représentées à deux niveaux : Un

niveau structurel et un niveau comportemental. Le niveau structurel sert à contrôler si les deux interfaces connectées correspondent correctement au niveau des noms des opérations et des type de données utilisées. Le niveau comportemental, spécifié souvent sous la forme de préconditions et de post conditions, expose les règles et les contraintes d'exploitation des services fournis.

La spécification de la correspondance entre un service requis et un service fourni se fait par la définition d'un connecteur entre ces deux services. En plus de cette correspondance, un connecteur est le lieu ou sont défini les interactions entre les services connectés. Une interaction valide doit respecter le niveau comportemental spécifié au niveau d'une interface.

La spécification d'une architecture logicielle se fait à l'aide de langage de description d'architecture logicielle souvent appelé ADL (Architecture Description Language). A ce jour, un grand nombre d'ADL ont été proposés. Le modèle de composant sur lequel se base ces ADL et les niveaux d'abstraction supportés représentent les éléments les plus importants pouvant distingués les divers ADL.

Un des problèmes auquel nous étions confrontés est la détermination de l'approche architecture logicielle (donc l'ADL si celui-ci existe) qui serait la plus intéressant pour nos travaux. Dans ce contexte il fallait déterminer l'approche qui propose le modèle de composant le plus adéquat et celle qui supporte les niveaux d'abstraction les plus intéressant. A titre d'exemple l'ADL Wright qui est complètement basé sur le langage formel CSP sera tout simplement écarté car il ne supporte qu'un niveau d'abstraction trop élevé. De plus le langage Wright ne fournit aucun support pour la transformation d'une architecture spécifiée à en CSP vers un niveau moins élevé tel qu'une description en terme de composant et connecteurs UML. D'autres ADL tels que Fractal et ArchJava sont trop proches du niveau implémentation et se trouve ainsi inadapté pour une spécification de plus haut niveau d'abstraction. Dans notre cas, nous nous sommes basés sur le modèle de composant de l'approche IASA (Integrated Approach to Software Architecture) définie dans le contexte d'un travail de recherche entre l'ESI[3] et laboratoire LINA[4] de l'Université de Nantes [6]. Ce modèle permet la spécification libre à un niveau

[3] L'école nationale supérieure d'informatique http://www.esi.dz/
[4] Laboratoire d'informatique de Nantes Atlantique http://www.lina.univ-nantes.fr/

d'abstraction très proche des modèles mentaux de topologies très variées. Cette capacité est due principalement au concept d'interface de ce modèle qui permet de manipuler les éléments structurels et comportementals d'une interface contrairement aux autres approches qui considère qu'une interface est atomique.

L'approche IASA, si elle fournit une définition très poussée des modèles de composant et de connecteurs, ne s'est pas encore dotée d'un ADL officiel. La première version de l'ADL de l'approche IASA, appelé SEAL [7] a été définie juste pour démontrer le concept de composant exécutable à un haut niveau d'abstraction. Or, le modèle nécessite un langage plus complet pour la spécification des divers caractéristiques du modèle, notamment la spécification orientée aspect d'architecturelle logicielle et la spécification des propriétés de déploiement. C'est pour combler les défaillances de SEAL et pour permettre d'exploiter le modèle IASA de manière efficace que nous avons proposé le langage x3ADL (eXtensible Architecture, Aspect and Action Description Language). Avec le modèle de composant IASA et x3ADL il est possible de définir des architectures à divers niveaux d'abstraction. L'approche IASA est dotée d'un processus de développement complètement basé sur la conception par assemblage de composant. Les gros composants abstraits sont raffinés et conçus par assemblage de composants moins complexe. Le raffinement s'arrête lorsque des composants dits primitifs sont utilisés. Un composant primitif est complètement défini au niveau implémentation.

Une autre caractéristique qui nous a guidé vers de l'approche IASA, c'est la spécification explicite des propriétés de déploiement pour chaque instance de composant[5]. Ainsi il est possible de spécifier pour une instance de composant la nature réelle que l'instance de composant aura à l'exécution. Ainsi, selon l'approche IASA, si un type de composant appelé TC1 a été instancié deux fois et que les noms de ces instances sont I1C1 et I2C1, il est alors possible de spécifier que I1C1 doit être déployé comme un composant EJB et I2CI comme un Web service. Dans IASA, le processus de transformation d'une instance de composant dans la nature spécifié par les propriétés de déploiement repose essentiellement sur le concept d'enveloppe qui est associé à chaque instance de composant.

[5] Un composant est un type dans l'approche IASA

3. Notre Approche:

L'objectif principal de notre étude est de résoudre le problème de composition de service Web en se basant sur une approche architecture logicielle qui est l'approche IASA. Cette dernière nous permettrait de porter le raisonnement à un haut niveau d'abstraction caractérisé par les éléments suivants:

- La facilité de la spécification et de validation de l'architecture d'une application. Il est plus facile pour un concepteur de raisonner dans un espace très proche de ses premières idées sur le système que dans l'espace technologie d'implémentation qui est devenue actuellement assez complexe à maîtriser.
- La spécification de la solution se fait par assemblage de composant. Cette dernière est un processus simple ne nécessitant pas de grandes compétences et une maîtrise des technologies d'implémentation comme ceci est le cas avec les techniques actuelles de conception de services Web composite. La transformation d'une architecture IASA en un service web composite étant à la charge de l'ADL de l'approche IASA.
- La capacité de l'approche architecture logicielle à cacher les technologies d'implémentation et le déploiement des composants. Ainsi il serait possible grâce à l'approche architecture logicielle de réaliser des systèmes dont des parties sont réalisées en utilisant les services web et d'autres parties utiliseront d'autres technologies de réalisation.

Il faut noter que la version actuelle d'IASA ne supporte que le langage Java comme cible technologique. Un composant primitif IASA est représenté par un POJO[6]. Ce dernier est un code Java indépendant de toute spécificité de déploiement. Ce code serait enrichi à travers le concept d'enveloppe par le code adéquat qui ferait de lui un Thread, une applet, une Servet, un EJB, et pourquoi pas un service Web primitif.

[6] Plain Old Java Object : classes Java pures ne dépendant d'aucun cadre de programmation (framework)

Actuellement ces cibles technologiques ne sont pas capables de supporter les services Web. Ainsi parmi les objectifs de notre travail, il y aura la recherche d'une méthode pour la prise en charge du processus de transformation d'une architecture IASA vers une architecture orientée service à base de services web.

4. Organisation du mémoire :

Nous avons organisé ce mémoire comme suit :

Dans le premier chapitre, nous présentons les approches orientée services, ses éléments fondamentaux, ainsi que ses différents modèles d'exécution. Nous accordons une attention particulière à notre contexte de recherche qui est bien évidemment les services Web.

Le second chapitre expose un état de l'art sur l'un des plus grands problèmes actuels dans le monde des services Web à savoir « la composition des services web ». Nous citons quelques approches industrielles et nous donnons un aperçu sur quelques travaux de recherche.

Le troisième chapitre est consacré à l'architecture logicielle, nous mettons l'accent sur les concepts de base de l'architecture logicielle ; nous nous focalisons essentiellement sur le modèle de composant qui constitue la brique de base pour notre contribution, celui de l'approche intégrée.

Dans le quatrième chapitre, nous présentons la composition de services web selon l'approche IASA, nous parlons aussi des concepts que nous avons ajouté à IASA pour l'adapter aux services Web.

Dans le cinquième et dernier chapitre ; nous présentons l'environnement de développement que nous avons développé pour valider notre travail en argumentant avec quelques exemples.

Nous terminons ce mémoire par une conclusion générale ainsi qu'un ensemble et de perspectives des travaux en cours et à venir.

CHAPITRE 1
L'ARCHITECTURE ORIENTEE SERVICES ET LES SERVICES WEB

1.1 Introduction

L'architecture orientée services est de plus en plus recommandée comme approche efficace permettant d'intégrer des services flexibles et réutilisables pouvant être rapidement déployés et modifiés. Dans ce chapitre nous mettrons l'accent sur ce type d'architecture. Nous commencerons tout d'abord par introduire les concepts clés de l'architecture orientée services et nous passerons ensuite aux Services Web.

Nous présenterons les diverses définitions des Services Web, leurs caractéristiques. Nous discuterons du modèle d'architecture en couche des services Web et nous exposerons en même temps les principaux standards mis en œuvre dans le monde des Services Web. Ce chapitre sera achevé par la présentation des principaux axes de recherches, à savoir la sélection, la découverte, la substitution et la composition des services Web.

1.2- Les architectures orientées services

L'architecture orientée service, connue sous l'acronyme SOA (Services Oriented Architecture), apporte beaucoup de nouveautés au monde des systèmes d'information et à l'informatique en général. Les entreprises opèrent de plus en plus leurs applications métiers par le biais de cette architecture. Il s'agit d'un paradigme fondé sur la description de services et sur la description de leurs interactions [8].

L'objectif d'une architecture orientée services est de décomposer une fonctionnalité en un ensemble de fonctions basiques (services), et de décrire finement le schéma d'interaction entre ces services. Le résultat de cette communication peut consister en un simple retour de données ou en une activité (coordination de plusieurs services).

Un Service est un composant logiciel [20], exposant les fonctionnalités à forte valeur ajoutée d'un domaine métier [9]. Un tel composant peut correspondre à une unité de traitement exécutable. Par exemple en J2EE[1], il peut s'agir d'un EJB[2] session ou d'une Servlet Java.

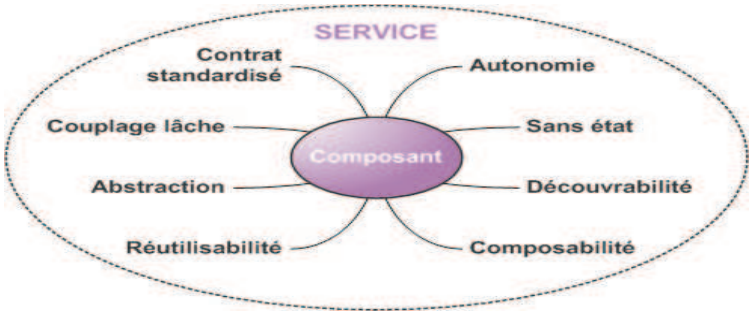


Figure 1. Présentation d'un service

1.2.1- Les caractéristiques d'un service

Thomas Erl [10] décrit le service par les caractéristiques suivantes (Figure 1) :

1. Interface **:** Les services d'un même système technique sont exposés au travers des interfaces respectant les mêmes règles de standardisation**.** Un service peut implémenter plusieurs interfaces et plusieurs services peuvent implémenter une interface commune.
2. Couplage lâche **:** il s'agit d'introduire le minimum de dépendances entre les services pour permettre d'assembler ceux-ci aisément, cela se fait via des standards comme le langage XML (eXtensible Markup Language).
3. Abstraction **:** Le principe d'abstraction consiste à fournir les services d'un Système d'information sur un modèle boîte noire. les seules informations accessibles aux consommateurs d'un service sont celles contenues dans son interface. Ainsi, les concepteurs et développeurs d'un consommateur de service ne sont pas au courant de la façon dont est implémenté le service.
4. Réutilisabilité : Un service doit être positionné comme une ressource réutilisable.

[1] Java 2 platform Enterprise Edition
[2] Enterprise JavaBean

5. Autonomie : Les services sont indépendants dans leur finalité et leur exécution. Autrement dit ces services sont développés, déployés et administrés indépendamment les uns les autres. Ils fournissent des fonctionnalités directement utilisables.
6. Sans état : Un service doit minimiser la consommation de ressources en déléguant la gestion des informations d'état quand cela est nécessaire, il reçoit les informations nécessaires dans la requête d'un client.
7. Découvrabilité : Un service peut être découvert et interprété de façon effective.
8. Composabilité : Un service doit être conçu de façon à participer à des compositions de services.

1.2.2- Eléments de l'architecture orientée services:

L'interaction dans une architecture orientée services, s'articule autour de trois grands rôles ; un producteur de service (fournisseur de service), un consommateur de service (client) et le répertoire de services (registre de stockage des services ou annuaire)

Le producteur a pour fonction de déployer un service sur un serveur et de générer une description de ce service. Cette dernière précise à la fois les opérations disponibles et leur mode d'invocation. Cette description est publiée dans un répertoire de services. Les consommateurs peuvent découvrir les services disponibles et obtenir leur description en lançant une recherche sur un répertoire. Ils peuvent ensuite utiliser la description du service ainsi obtenue pour établir une connexion avec le fournisseur et invoquer les opérations du service souhaité (Figure 2) [8]

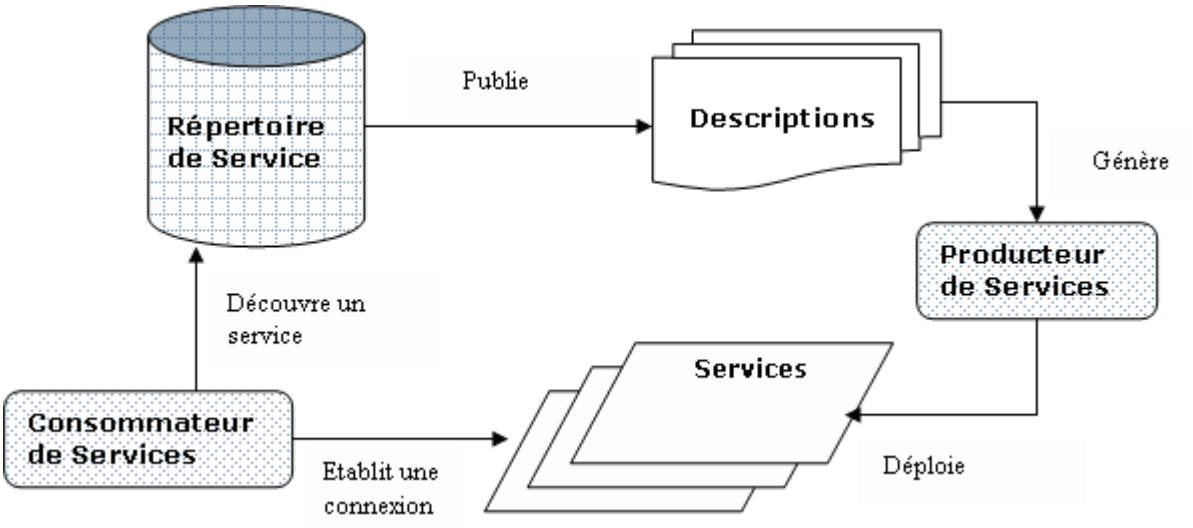


Figure 2. Les interactions dans une architecture orientée services

1.2.3- Les technologies mises en œuvre en architecture orientée services

Généralement, les entreprises développent leur propre solution SOA ou plate-forme d'intégration, comme elles peuvent se contenter de plates-formes SOA existantes. Les premières technologies à base de services sont DCOM et ORB, D'autres approches plus récentes ont émergé comme les bus de services et l'architecture SCA.

- DCOM (Distributed Component Object Model) : il s'agit d'une technologie proposée par Microsoft pour réaliser des appels distants à des objets COM (Component Object Microsoft).

- ORB (Object Request Brokers) : est une entité qui fournit des mécanismes d'interrogations permettant de récupérer des objets, des procédures qui constituent une application.

- CORBA (Common Object Request Broker Architecture): CORBA est un bus qui propose un modèle orienté objet de coopération entre les applications réparties. Chaque application peut exporter ses services sous forme d'objet CORBA. CORBA s'est avéré très complexe à implémenter. Le cœur de CORBA est constitué par L'ORB. Le protocole IIOP (Internet Inter ORB

Protocol) défini par l'OMG[3] (Object Management Group) permet d'assurer l'interaction entre les objets CORBA.

- SOC (Service Oriented Computing) [11] est une initiative académique qui vise à étendre le modèle d'architecture orientée service pour permettre d'administrer les services de façon flexible. Les activités de l'équipe de recherche SOC visent deux objectifs primordiaux [12] :

    - Proposer des modèles et des techniques relatives aux fondements des services Web pour simplifient leurs découverte et composition.

    - Concevoir et développer des architectures orientées services dans différents domaines d'applications comme le médical, les systèmes collaboratifs, le partage de ressources, le eGovernment, etc.

- Le bus de services, aussi nommé ESB (Enterprise Service Bus) a un rôle de médiateur entre le fournisseur et le client du service [13]. Il est défini au dessus d'une architecture distribuée combinant des services et des technologies traditionnelles pour l'intégration (courriers, systèmes de routage et de transformation de données, etc.) [14].

- SCA (Service Component Architecture), propose un modèle de composants pour construire des applications respectant le paradigme SOA. SCA encourage une organisation basée sur des composants explicites qui implémentent la logique métier et communiquent au travers d'interfaces de services. Le modèle distingue l'étape d'implémentation des composants de celle d'assemblage de ces composants [15].

- Les services web : les deux modèles CORBA et DCOM ont ouvert la voie aux services web. Les interactions dans les systèmes d'information qui découlent de ces deux technologies doivent être soigneusement décrites Une modification dans un service conduit généralement à une défaillance du système. Les

[3] **Object Management Group** est une association américaine à but non lucratif créée en 1989 dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes.

services web s'inscrivent dans la continuité de ces deux standards (CORBA et DCOM) en apportant toutefois une réponse plus simple, en s'appuyant sur des technologies et standards reconnus et qui sont aujourd'hui largement acceptés par tous [16].

1.3- Les services Web

Les services Web sont la déclinaison du paradigme des architectures orientée services, sur le Web. Ils ont été proposés initialement par IBM et Microsoft, puis en partie standardisés par W3C[4]. Actuellement, dans la littérature, Le concept «Service Web » est le sujet de définitions très variées.

Nagy et al. [17] définit un service Web comme étant « une application accessible à partir du Web et qui utilise les protocoles d'Internet pour communiquer et un langage standard pour décrire son interface »

En 2003, J.Daniel [18] donnait la définition suivante ; « les services Web sont des applications auto descriptives, modulaires, indépendantes et faiblement couplées qui fournissent un modèle simple de programmation et de déploiement d'applications, basés sur des normes s'exécutant à travers l'infrastructure Web, et qui peuvent être découvertes et invoquées dynamiquement via Internet »

Le consortium W3C considère un service Web comme un composant logiciel vérifiant les propriétés suivantes:

- identifié par une URI (Uniform Ressource Identifier) ;
- ses interfaces et ses liens sont décrits en XML ;
- sa définition peut être découverte par d'autres services Web ;
- peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant des protocoles Internet.

[4] **W3C** (**W**orld **W**ide **W**eb **C**onsortium) est l'Organisme de standardisation du web, spécifications et normes techniques (HTML, XML, CSS, DOM ...) pour le Web, basé sur la mise en place de consensus technologiques universels au-delà des lourdes pressions des grands industriels de l'informatique.

Les services web permettent d'assurer une ouverture des systèmes d'information. Ils sont préférés aux autres approches de mise en place de systèmes distribués pour deux grandes raisons : La première concerne les pare-feux. Ces derniers installés un peu partout sur internet acceptent le passage du flux d'information relatifs aux Services Web. Le deuxième est l'utilisation exclusive d'XML comme format de description des diverses ressources et mécanismes, ce qui leur procure l'avantage d'être non propriétaire et multi plateforme.

L'objectif ultime de l'approche services web est de transformer le Web en un dispositif distribué de calcul où les services peuvent interagir de manière intelligente en étant capables de se découvrir, de négocier entre eux et de se composer en des services plus complexes [19].

1.4- Les couches de l'architecture orientée service Web

Le groupe de travail WSA[5] du W3C a proposé une vision multicouche de l'architecture orientée services. Cette vision n'étant pas unique, nous avons choisi une représentation simplifiée proposée par [8] et présentée dans la Figure 3.

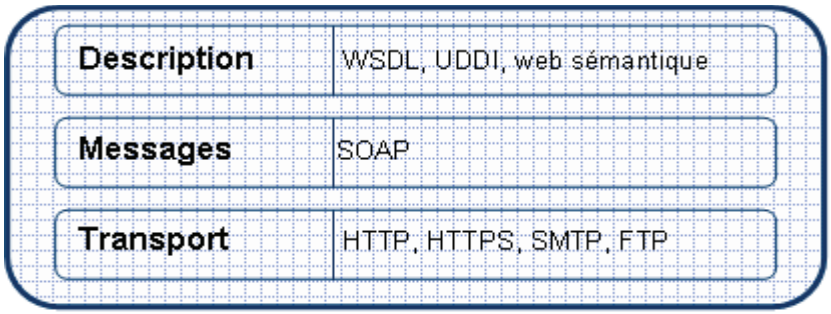


Figure 3. Les différentes couches des architectures orientées service Web

1.4.1- La couche « transport » [6]

Cette couche de base définit les aspects liés au transport des messages B2C ou B2B, ainsi que le style et le mode des communications. Le protocole HTTP

[5] WSA : Web Service Architecture du W3C a pour objectif de guider la progression des spécifications liées aux services Web

[6] La couche transport défini pour les Web services ne correspond pas à la couche transport d'OSI

représente le protocole de communication le plus utilisé par cette couche de transport. D'autres protocoles comme SMTP[7] et FTP, sont rarement utilisés.

Deux styles de communication peuvent être mis en oeuvre: le style RPC (Remote Procedure Call) et le style Document. En RPC, le Web service est similaire à l'approche distribuée traditionnelle. Le style Document correspond à une communication où le client interagit avec le serveur non plus sous la forme de procédures mais de documents XML auto-descriptifs qui sont échangés. Concernant les modes de communication mis en œuvre, nous distinguons trois modes : le mode RPC ou mode « requête –réponse », le mode « one-way-messaging » et le mode « asynchronous callback ».

1.4.2- La couche « messages » :

La communication par messages constitue un point central dans toutes architectures orientées service Web. Ces messages sont basés sur XML qui permet l'échange de données structurées, indépendamment des langages de programmation ou des systèmes d'exploitation. Les types de données utilisés sont eux aussi décrits en XML. C'est ce qu'on appelle l'encodage de type. Cet encodage peut se faire selon l'une des deux approches suivantes :

- Literal : suit littéralement les définitions des schémas XML
- SOAP encoded : suit la spécification du protocole SOAP (Simple Object Access Protocol).

1.4.2.1- Le protocole SOAP (Simple Object Access Protocol)

Le formalisme XML étant basé sur du texte, il est tout à fait adapté pour être véhiculé par le protocole de communication tel que HTTP pour assurer la communication entre les participants d'un service Web. Le couple HTTP/XML est donc naturellement utilisé. Les messages exprimés à l'aide de XML respectent un format standard définis par le W3C dans le contexte du protocole SOAP.

[7] **SMTP** est un protocole de communication utilisé pour transférer le courrier électronique.

SOAP est le protocole d'échange de messages permettant d'interagir avec les services Web. SOAP est aujourd'hui dans sa version 1.2 [21]. Le transfert des messages SOAP se fait le plus souvent à l'aide du protocole http. SOAP peut être également transporté par d'autre protocole de niveau application tels que SMTP et FTP. Le protocole HTTP est souvent employé car il illustre parfaitement la capacité de SOAP à s'adapter aux échanges sur Internet réutilisant une technologie largement déployée et acceptée.

Le protocole HTTP est un protocole simple capable de s'accommoder à la fois de la qualité de service et des temps de latence très variables de l'Internet; ce que n'est pas le cas par exemple des protocoles d'environnements répartis tels que DCOM ou CORBA. De plus, en s'appuyant sur HTTP, on ne se heurte pas aux problèmes de pare-feu (firewall) ou de configuration de réseau IP qui rendent parfois difficile le déploiement d'applications à objets répartis au-delà du périmètre du réseau d'entreprise [16].

Un message SOAP est un document XML dont la structure est spécifiée par des schémas XML. Plus précisément tout message SOAP se compose d'un élément enveloppe qui englobe un élément entête et un élément corps (Figure 4).

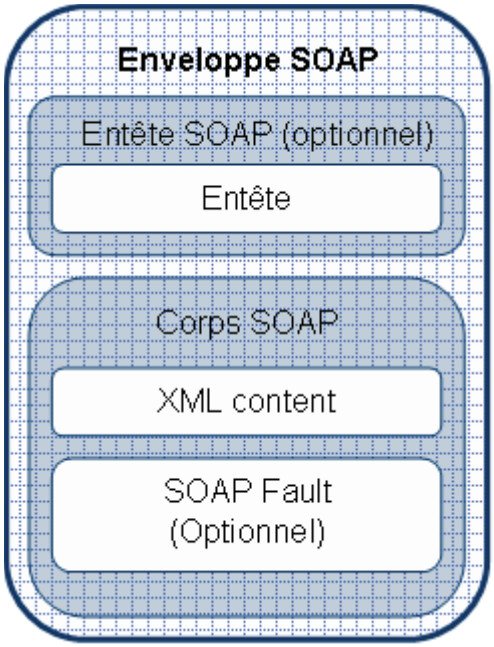


Figure 4. Structure d'un message SOAP

La partie entête contient l'entête du protocole de transport (par exemple HTTP) ainsi que les métadonnées qui portent sur d'éventuelles propriétés non fonctionnelles du service (jeton de sécurité, contexte de transaction, certificat de livraison, etc.).
La partie corps regroupe, quant à elle, les éléments métiers tels que :

- Les appels de méthodes, avec transferts de données spécifiques, dans le cadre d'une requête (le nom de la méthode ainsi que la valeur de ses paramètres),
- Seulement les transferts de données spécifiques dans le cadre d'une réponse (la valeur des paramètres de retour de la méthode).

1.4.3- La couche « description » :

Selon le modèle de référence OASIS[8] (Organization for the Advancement of Structured Information Standards) [22] une description de service représente les informations nécessaires permettant d'exploiter un service. Elle doit faciliter la visibilité et l'interaction entre les consommateurs et les fournisseurs de services. Le protocole SOAP ne fournit en aucun cas une indication sur la structure que le message doit respecter vis à vis du service Web sollicité.

Un service Web est décrit à l'aide du langage WSDL (Web Service Description Language). La spécification WSDL a vu le jour pour offrir une grammaire qui décrit l'interface des services Web de manière générique. Les deux standards, SOAP et WSDL, définissent ensemble l'aspect le plus basique du développement de l'infrastructure des services Web.

1.4.3.1- WSDL – Web Service Description Language

WSDL [23] est un langage de description fondé sur XML. Il a été soumis au W3C comme standard industriel pour la description des services Web. La spécification WSDL présente les services comme des boites noires et s'intéresse à

[8] Consortium qui dirige le développement, la convergence ainsi que l'adoption des standards de l'e-commerce. Il comprend plus de 600 membres (entreprises et individus) dans une centaine de pays.

fournir une abstraction fonctionnelle du service. La spécification du service est composée de deux parties :

Une première partie définit de manière abstraite les éléments, les opérations et les types de données, tandis qu'une seconde partie précise de manière concrète la définition des mécanismes de liaison entre les définitions abstraites et un ensemble de techniques de déploiement (généralement des protocoles Internet). WSDL joue un rôle important dans l'interopérabilité des services Web et permet de définir ce qui est nécessaire à leur invocation (on parle de la notion d'invocation de service par abus de langage car ce n'est pas le service lui même qui est invoqué mais bien une opération de ce service).

La spécification WSDL est définie selon une sémantique totalement indépendante du modèle de programmation de l'application. Elle sépare clairement la définition abstraite du service (échange de messages) de ses mécanismes de liaison (définition des protocoles applicatifs). Cette dernière caractéristique permet d'interagir avec un service même si ce dernier a été modifié, ce qui est un point important pour assurer l'interopérabilité des services(Figure 5).

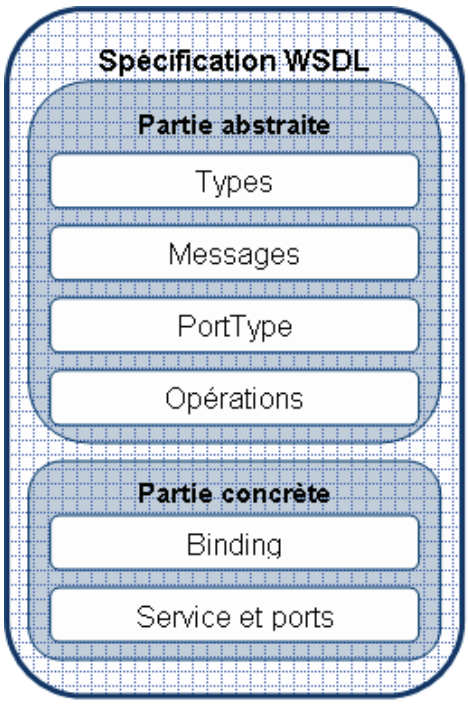


Figure 5. Anatomie d'un document WSDL

Actuellement la dernière version de WSDL est la version ”2.0” [23]. Le format général d’un document WSDL est présenté ci-dessous :

```
<definitions>
  <types> <!--type abstrait de données> </types>
  <message> <!--structure du message> </message>
  <portType><!--interface du Web service>
          <operation>
             <!-- une description d’une action supportée par le service
          </operation>
  </portType>
  <binding>
               <!--comment peut-on y accéder>
  </binding>
  <service>
           <!--qui  fournit le service>
           <port> <!--point de terminaison> </port>
  </service>
</definitions>
```

1. L’élément types : décrit sous la forme d’un schéma XML les types des données échangées entre le client et le fournisseur de services.
2. L’élément message : décrit les noms et les types d’un ensemble de champs à transmettre. Il peut être comparé aux paramètres d’un appel de procédure.
3. L’élément opérations : Décrit les opérations invoquées à l’aide des messages reçu, émis par le service et éventuellement des messages d’erreur.
4. L’élément portType : définit les opérations (collection des éléments operation) en terme de paramètres d’entrée et de sortie, il peut être comparé à une interface Java.
5. L’élément binding : spécifie le protocole de transfert (HTTP / SMTP / FTP) et le format d’encodage des données (encodage RPC, Document, etc.) pour une liste d’opérations,
6. L’élément port : un point de terminaison (End point), identifié de manière unique par la combinaison d’une adresse Internet et d’une liaison.
7. L’élément service : regroupe un ensemble de points de terminaison (éléments endpoint), offrant chacun une alternative (différents protocoles, etc.) pour accéder aux opérations du service en identifiant de manière unique la combinaison d’un élément binding et d’une adresse interne.

1.4.3.2- UDDI – Universal Description, Discovery and Integration

Dans un environnement ouvert comme Internet, le modèle de description des services Web n'est d'aucune utilité s'il n'existe pas un moyen de localiser aussi bien les services que leurs descriptions WSDL. un troisième standard de découverte des services Web a été conçu: il s'agit de UDDI standardisé par l'OASIS depuis 2005, la dernière version est la version 3.0.2 [24].

UDDI est un annuaire mondial d'entreprises s'appuyant sur le réseau Internet. Il permet d'automatiser les communications entre prestataires, clients, etc. Dans ce but, celui-ci propose plusieurs entrées : nom, carte d'identité des sociétés, description des produits et services. Et, si ces derniers sont des applicatifs invocables à distance, il fournit les références des connexions permettant de communiquer avec eux. UDDI se base sur SOAP et suppose que les requêtes et les réponses sont des objets UDDI envoyés comme des messages SOAP

L'enregistrement des services Web dans un annuaire UDDI s'effectue auprès d'un opérateur en accédant au site Web de ce dernier à partir d'un navigateur ou d'un outil intégré à un environnement de développement. Des recherches précises peuvent s'effectuer dans l'annuaire par catégories d'entreprise en utilisant des standards taxinomiques d'identification d'entreprise et par mots clés. La spécification UDDI constitue une norme pour les annuaires des services Web. Les fournisseurs disposent d'un schéma de description permettant de publier des données concernant leurs activités, la liste des services qu'ils offrent et les détails techniques sur chaque service. Elle offre aussi une API aux applications clientes, pour consulter et extraire des données concernant un service et/ou son fournisseur (Interrogation de service). Le fonctionnement d'un UDDI est décrit par les modèles suivants:

1.4.3.2.1- Modèle de données

Les différents composants de UDDI sont représentés sous forme de documents XML (Figure 6) : PublisherAssertion, Business Entity, Business services, BindingTemplate et Tmodel.

- Publisher Assertion : Cette partie est facultative. Elle permet de décrire l'organisation dans son intégrité si elle est divisée en plusieurs divisions. [18]
- BusinessEntity (les pages blanches): Décrivent les organisations ayant publié des services dans le répertoire BusinessKey ;
- BusinessService (les pages jaunes): Décrivent de manière non technique les services;
- BindingTemplates : Spécifient les coordonnées des services;
- Tmodel (les pages vertes) : Décrivent de manière technique les services;

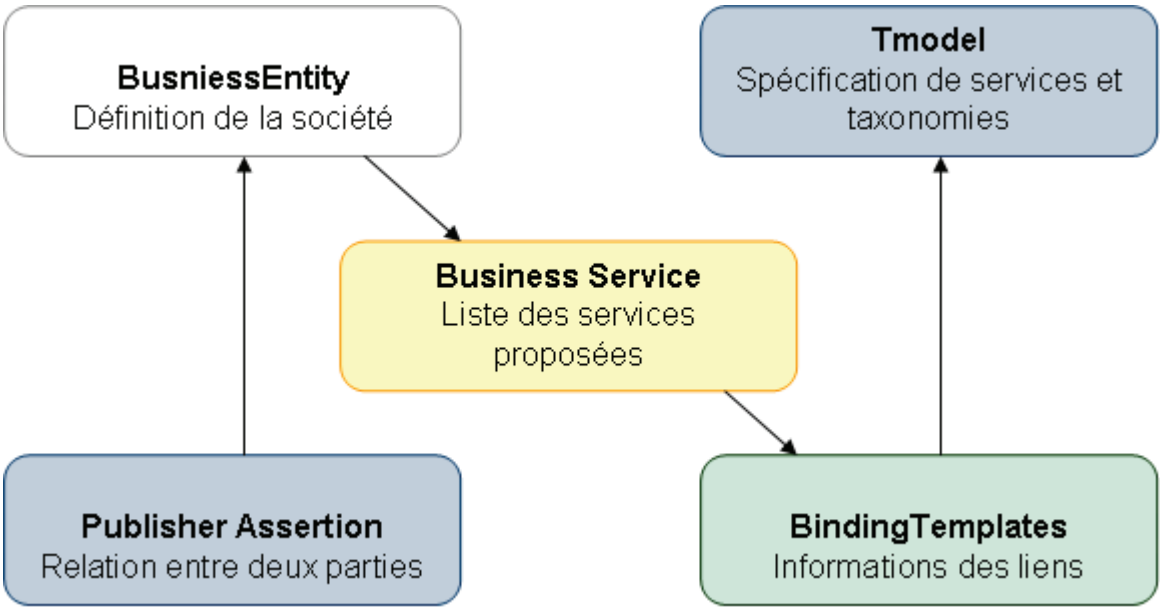


Figure 6. Architecture d'UDDI

1.4.3.2.2- Modèle de programmation:

UDDI est divisée en une interface de programmation pour l'enregistrement de services Web dans un annuaire UDDI et une interface de programmation pour la recherche d'informations. Cette API est composée de 2 grandes bibliothèques :

- API de requête (interrogation du Registre).
- API de publication;

1.4.3.2.3- Modèle d'usage:

Publication de service, Découverte de service et Description d'utilisation de service.

1.4.4- Le Web sémantique

Le Web sémantique, proposé initialement par le W3C, est une nouvelle infrastructure devant permettre à des agents logiciels d'aider plus efficacement différents types d'utilisateurs dans leur accès aux ressources sur le Web (sources d'information et services). Différents langages de niveau de complexité croissante sont proposés afin de mieux exploiter, combiner et raisonner sur les contenus de ces ressources.

Cette approche et les approches services Web partagent le but commun de rendre l'information sur le Web plus accessible aux machines. Un certain nombre de recherches se proposent de les coupler. Des idées partagées par ces travaux est de répondre aux limites de WSDL par l'ajout d'une couche sémantique au dessus de WSDL décrivant le quoi et le pourquoi, pas seulement le comment.

En effet, WSDL fournit une description concrète mais de bas niveau d'un service Web ; localisation, opérations disponibles, messages associés, types de données et formats de leurs paramètres d'entrée ou de sortie. Ces descriptions sont insuffisantes pour qu'un agent logiciel puisse interpréter la signification réelle des opérations WSDL. Pour que cette interprétation soit possible, il faut annoter sémantiquement les services Web. DAML-S se considère parmi les recherches qui se fixent pour but d'enrichir l'approche des services Web, qui propose une ontologie[9] de haut niveau des services Web sous forme d'un ensemble de classes. La classe SERVICE qui est le haut de l'ontologie DAML-S se voit ainsi associer un ensemble de connaissances par l'intermédiaire de deux classes. La première SERVICEPROFILE fournit, par un ensemble d'attributs et de propriétés, l'information

[9] L'ontologie constitue un modèle de données représentatif d'un ensemble de concepts dans un domaine, ainsi que les relations entre ces concepts. Elle est employée pour raisonner à propos des objets du domaine concerné.

nécessaire pour qu'un agent puisse découvrir un service. La deuxième SERVICEMODEL décrit comment utiliser le service en des termes plus abstraits que WSDL [26].

1.5- Fonctionnement d'un service web

SOAP, WSDL et UDDI, sont les trois standards utilisés dans les architectures orientées service Web. La figure 7 résume le principe de fonctionnement de cette architecture.

1- Une fois le service Web créé par le fournisseur, il sera déployé sur Internet et son WSDL sera publié dans le registre UDDI.
2- Le client (ayant des besoins spécifiques) va rechercher un service correspondant à ses critères à l'aide du registre UDDI.
3- Une fois le service trouvé, le client va récupérer sa description (document WSDL correspondant au service)
4- Le client va invoquer le service ; une communication va être mise en place entre l'utilisateur et le service Web.

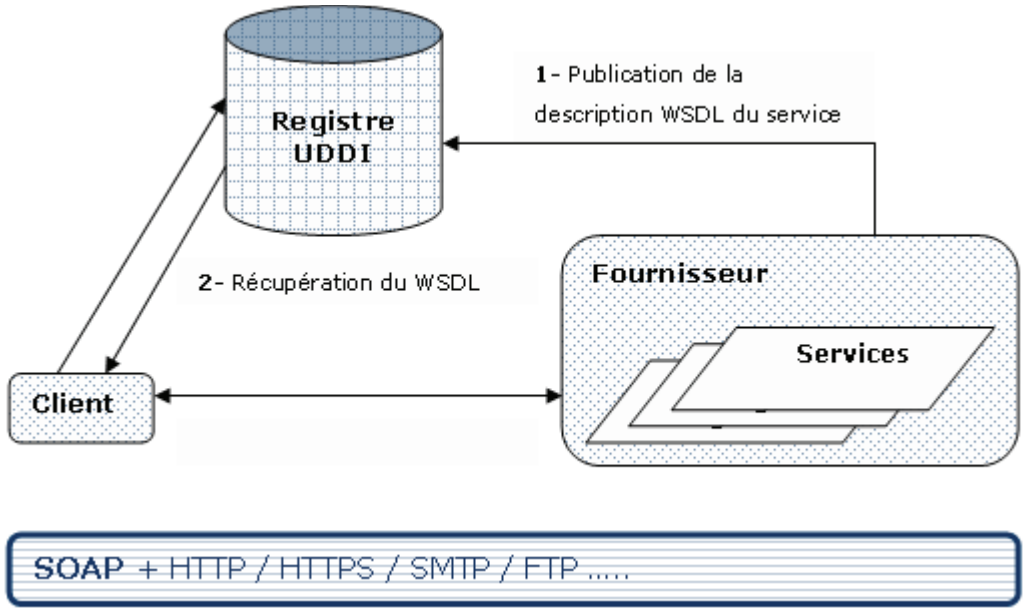


Figure 7. Cycle de vie d'un service Web

1.6- Les problématiques de recherche sur les services Web

Actuellement, les sujets de recherche dans le domaine des services Web sont très nombreux. Les plus actifs tournent autour de deux grands axes, le premier couvre la découverte de services Web et ses sujets rattachés tel que la sélection et la substitution, le second s'intéresse à la composition des services Web.3

1.6.1- La sélection des services Web

La sélection des services Web, consiste à choisir le meilleur fournisseur d'un service Web parmi un ensemble de fournisseurs, ce choix se fait sur la base de la qualité de Service (QoS*)*. La qualité de services désigne l'aptitude d'un service à répondre adéquatement à des exigences, exprimées ou implicites, qui visent à satisfaire ses usagers. Le concept de QoS recouvre un large domaine de propriétés inhérentes au domaine de la transmission de données. Plusieurs travaux pour définir la QoS dans un système distribué sont à l'origine d'une proposition de standardisation [27]. La référence normative est le « Quality of Service Framework » publiée dans [28]. Dans le cadre des services Web, le W3C a identifié un ensemble de caractéristiques de QoS pertinentes pour le domaine des services Web [29] :

- La performance : représente la vitesse avec laquelle un service Web répond à une requête. Cette qualité peut être mesurée en termes de débit, temps de réponse, latence, temps d'exécution, temps de transaction, etc.
- La confiance : qualité d'un service à exécuter certaines fonctions sous certaines conditions et dans un intervalle de temps donné. Il s'agit d'une mesure de la capacité d'un service à maintenir sa qualité (généralement le nombre de coupures de service par jour, semaine, mois). La confiance est également associée à la garantie de l'ordre et de la bonne livraison des messages. Dans notre contexte, WS-ReliableMessaging est un protocole qui permet de délivrer des messages selon certaines caractéristiques de livraison.
- Passage à l'échelle : permet de quantifier le nombre de requêtes auxquelles le service peut faire face dans un intervalle de temps donné.
- La capacité : le nombre de requêtes qu'il est possible de traiter simultanément.

- La robustesse : la capacité à fonctionner alors que les données en entrée provoquent des conflits ou sont incomplètes.
- Traitement des exceptions : un grand nombre de situations ne peuvent être prédites à l'avance ce qui implique de supporter les exceptions de façon adaptée.
- La précision : le taux d'erreurs générées par le service.
- L'intégrité : propriété garantissant que l'intégrité des données et des transactions est bien respectée, afin de ne pas aboutir à une situation inconsistante.
- L'accessibilité : capacité à répondre à des requêtes.
- La disponibilité : le service devrait être prêt pour une consommation immédiate lors d'une invocation. La disponibilité est souvent associée au temps de réparation (ou TTR[10]).
- L'interopérabilité : capacité à pouvoir communiquer quels que soient la plateforme et les langages utilisés. En ce qui concerne les services Web, c'est la normalisation et l'extensibilité de SOAP et des méta-données qu'il contient qui permet de garantir l'interopérabilité.
- La sécurité : il existe plusieurs traitements permettant de sécuriser les communications entre services Web (Authentification, Autorisation, Confidentialité, Traçabilité, Cryptage de données, Non répudiation). De multiples normes sont à mettre en relation avec la sécurité, dont principalement WS-Security, WS-Trust, WS-SecureConv,WS-Federation, etc.

1.6.2- La découverte des services Web

Dans le contexte d'une application qui a besoin d'exécuter une fonctionnalité implémentée comme un service Web par plusieurs fournisseurs, la découverte fait référence au processus de recherche des services Web implémentant la fonctionnalité souhaitée. Les registres UDDI sont les entités qui servent d'appui à la découverte de services web pour les applications client. De cette façon une application interroge un registre UDDI pour les fournisseurs d'un service Web. Parmi

[10] Time to Repair

les solutions proposées pour cette problématique : OWL-S (Ontology Web Language - Service), Web Services Dynamic Discovery et Web Service Discovery Architecture.

1.6.3- La substitution des services Web

Comme son nom l'indique, il s'agit de substituer un service Web par un autre dans les cas suivants :

- Pendant l'exécution, le service Web échoue ou ne peut pas être atteint.
- Il y a un nouveau service Web qui fourni un service mieux, en ce qui concerne les paramètres de qualité de service.

1.6.4- La composition des services Web

Les services Web sont conceptuellement limités à des fonctionnalités relativement simples qui sont modélisées par une collection d'opérations. Toutefois, pour certains types d'application, il est nécessaire de combiner un ensemble de services Web basiques pour obtenir des services Web plus complexes, afin de répondre à des exigences plus complexes.

1.7- Conclusion

Les services Web ont ouvert la voie vers la production d'une nouvelle génération des systèmes d'information caractérisés par une meilleure intégration des applications hétérogènes. La grande acceptation des services web et le nombre de plu sen plus croissant de travaux de recherche sur les divers aspects des services web sont du principalement à l'utilisation exclusive de standard tels que SOAP, WSDL et UDDI et de XML. C'est grâce à ces standards que les applications de diverses organisations peuvent dialoguer sans se soucier des langages avec lesquels ont été implémenté ni des plates forme d'exécution sur lesquelles elles tournent. Cette interopérabilité assurée par les services Web prévoit son utilisation de plus en plus croissante dans un futur proche et engendre une piste de recherche très active pour la communauté des chercheurs. Nous nous concentrons dans ce mémoire sur le problème de la composition des services Web, que nous allons présenter dans le chapitre qui suit.

CHAPITRE 2
LA COMPOSITION DES SERVICES WEB : ETAT DE L'ART

2.1- Introduction

Le développement d'application à base de services Web s'appuie sur la composition de ces derniers afin d'obtenir un service Web composite plus riche et plus intéressant. Composer plusieurs services Web pour engendrer un seul service Web est devenu aujourd'hui un grand défi. Plusieurs travaux de recherches ont été mené et ont donné naissance à ce jour à deux grandes philosophies : l'orchestration et la chorégraphie. Divers langages de composition de services Web, basés sur ces deux techniques ont été proposés. Cependant leur utilisation reste toujours trop limitée. Dans ce chapitre nous présentons les différents langages de composition ainsi que quelques travaux de recherches

2.2. Composition de service Web

Pour accomplir une tâche complexe, il est parfois nécessaire de composer plusieurs services plus simples en les connectant adéquatement. Les services échangent des messages pour se synchroniser dans leur exécution et se procurer des données. Une telle collaboration entre services est appelée composition de services. La composition de services web et donc le processus consistant à combiner des services existants pour former de nouveaux services. A partir d'un ensemble donné de services Web disponibles et une requête d'un client, le problème de la composition de services Web est de synthétiser un nouveau service composite qui réalise la requête du client. Une telle spécification peut être obtenue statiquement ou dynamiquement.

2.2.1 Composition statique

Ce type de composition peut être appliqué dans des environnements « stables » où les services Web participants sont toujours disponibles et où le comportement du service composite est le même pour tous les clients. La composition des services Web prend place durant l'étape de conception au moment où la conception du système logiciel est planifiée. Les services sont choisis, reliés entre eux, compilés et enfin déployés. Le service composite ainsi obtenu fonctionnera bien tant que son environnement et les services qui le composent ne changent pas ou ne changent que rarement. Les stratégies de récupération en cas d'échec des services, tel que substitution du service ou la gestion des erreurs, sont prédéfinis. Ce type de composition est l'approche adoptée par l'industrie. Elle est définie à l'aide de processus métier [1] ; autrement dit à l'aide de l'orchestration et la chorégraphie.

2.2.1.1 L'orchestration

Dans l'orchestration le processus principal (orchestrateur) prend le contrôle du déroulement de la composition en coordonnant les différentes opérations des différents services web. À aucun moment les autres services servant à la composition n'ont connaissance de cette composition (Figure 8): ils remplissent leur rôle de service sans se soucier si un client humain ou applicatif interagit avec eux. L'orchestration est une méthode très utilisée, et pour plusieurs raisons :

- Il est relativement simple d'écrire un processus centralisé gérant l'invocation de sous services ;
- Dans le cas d'une réutilisation de services web basiques, seule la partie centrale est à développer.

[1] [1]En anglais « Business Process » Ensemble des activités internes d'un métier dont l'objectif est de fournir un résultat observable et mesurable pour un utilisateur individuel du métier.

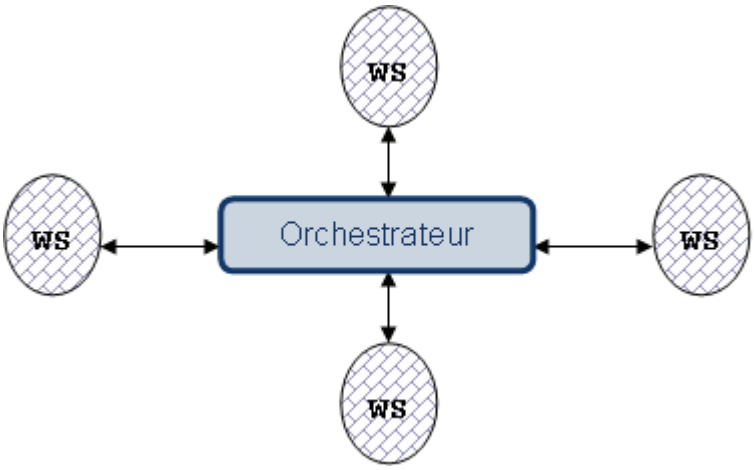


Figure 8. Orchestration de services Web

2.2.1.2 La chorégraphie :

La chorégraphie ne repose pas sur un processus principal. Chacun des services intervenant dans la composition sait exactement ce qu'il doit faire, quand il doit le faire et avec qui. Donc ils ont tous une connaissance plus ou moins globale du processus métier dans lequel ils se retrouvent (Figure 9).

Contrairement à la méthode basée sur l'orchestration, la chorégraphie demande nettement plus de développement et de test : il faut développer chaque service Web pour qu'il participe correctement dans la composition qui l'utilise. Cela dit, en utilisant des méthodes de développement et des stratégies bien pensées, l'intérêt de la méthode réside dans la décentralisation des traitements.

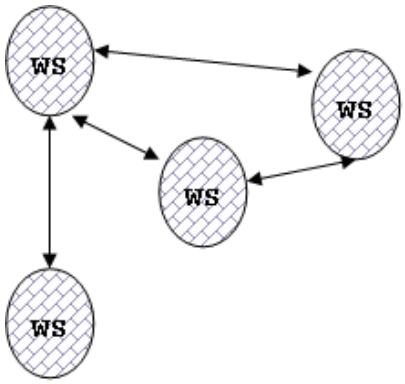


Figure 9. Chorégraphie de services Web

Il y a une différence importante entre l'orchestration et la chorégraphie de services web. L'orchestration se base sur un processus métier exécutable pouvant interagir avec les services web internes ou externes. L'orchestration offre une vision centralisée, le processus est toujours contrôlé du point de vue d'un des partenaires métier. La chorégraphie est de nature plus collaborative, chaque participant impliqué dans le processus décrit le rôle qu'il joue dans l'interaction.

L'approche la plus simple à notre avis pour les Architecture Orientées Service est l'orchestration. En effet, le but principal de la composition est la réutilisation de services (basiques ou composites) sans modifier ceux-ci. Ces services peuvent être hébergés par une autre compagnie et nous n'avons alors aucun moyen de contrôle sur ceux-ci. Le processus principal doit alors pouvoir s'adapter aux différentes erreurs possibles (exp : non disponibilité d'un service, annulation, etc). Initialement les standards se sont intéressés soit à l'orchestration soit à la chorégraphie. Récemment, un nouveau standard combine les deux stratégies [30].

2.2.2 Composition dynamique

Dans ce type de composition, les services Web à composer sont déterminés lors de l'exécution de la requête d'un client. Ils peuvent être déterminés selon les contraintes de chaque client, la disponibilité des services Web, …etc. La composition dynamique est mise en œuvre dans un environnement très dynamique dans lequel des services apparaissent et disparaissent rapidement. Elle offre le potentiel de réaliser des applications flexibles et adaptables en sélectionnant et en combinant les services de manière appropriée sur la base de la requête et du contexte de l'utilisateur.

2.3 - La vision industrielle de la composition de services web

L'industrie s'intéresse au développement d'outils communs pour la composition manuelle des services Web où tous les services sont représentés par des fichiers WSDL, et suppose un mode d'interaction atomique. Comme nous avons

déjà cité, la composition de services Web est fondée sur deux activités différentes: l'orchestration et la chorégraphie. En effet, Depuis la naissance de ces dernières, de nombreux langages de composition de services web sont apparus. Chacun de ces langages a apporté de nouveaux concepts, et a redéfini certains concepts déjà connus par les autres langages. Dans cette section nous allons présenter les principaux langages de composition de services web, qui sont (Figure 10)

- XLANG (XML Business Process LANguGe)
- BPML (Business Process Modeling Language)
- WSFL (Web Service Flow Language)
- WSCL (Web Service Conversation Language)
- WSCI (Web Service Choreography Interface)
- BPEL4WS (Business Process Execution Language for Web Services)

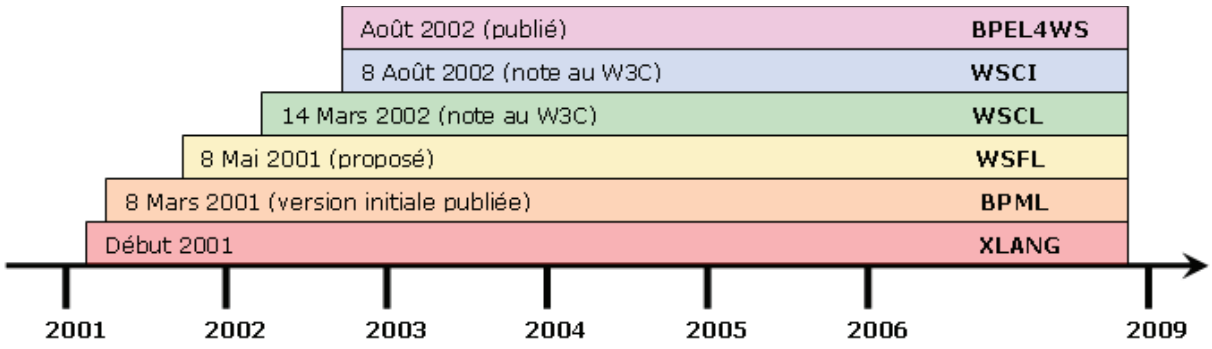


Figure 10. Six langages de composition de services web dans l'ordre chronologique de leur apparition.

2.3.1 XLANG – XML Business Process LANguGe

Le langage XLANG [31] est une extension de la spécification WSDL, créée par Microsoft et implémentée dans le produit BizTalk Server 2002. Le processus complet ne s'exécute pas sur un moteur unique, il est constitué de la collaboration entre plusieurs sous processus éventuellement répartis sur des environnements d'exécutions hétérogènes.

Le fichier XLANG contient donc la description WSDL, et y ajoute le comportement, le contexte et le contrat (Figure 11).

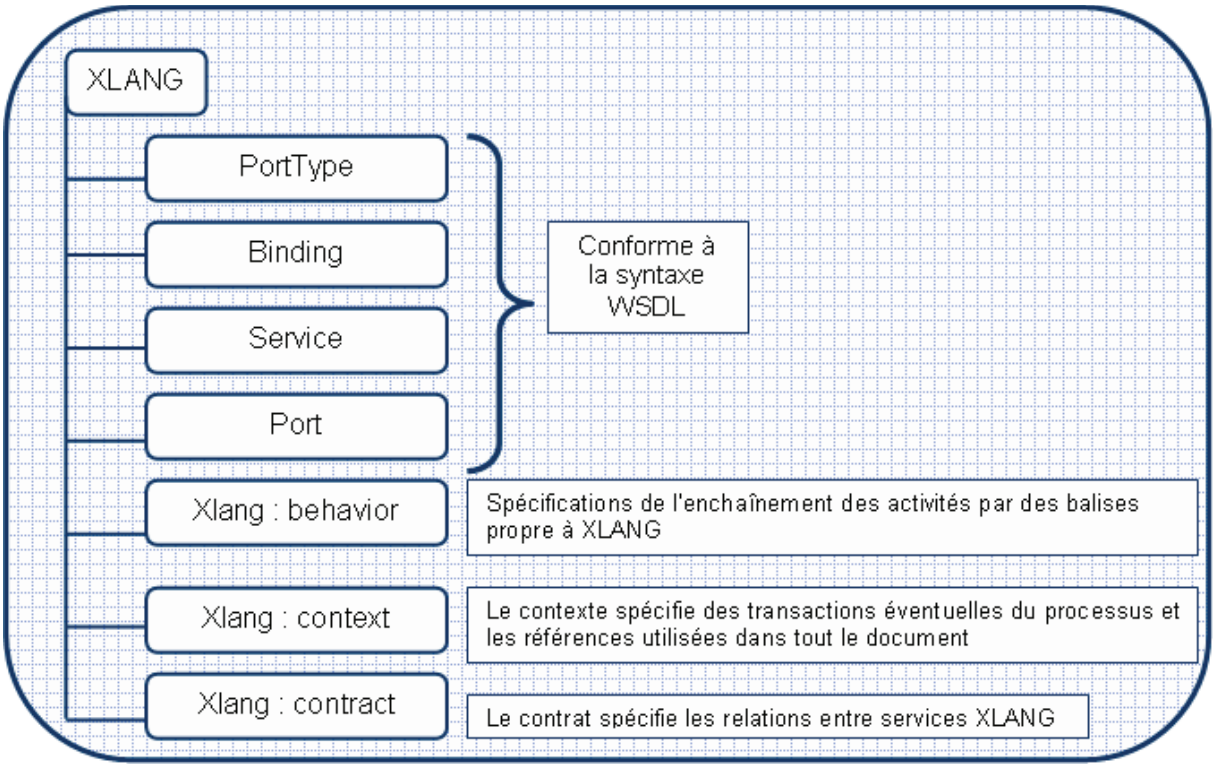


Figure 11. Structure d'un document XLANG

2.3.1.1Le comportement (behavior) :

Le comportement spécifie l'enchaînement des activités par des balises propres à XLANG. La partie comportement contient :

- Les actions XLANG : operation, delayFor, delayUntil, et raise (tableau1);

Action	Description
Operation	Une opération au sens WSDL i.e. un échange de message faisant référence à un port d'un service donné <action operation= '' AskTradePrice ''portType=operation=''pRequest''
Delayfor	Suspend l'exécution pendant le temps indiqué <delayFor period=''RequestTimeOut''>
DelayUntil	Suspend l'exécution jusqu'au temps indiqué

	<delayUntil clock=''Shipping Deadline''>
Raise	Signalisation des défaillances <raise signal = ''Invalid Login ''>

-Tableau 1 - Les Actions XLANG

- Le contrôle XLANG : qui définit l'enchaînement de l'exécution des actions, et qui peut être l'une des commandes : empty, sequence, switch, while, all, et pick (voir tab 2).

Commande	Description
Empty	Processus nul (qui ne fait rien)
sequence	Exécution séquentielle d'une ou plusieurs actions
Switch	Exécution conditionnelle
While	Boucle
All	Exécution en parallèle
Pick	Le processus attend que l'un des événements prévus se produise et exécute les actions qui lui sont associées.

-Tableau 2- Les contrôles XLANG

2.3.1.2 Le contexte (context) :

Le contexte spécifie les transactions éventuelles du processus et les références utilisées dans le document. Il permet de définir :

- des variables locales,
- des transactions.
- des signaux envoyés en cas de défaillance.

2.3.1.3 Le contrat (contract) :

Le contrat spécifie les relations entre services XLANG dans un fichier à part. Ce fichier importe les fichiers XLANG des services participant à ce contrat, et il précise la manière dont les ports de ces services sont liés.

2.3.1.4 Implémentations basés sur XLANG

Microsoft a exploité le langage XLANG dans son environnement BizTalk [32]. BizTalk contient un environnement de développement : "BizTalk Orchestration Designer", permettant de générer les "schedules" XLANG (les fichiers XLANG sont appelés schedules). Le serveur BizTalk utilise ensuite ces "schedules" XLANG pour instancier les processus à exécuter (Figure 12).

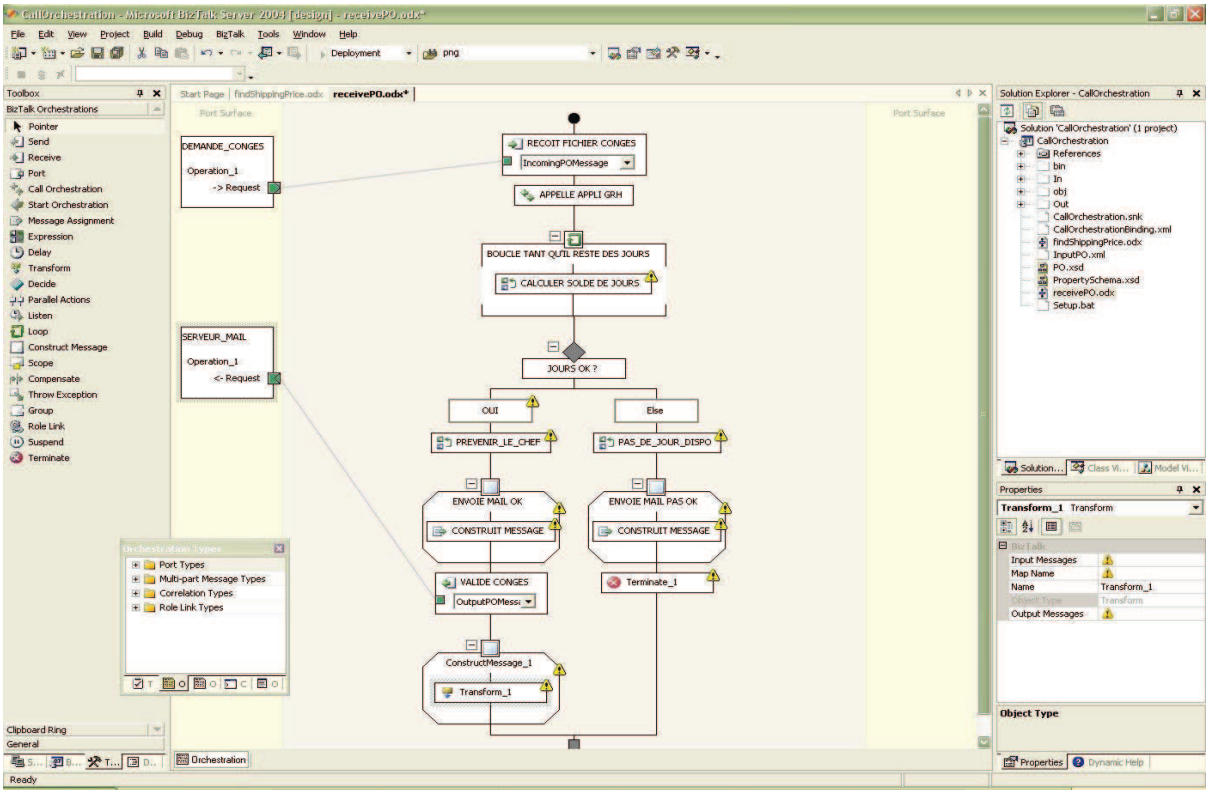


Figure 12. Interface de Biztalk server 2004

2.3.2 BPML – Business Process Modeling Language

Le langage BPML [33] est un langage proposé par l'organisation BPMI.org (Business Process Management Initiative) .Ce langage s'intéresse aux processus métier. Il gère la coordination de toute sorte de participants, et non pas seulement de services Web, ils peuvent aussi être des applications, des utilisateurs humains, des partenaires commerciaux, et d'autres processus.

Un processus métier BPML est un enchaînement d'activités simples, complexes, et de processus incluant une interaction entre participants dans le but de réaliser un objectif métier.
Les éléments de définition d'un processus sont :

2.3.2.1 Les messages :

Le message est l'unité d'interaction du processus. Les interactions à base de messages modélisent le stockage et la recherche de données, l'invocation de méthodes, et la gestion d'éléments de travail.

2.3.2.2 Les participants :

Les participants peuvent être des systèmes, des applications, des services Web, des utilisateurs humains, des partenaires commerciaux, et d'autres processus.
Il existe deux types de participants dans BPML :

- Les participants statiques : où l'identité et le comportement sont connus à l'avance.
- Les participants dynamiques : ajoutés au moment de l'exécution.

2.3.2.3 Les activités :

L'activité est l'unité d'exécution du processus. Les participants et les processus sont représentés en BPML comme des activités productrices et consommatrices de messages. Il existe deux classes d'activités dans BPML : les tâches simples, et les tâches complexes composées à partir de tâches simples.

2.3.2.4 Les transactions :

L'activité peut posséder un attribut supplémentaire afin de spécifier si elle est exécutée dans un contexte transactionnel. Il est également possible de définir des compensations pour les transactions longues partielles.
BPML définit deux modèles de transactions : *Coordonnées* pour les délais courts et *Etendues* pour les délais longs.

2.3.2.5 Les exceptions :

BPML définit un système de gestion d'exceptions, en ajoutant un dispositif de récupération d'erreur au moteur de processus.

2.3.2.6 Les règles métier :

Les règles du processus gouvernent le choix des tâches, la gestion de la consommation ou de la production de messages, et la réaction aux erreurs. Elles sont du style

Si {conditions} alors {actions}.

Les conditions portent sur les variables globales du processus ou sur les données échangées entre participants. Les actions déclenchent d'autres tâches BPML.

2.3.2.7 Implémentations de BPML

La société Intalio [34] a construit, en mars 2001, le produit "Intalio|n³" qui est un système de gestion de processus métier, basé sur le standard BPML, et non dépendant de la plate-forme d'exécution. Intalio|n³ contient cinq composants permettant de concevoir, déployer, exécuter, maintenir, et optimiser le processus métier. Le coeur de ce système, Intalio|n³ Server, est une machine virtuelle de processus, permettant d'exécuter les processus métier conformément à la spécification BPML [30].

En Août 2009 Intalio a créé une nouvelle plateforme "Intalio|BPM 6.0" une plate forme pour la gestion des processus métiers, ce nouveau produit se base sur un moteur pour gérer les processus métiers.

2.3.3 WSFL – Web Service Flow Language

Le langage WSFL [35] est une proposition effectuée par IBM, concernant la composition de services web.
WSFL est un langage basé sur XML et qui permet de décrire :

1. un processus métier exécutable décrivant l'enchaînement des appels d'opérations de services web (appelé flowModel);
2. une collaboration métier représentant les interactions entre services web, en explicitant les relations "producteur/consommateur de messages" entre leurs descriptions WSDL. Cette description est abstraite, basée sur les fichiers WSDL, en laissant les services web "en blanc" afin de les implémenter plus tard (appelé globalModel) [30].

2.3.3.1 La syntaxe WSFL

Dans le langage WSFL, six éléments spécifient une composition de services web Résumé dans le tableau ci après :

Elément	Description
flowSource et flowSink	Indiquent respectivement les données en entrée et en sortie de la composition
serviceProvider	indique les services web participant à la composition
Activity	décrit une opération ou une étape du dialogue
controlLink et dataLink	spécifient respectivement comment se succèdent les étapes et les données qui sont passées d'une étape à la suivante

-Tableau 3- Les éléments WSFL

Le processus WSFL est donc une succession d'opérations dont le flot de contrôle et le flot de données sont explicités séparément.
L'implémentation de l'activité est soit déléguée à un service externe (par la balise Export), soit exécutée à partir d'autres activités du processus WSFL (indiqué par la balise Internal). Dans le cas de la délégation, le service externe est à son tour soit explicitement nommé, soit implicitement indiqué par un élément "Locator" identifiant le fournisseur de service. L'activité comporte des attributs facultatifs : l'attribut "exitCondition" : condition de sortie.l'attribut "join" : condition jointe. Ainsi qu'une condition de transition facultative, qui peut être ajoutée dans les liens de contrôle

"controlLink". WSFL définit également des opérations de "contrôle du cycle de vie". Ces opérations lancent et contrôlent les activités :

Opération	Description
Spawn	crée une instance de l'activité ou de la composition, et la démarre
Call	exécute une instance d'activité ou de composition.
Enquire	renvoie l'état d'une activité ou d'une composition.
Terminate	arrête une instance d'activité ou de composition.
Suspend	suspend l'exécution en cours de l'instance.
Resume	reprend l'exécution d'une instance suspendue.

-Tableau 4- Les opérations WSFL

2.3.3.2 Exemple

Pour comprendre le principe de WSFL, nous allons dérouler l'exemple du processus métier «achat d'actions », ce dernier implique trois services web:

1. StockQuote.com : stocke le prix de chaque action en temps réel (le prix des actions change d'un moment à un autre).
2. Strong.com : gère les comptes bancaires des clients.
3. NYSE.com : responsable de la vente et l'achat d'actions.

nous souhaiterons regrouper ses trois web service en un seul service web ,pour cela nous allons suivre la procédure présentée dans le graphe suivant :

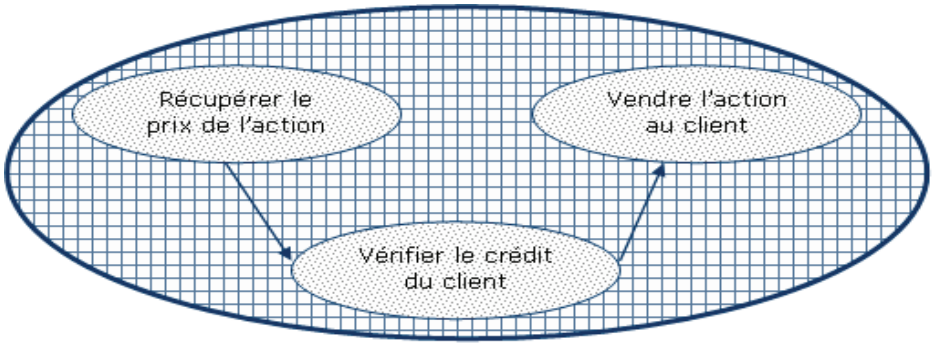


Figure 13. Procédure d'achat

Lorsqu'un client veut acheter des actions, il doit d'abord consulter le service web RTStockQuote (Figure 14) hébergé par StockQuote.com; afin de déterminer le prix de l'action, l'opération impliquée est getStockPrice

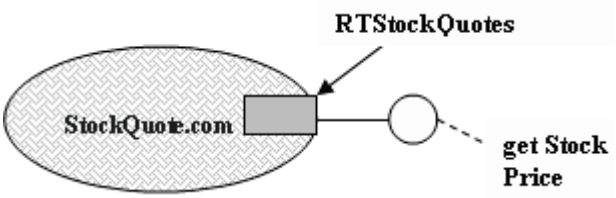


Figure 14. Service Web hébergé à StockQuotes.com

Une partie PortType du fichier WSDL correspondant à RTStockQuote est présentée ci-dessous

```
<message name="IngetStockPriceRequest">
      <part name="meth1_inType1" type="xsd:string"/>
</message>

<message name="OutgetStockPriceResponse">
      <part name="meth1_outType" type="xsd:float"/>
</message>
<portType name="StockQuoteServiceImpl_Service">
      <operation name="getStockPrice">
        <input message="InGetStockPriceRequest"/>
         <output message="OutGetStockPriceResponse"/>
      </operation>
</portType>
```

-Une partie du descripteur WSDL du service Web RTStockQuote-

Après avoir déterminé le prix de l'action, le Web service AccountManagement correspondant au Strong.com permettra de déterminer si le client a un fonds suffisant pour couvrir la transaction via l'opération checkCredit.

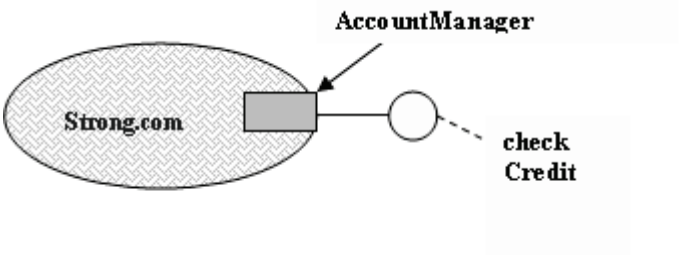


Figure 15. Service Web hébergé à Strong.com

La description du service est présentée dans le listing suivant

```
<message name="InCheckCreditRequest">
  <part name="clientID" type="xsd:long"/>
  <part name="stockPrice" type="xsd:int"/>
 <part name="quantity" type="xsd:int"/>
</message>

<message name="OutCheckCreditResponse">
  <part name="response" xsd:type="xsd:boolean"/>
</message>

<portType name="AccountManagementPT">

            ...
  <operation name="checkCredit">
        <input message="InCheckCreditRequest"/>
        <output message="OutCheckCreditResponse"/>
  </operation>

            ...
</portType>
```

-Une partie du descripteur WSDL du service Web AccountManagment-

Si le client possède le crédit nécessaire pour acheter les actions, la transaction est exécutée en invoquant le service Web VirtualTradingFloor (l'opération purchaseStockBlock) de chez NYSE.com

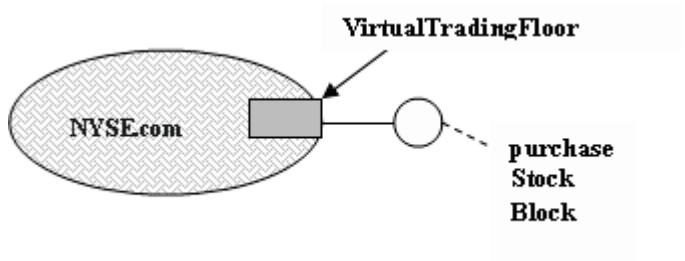


Figure 16. Service Web hébergé à NYSE.com

```
<message name="InPurchaseStockBlockRequest">
  <part name="symbol" type="xsd:string"/>
  <part name="shareCount" type="xsd:int"/>
</message>

<message name="OutPurchaseStockBlockResponse">
  <part name="result" type="xsd:boolean"/>
</message>

<portType name="MarketExchangePT">
  <operation name="purchaseStockBlock">
        <input message="InPurchaseStockBlockRequest"/>
        <output message="OutPurchaseStockBlockResponse"/>
  </operation>
```

```
</portType>
```

-Une partie du descripteur WSDL du service Web VirtualTradingFloor-

Le résultat de la composition de ces 3 services web, selon la syntaxe WSFL est représenté dans le fichier ci après :

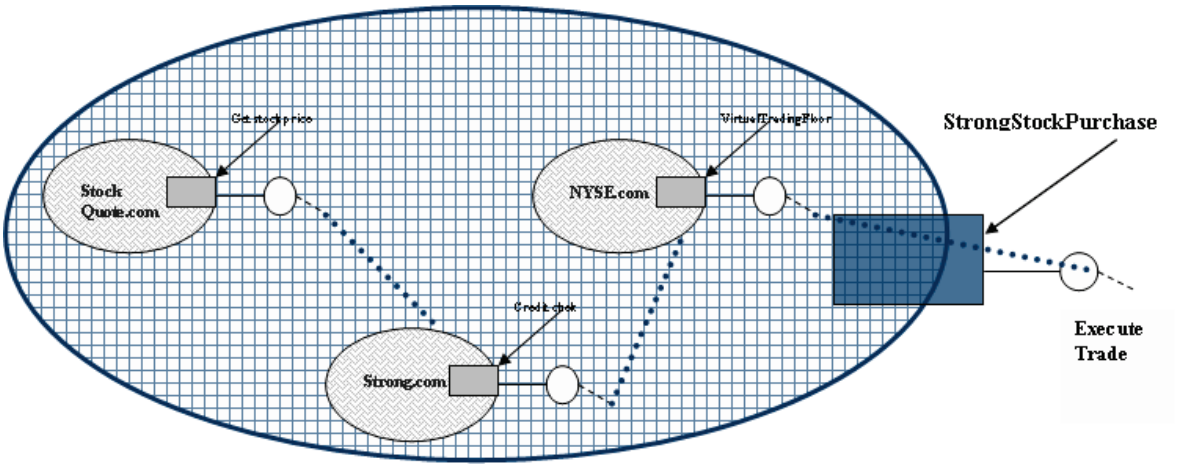


Figure 17. Service Web généré après la composition des trois web services précédents

```
<flowModel name="StrongStockPurchaseFlow"
  serviceProviderType="StockPuchaseFlow">
!-------------------------------------------------------------------------!
  <serviceProvider name="NYSE_VirtualTradingFloor"
    type="VirtualTradingFloor">
        <locator type="static" service="nyse.com/vtf.wsdl"/>
  </serviceProvider>
!-------------------------------------------------------------------------!
  <serviceProvider name="StockQuoteRealTimeStockQuoteProvider"
    type="RealTimeStockQuoteProvider">
          <locator type="uddi" bindTime="startup"         selectionPolicy="first">
      <uddi-api:find_service businessKey="uuid_key" generic="1.0"
        xmlns:uddi-api="urn:uddi-org:api">.
            <tModelBag>
              <tModelKey>
               UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4
              </tModelKey>
            </tModelBag>
      </uddi-api:find_service>
    </locator>
  </serviceProvider>
!-------------------------------------------------------------------------!
  <serviceProvider name="StrongAccountManager"
   type="AccountManager">
        <locator type="static"
            service="Strong.com/stockquoteserviceimpl_service-interface.wsdl"/>
  </serviceProvider>

!-------------------------------------------------------------------------!
  <activity name="retrieveStockQuote">
```

```
        <performedBy          serviceProvider="StockQuoteRealTimeStockQuoteProvider"/>
        <implement>
          <export>
              <target portType="StockPortfolioTrackerPT"
                                                                operation="retrieveStockQuote"/>
          </export>
        </implement>
  </activity>
!-------------------------------------------------------------------------!
  <activity name="creditCheck">
        <performedBy serviceProvider="StrongAccountManager"/>
        <implement>
          <export>
              <target portType="StockPortfolioTrackerPT"
                                                                operation="verifyCredit"/>
          </export>
        </implement>
  </activity>
!-------------------------------------------------------------------------!
  <activity name="executeTrade">
        <performedBy serviceProvider="NYSE_VirtualTradingFloor"/>
          <implement>
            <export>
                <target portType="StockPortfolioTrackerPT" operation="bid"/>
            </export>
          </implement>
  </activity>
!-------------------------------------------------------------------------!

<controlLink source="retrieveStockQuote" target="creditCheck"/>
  <controlLink source="creditCheck" target="executeTrade"
    transitionCondition="OutCheckCreditResponse/response"/>
!-------------------------------------------------------------------------!
  <dataLink name="dataLink0" source="flowSource" target="retrieveStockQuote">
        <map sourceMessage="InPurchaseStockRequest"
targetMessage="InRetrieveStockQuoteRequest"
          sourcePart="stockSymbol" targetPart="meth1_inType1"/>
  </dataLink>
!-------------------------------------------------------------------------!
  <dataLink name="dataLink1" source="flowSource" target="creditCheck">
         <map sourceMessage="InPurchaseStockRequest"
                                                  targetMessage="InCheckCreditRequest"
          sourcePart="clientID" targetPart="clientID"/>
        <map sourceMessage="InPurchaseStockRequest"
                                  targetMessage="InCheckCreditRequest"
          sourcePart="numOfShares" targetPart="quantity"/>
  </dataLink>
!-------------------------------------------------------------------------!
  <dataLink name="dataLink2" source="retrieveStockQuote" target="creditCheck">
        <map sourceMessage="OutRetrieveStockQuoteResponse"
                                  targetMessage="InCheckCreditRequest"
          sourcePart="meth1_outType" targetPart="stockPrice"/>
  </dataLink>
!-------------------------------------------------------------------------!
  <dataLink name="dataLink3" source="flowSource" target="executeTrade">
        <map sourceMessage="InPurchaseStockRequest"
                                                  targetMessage="InBidRequest"
          sourcePart="stockSymbol" targetPart="symbol"/>
        <map sourceMessage="InPurchaseStockRequest"
                                                  targetMessage="InBidRequest"
```

```
            sourcePart="numOfShares" targetPart="shareCount"/>
  <dataLink>
!-------------------------------------------------------------------------!
   <dataLink name="dataLink4" source="executeTrade" target="flowSink">
          <map sourceMessage="OutBidResponse" ="OutPurchaseStockResponse"
            sourcePart="result" targetPart="sucessfulTrade"/>
   </dataLink>

!-------------------------------------------------------------------------!
 </flowModel>
```

2.3.3.3 implémentations de WSFL

Le logiciel "MQ Series Workflow" de IBM, aujourd'hui connu sous le nom de "WebSphere Process Manager", a pris en charge la spécification WSFL, afin d'automatiser les flots de processus métier.

2.3.4 WSCL – Web Service Conversation Language

Le langage WSCL [36] est une soumission d'HP au W3C, sous forme d'une note, le 14 mars 2002. Ce langage se concentre sur la description de conversations entre paires de services web.

2.3.4.1 Concepts du langage

WSCL est un langage XML, il permet de représenter simplement les interactions entre deux services web.
La spécification WSCL est composée de quatre éléments principaux :

1. Les schémas des documents XML échangés au cours d'une conversation. Ces schémas ne font pas partie du document de spécification WSCL, ce sont des documents séparés que l'on référence par une URL dans la spécification de la conversation ;
2. Les interactions modélisant les actions de la conversation comme des échanges de documents entre deux participants. Il existe cinq types d'interactions dans WSCL (tableau 5) :

Interaction	Description
Send	émission d'un message

Receive	réception d'un message
SendReceive	émission puis réception d'un message
ReceiveSend	réception puis émission d'un message
Empty	ne contient pas de message à échanger, il est utilisé pour modéliser le début et la fin d'une conversation

-Tableau 5 - Les interactions WSCL

3. Les transitions spécifiant l'ordonnancement les relations entre les interactions. Chaque transition spécifie l'interaction source et l'interaction de destination, et éventuellement le type de message de l'interaction source ;
4. La conversation donne la liste de toutes les interactions et les transitions composant la conversation, ainsi que quelques informations additionnelles, comme le nom de la conversation, et l'interaction qui démarre la conversation, et celle qui la termine.

WSDL se charge de décrire les services Web, et WSCL se charge de décrire les conversations entre deux services Web. Ainsi, ces deux descriptions sont séparées, afin de permettre la réutilisation (une même conversation WSCL peut avoir lieu entre différentes paires de services web décrites en WSDL, de même un service web décrit en WSDL peut participer à plusieurs conversations WSCL). Cette approche différencie WSCL des autres approches, tels que XLANG et WSFL.
Un fournisseur de service peut soit fournir la définition de conversation WSCL directement l'utilisateur du service, soit l'enregistrer en tant que des "tModel" dans un annuaire UDDI, pour qu'elle soit à disposition de tous ceux qui ont accès à cet annuaire [30].

2.3.4.2 Mise en œuvre de ce langage

Aucun système permettant d'exécuter une conversation basée sur WSCL n'existe. Ce qui confirme la difficulté de réaliser une coopération décentralisée de services Web.

2.3.5 WSCI – Web Service Choreography Interface

Le langage WSCI [37] est une Initiative proposée par Sun, SAP, BEA, et Intalio. L'objectif consiste à prendre en compte la collaboration d'application à application. Cette initiative est reconnue par le W3C, depuis le 8 août 2002[38].

2.3.5.1. Concepts du langage

WSCI est un langage XML, permettant de décrire la chorégraphie entre les services web.
Ainsi, les interfaces statiques des services web sont décrites en WSDL, et la chorégraphie entre eux est décrite en WSCI. Le langage WSCI contient les concepts suivants :

2.3.5.1.1 L'interface :

Le but de WSCI est de décrire les détails du comportement d'un service web, en termes de dépendances temporelles et logiques entre les messages que ce service web échange avec d'autres services web, dans le contexte d'un scénario. Ce comportement est décrit dans un ou plusieurs processus contenus dans l'interface. Un service web peut avoir plusieurs interfaces lui permettant de jouer différents scénarios.

2.3.5.1.2. Les activités et leur chorégraphie :

WSCI décrit le comportement d'un service Web en termes d'activités chorégraphiées. La chorégraphie décrit les dépendances temporelles et logiques entre les activités.
Les activités peuvent être atomiques (activité d'envoie et/ou de réception d'un message, ou activité d'attente d'une durée définie) ou complexes (composées d'autres activités). L'activité complexe définit la chorégraphie des activités dont elle est composée. Plusieurs types de chorégraphie existent dans WSCI : l'exécution séquentielle, l'exécution parallèle, la boucle, et l'exécution conditionnelle.

2.3.5.1.3. Les processus :

Le processus contient une partie du comportement du service web, il représente l'unité de réutilisation dans WSCI. Deux types de processus sont définis dans WSCI : les processus définis au niveau de l'interface, et les processus définis à l'intérieur des activités complexes.

2.3.5.1.4. Les propriétés :

Les propriétés dans WSCI sont l'équivalent des variables dans les langages de programmation. Les propriétés sont utilisées pour éviter, dans le fichier WSCI, les références explicites vers les messages abstraits décrits en WSDL.

2.3.5.1.5. Le contexte :

Le contexte décrit l'environnement dans lequel s'exécute un ensemble d'activités. Chaque activité est définie dans un et un seul contexte.
Le contexte décrit l'environnement d'exécution en termes : des déclarations (propriétés locales ou définitions locales de processus) disponibles pour les activités, des événements d'exception qui peuvent arriver, les propriétés transactionnelles associées à l'exécution des activités. Les contextes peuvent être emboîtés.

2.3.5.1.6. La corrélation de messages :

Dans WSCI, une conversation représente un échange de messages entre deux ou plusieurs services web, participant à un scénario. Un service web peut être engagé dans plusieurs conversations en même temps, avec le même service ou avec des services différents.

La corrélation est le mécanisme par lequel un message, reçu par le service, est associé à une conversation particulière. Ainsi, les différentes conversations sont distinguées par les instances de corrélation (une instance de corrélation est un ensemble de valeurs de propriétés).

2.3.5.1.7. Les exceptions :

WSCI permet de déclarer des exceptions dans la définition de contexte, et de définir un ensemble d'activités que le service web aura à exécuter lorsque cette exception se produit. Les exceptions déclarées peuvent être : la réception d'un message d'exception, la production d'une erreur ou le dépassement de la date limite de fin.
Ainsi, lorsqu'une exception se produit, cela n'arrête pas la chorégraphie en entier, mais seulement le contexte où l'exception s'est produite après avoir exécuté les activités spécifiques à cette exception. Si le contexte ne contient pas de déclaration d'exceptions, l'exception est remontée au contexte "parent".

2.3.5.1.8. Les transactions :

WSCI permet d'associer une transaction dans le contexte. Ainsi, les activités contenues dans cette transaction seront exécutées de la manière "tout ou rien". La transaction peut contenir un ensemble d'activités de compensation, exécutées s'il y a besoin de défaire la transaction une fois qu'elle est terminée. Les transactions sont soit atomiques, soit emboîtées (composées d'autres transactions).

2.3.5.2 Implémentation de ce Langage :

Sur Microsystems a réalisé un éditeur d'interfaces WSCI : le Sun ONE WSCI Editor. Une version initiale de cet éditeur a été publiée en accès gratuit. Cette version est assez légère mais elle fournit les fonctionnalités de base pour construire des descriptions d'interfaces WSCI à partir de descriptions WSDL.

2.3.6 BPEL4WS – Business Process Execution Language for Web Services

Le langage BPEL4WS ou simplement BPEL est une spécification d'IBM, Microsoft, et BEA. Elle remplace les précédentes spécifications XLANG de Microsoft, et WSFL d'IBM. Ce langage a été défini dans sa version 2.0 par une spécification du

consortium OASIS à la fin du mois de mars 2007 sous le nom WS-BPEL afin de rester conforme au mode de nommage des extensions des Services Web [39].
BPEL4WS décrit l'interaction des processus métiers basés sur les services Web, à la fois au sein des entreprises et entre elles. BPEL4WS utilise un processus centralisé pour décrire la coordination de services web.

2.3.6.1. Concepts du langage

Le modèle de processus BPEL4WS forme une couche au-dessus du WSDL (garde les balises <type>, <message>, <portType>, <binding>, <port> et <service>). Il définit la coordination des interactions entre l'instance de processus et ses partenaires. Les processus dans BPEL exportent et importent les fonctionnalités en utilisant des interfaces de services web uniquement.
BPEL4WS permet de modéliser deux types de processus métier :

1. Le processus abstrait : Il correspond à une modélisation ou en terme plus approprié la description d'un protocole métier. Un protocole métier est décrit via les messages échangés et visibles entre les différents participants au protocole sans révéler leur comportement intrinsèque (spécifie les échanges de messages entre les différentes parties, sans spécifier le comportement interne de chacun d'eux) ;
2. Le processus exécutable : Il correspond à une modélisation comportementale d'un participant aux interactions du processus (spécifie l'ordre d'exécution des activités constituant le processus, des partenaires impliqués dans le processus, des messages échangés entre ces partenaires, et le traitement de fautes et d'exceptions spécifiant le comportement dans les cas d'erreurs ou d'exceptions).

2.3.6.1.1 La balise <process>

C'est l'élément racine (au sens XML) du fichier BPEL4WS. C'est à l'intérieur de cette balise que se retrouvera la description complète du processus.

2.3.6.1.2 Les liens de partenariat

Un partenaire correspond au service avec lequel le processus échange des informations. Le lien de partenariat représente la relation de conversation entre deux processus partenaires. La relation entre deux processus partenaires est une relation paire à paire (peer-to-peer). Le partenaire est en même temps le consommateur d'un service que le processus produit, et le producteur d'un service que le processus consomme.

Le lien de partenariat (partner link) définit le rôle que joue chacun des deux partenaires dans une conversation. Chaque lien de partenariat a un type (partnerLinkType).

2.3.6.1.3 Les activités

Le processus dans BPEL4WS est constitué d'activités, liés par un flot de contrôle. Ces activités peuvent être *basiques* ou *structurées*.
Les activités basiques sont : invoke, receive, reply, wait, assign, throw, terminate, empty ;

Activité basique	Description
Invoke	permet d'appeler le port d'un service Web partenaire
Receive	pour attendre un message d'une source externe
Reply	pour répondre à une source externe
Wait	pour attendre un certain temps
Assign	copie des données d'une place à l'autre
throw	lance une erreur d'exécution (Exception)
terminate	termine l'instance du service web
empty	qui ne fait rien (utile pour la synchronisation des activités concurrentes).

-Tableau 6 - Les activités basiques de BPEL4WS

Les activités structurées sont composées d'autres activités basiques et structurées. Les types d'activités structurées sont : sequence, switch, while, pick, flow, scope et compensate.

Activité structurée	Description
sequence	pour définir un ordre d'exécution
switch	pour l'acheminement conditionnel
while	pour les boucles
Pick	bloque le processus métier jusqu'à ce qu'un événement spécifique se produise
Flow	pour l'acheminement parallèle
scope	pour regrouper les activités afin qu'elles soient traitées par le même gestionnaire d'erreur
compensate	pour invoquer les activités de compensation par le gestionnaire d'erreur, pour défaire l'exécution déjà complétée d'un regroupement d'activité.

-Tableau 7 - Les activités structurées de BPEL4WS

2.3.6.1.4 Les données

Le processus dans BPEL4WS a un état, cet état est maintenu par des *variables* contenant des données.

Ces données sont combinées afin de contrôler le comportement du processus pour cela nous utilisons les "expressions". Les expressions permettent d'ajouter des conditions de transition ou de jointure au flot de contrôle.

L'affectation (assignment) permet de mettre à jour l'état du processus, en copiant les données d'une variable à une autre, ou en introduisant de nouvelles données en utilisant les expressions.

Dans BPEL4WS il n'y a pas de flot de données, BPEL4WS se sert des variables pour passer une donnée d'une activité à une autre, à l'aide de l'affectation.

2.3.6.2 -L'organisation du fichier :

Tous les éléments cités précédemment sont combinés dans un fichier qui contient la définition du processus (Figure 18)

1. Attributs supérieurs : cette section correspond à la définition des attributs de description du fichier comme le nom et les espaces de noms à utiliser pour le processus.
2. PartnerLinks (lien de partenariat) **:** cette section correspond à la description des tous les partenaires qui participent dans le processus
3. Variables **:** cette section correspond aux espaces de définition de variables utilisés dans le processus.
4. Orchestration **:** dans cette section on trouve la définition du processus en termes de différents types d'activités qui offre BPEL4WS

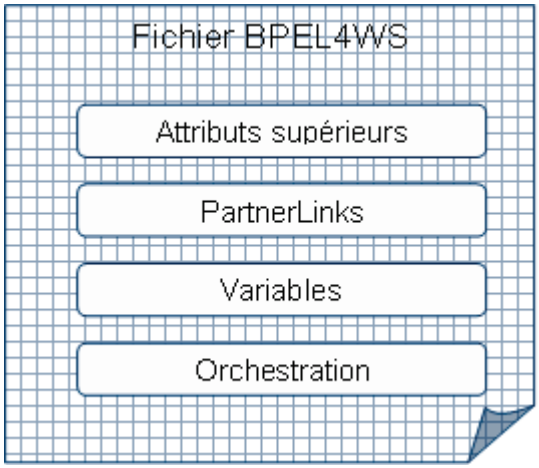


Figure 18. Structure d'un fichier BPEL4WS

Le fichier suivant présente un exemple minimaliste de code BPEL4WS déclarant deux partenaires ; l'interface du composite et un service Web de météorologie, spécifiant deux variables et déclarant une activité composite de type séquence composée de trois sous activités simples (receive, invoke et reply) dont le rôle est de recevoir le message du client de la composition, de le transmettre au service météo, puis de renvoyer la réponse au client.

```
<process >
  name="processName"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"

!-------------------------------------------------------------------!
  <partnerLinks >
    < partnerLink myRole =" proxy " name =" proxyPLT " partnerLinkType ="     proxyPLT "/>
```

```
    < partnerLink name =" Forecast " partnerLinkType =" ForecastLinkType "/>
  </ partnerLinks >
!-------------------------------------------------------------------------!
  <variables >
    <variable messageType =" proxyRequest " name =" proxyRequest "/>
    <variable messageType =" proxyResponse " name =" proxyResponse "/>
  </ variables >
!-------------------------------------------------------------------------!

  <sequence >
                                !-------------------------- ----------------------------!

    <receive createInstance =" yes" operation =" proxy " partnerLink ="    proxyPLT "
      portType =" ProxyService " variable =" proxyRequest "/>
                                !-------------------------- ----------------------------!
    <invoke inputVariable =" proxyRequest " operation =" invokeWF "
      outputVariable =" proxyResponse " partnerLink =" Forecast "
      portType =" invokeWFPT "/>
                                !-------------------------- ----------------------------!
    <reply operation =" proxy " partnerLink =" proxyPLT " portType =" ProxyService "
      variable =" proxyResponse "/>
  </ sequence >
!-------------------------------------------------------------------------!

</ process >
```

2.3.6.3 Systèmes exécutables basés sur BPEL4WS

BPEL4WS est largement utilisé. Plusieurs implémentations industrielles de ce langage existent actuellement.

- La première implémentation était proposée par la société "Collaxa Inc.", qui a construit un serveur d'orchestration de services web basé sur BPEL4WS.

- Le 29 juin 2004, Oracle a racheté Collaxa, afin de l'intégrer dans son serveur d'applications "Oracle Application Server 10g" [40], ce qui en a fait un produit maîtrisant l'orchestration de processus conformément à BPEL4WS (plus précisément le module Oracle BPEL Process Manager) [41].

- Le laboratoire alphaWorks d'IBM a produit BPWS4J (Business Process Execution Language for Web Services Java Run Time), disponible sur

leur site web alphaWorks. BPWS4J permet de créer et d'exécuter des processus métiers conformément à la spécification BPEL4WS [42].

- IBM a remplacé dans son logiciel WebSphere, la spécification WSFL, par cette nouvelle spécification BPEL4WS [43].

- Microsoft a également introduit cette nouvelle spécification dans son produit BizTalk Server 2004, afin qu'il puisse importer et exporter des processus métier écrits en BPEL4WS [44].
- La société Momentum[2] a également choisi BPEL4WS parmi les standards d'orchestration de services web, pour son produit d'automatisation de processus ChoreoServer [45] for .NET, conçu spécifiquement pour les vendeurs de logiciels indépendants et la redistribution commerciale.

- l'application BSOA Orchestra (Bull Service Oriented Architecture) [46], qui est actuellement dans sa version 2.3, est une plate-forme qui contient deux applications : La première est l'application Zenflow [47] qui permet de construire graphiquement, de modifier et de visualiser des processus BPEL4WS, avec les différents types d'activités qui présente la spécification. elle fournit également des interfaces pour faire la définition des variables, partenaires et lier avec des fichiers WSDL, aussi. La deuxième application est BSOA Orchestra Console, qui fonctionne sur le serveur d'application Jonas. BSOA Orchestra Console permet de suivre le fonctionnement du processus au moment de l'exécution (monitoring), déployer et exécuter des processus BPEL, de faire l'analyse de l'information dans l'état final d'un processus qui a déjà fini son exécution.
- BPEL Designer [48], qui est un plug-in Eclipse, qui offre une interface ergonomique simple à prendre en main (glisser - déposer) et permettant aux développeurs d'intégrer visuellement des services Web multiples

[2] La société "Momentum SI" http://www.momentumsi.com/

dans un processus BPEL, il offre une palette avec toutes les opérations possibles du langage BPEL4WS (Figure 19).

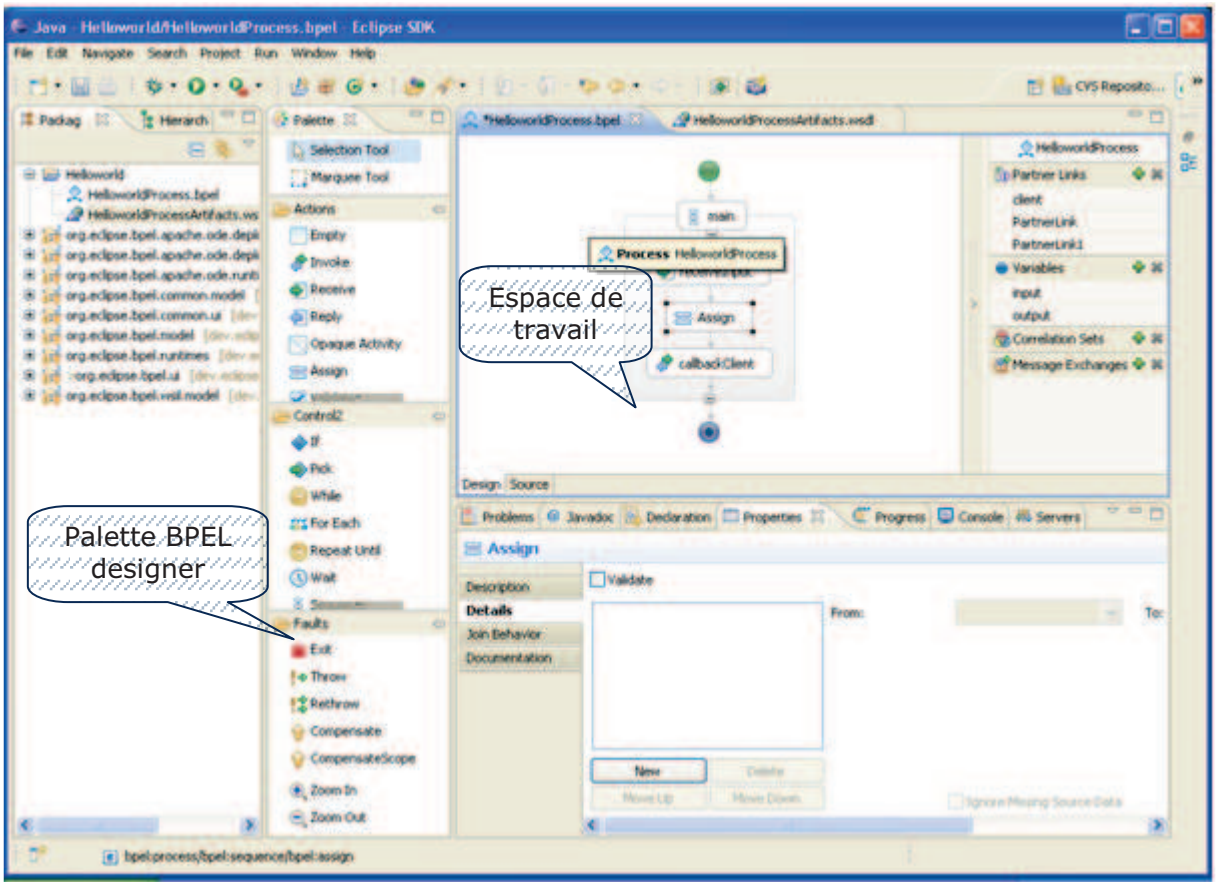


Figure 19. Le plugin Eclipse BPEL Designer

2.4 Rapport entre les langages de programmation et les fichiers WSDL :

Les langages ne gérant que la composition de services web, comme XLANG, WSFL, WSCL, WSCI et BPEL4WS forment souvent une surcouche aux fichiers WSDL. Il existe deux façons de réaliser cette surcouche :

- En étendant le fichier WSDL en lui ajoutant les informations de coordination, comme dans le cas du langage XLANG ;
- En séparant les informations additionnelles dans un fichier à part, collaborant avec le fichier WSDL, c'est le cas par exemple du langage BPEL4WS. ils proposent de créer des modèles de composition récursifs. Une composition

de services web écrite en WSFL ou en BPEL4WS peut être considérée comme un nouveau service web que l'on peut utiliser dans une nouvelle composition. Cette composition récursive sert également à faire, de manière indirecte, de la composition hiérarchique.

2.5 Evaluation

Nous avons synthétisé les caractéristiques des langages de composition vus précédemment dans le tableau qui suit :

Langage	Editeur	Orchestration	Chorégraphie	Types de participants	Nombre de participants
XLANG	Microsoft	Oui	Non	Services web	plusieurs
WSFL	IBM	Oui	Non	Services web	plusieurs
BMPL	BMPI	Oui	Non	Toute sorte de participants	plusieurs
WSCI	Sun, SAP, BEA, INTALIO	Non	Oui	Services web	plusieurs
WSCL	HP	Non	Oui	Services web	Deux
BPEL4WS	IBM, Microsoft, BEA	Oui	Oui	Services web	Plusieurs

-Tableau 8 – Tableau Récapitulatif des langages de composition

Le principe de certains langages de composition de service Web, comme XLANG et WSFL, est de construire un processus centralisé, se chargeant de coordonner les services Web. Ces langages font donc de l'orchestration.

Le langage BPML suit la même politique, mais il gère la coordination entre toutes sortes de participants et non pas des services web uniquement (ces

participants peuvent aussi être des applications, des utilisateurs humains, des partenaires commerciaux…etc). BPML utilise également un processus centralisé pour coordonner les participants au système.

Dans des langages comme WSCI, chaque service web connaît son comportement, et la réaction qu'il aura suite à chaque opération et à chaque message arrivant. Ce langage offre en effet une description du comportement du service web, formant une couche au dessus du fichier WSDL, décrivant les opérations et les messages de ce service web. La collaboration entre services web se passe d'une manière décentralisée. Ces langages font donc de la chorégraphie.

D'autres langages, comme WSCL, ne s'occupent que de la description de "conversations" entre les paires de services web. Autrement dit, il ne se charge que de décrire les interactions entre chaque couple de services web, sans spécifier comment est créé le contenu des messages échangés. Cette description de conversation prend une forme de processus, mais son exécution n'est pas centralisée. La collaboration des deux services web entre donc, dans ce cas-là, dans le domaine de la chorégraphie.

Le langage BPEL4WS quant à lui permet de décrire les deux types de collaboration :

1. La coordination centralisée, lorsqu'il s'agit de décrire l'enchaînement des appels d'opérations de services web (le "processus exécutable" dans le langage BPEL4WS). Ce qui correspond à faire de l'orchestration ;
2. Et la collaboration décentralisée, lorsqu'il s'agit de décrire le contrat d'interactions entre deux services web, ou ce qu'on a appelé "la conversation" (le "processus abstrait" dans BPEL4WS). Ce qui correspond à faire de la chorégraphie.

Après l'étude que nous avons faite sur les langages de composition, nous avons pu ressortir les défauts suivants :

1- La présence de beaucoup de concepts spécialisés ; chaque langage possède ses propres concepts, même en utilisant les éditeurs graphiques, l'utilisateur

doit posséder le bagage technique nécessaire pour pouvoir se débrouiller (voir Annexe A).

2- La plupart de ses langages utilisent des formes très restrictives ; limitées seulement aux services web. Elles ne sont donc pas généralisées pour toute sorte de participants (à l'exception du langage BPML).

3- Les langages de composition vus précédemment ressemblent fortement aux langages de programmation, ils sont donc des langages de bas niveau qui utilisent des structures à sémantique prédéfinies (les activités).

2.6 D'autres techniques de composition

2.6.1 SCA - Service Component Architecture

SCA est un modèle d'architecture à composants qui visent à simplifier la création des applications et systèmes par composition de services. Ses spécifications ont été écrites en fin 2005 par le groupe Open SOA (IBM, Oracle, SAP, Iona, Xcalia, BEA..), la version 1.0 été officiellement publiée en Mars 2007, et sont en cours de normalisation par l'OASIS [49].

La genèse de SCA réside dans le fait que les Services Web ne sont pas structurellement suffisantes, il faut un modèle pour organiser l'architecture globale des applications. Les Services Web n'assurent en fait que les communications entre les entités. SCA accepte un grand nombre de langages de programmation tels que Java, PHP, C++, Cobol, BPEL, XSLT, SQL et XQuery, C#, chacun doté de son propre système de spécifications.

SCA propose un modèle de programmation pour la construction d'applications à base de composants suivant le paradigme SOA. Ce modèle se base notamment sur l'idée qu'un service de niveau N se construit par assemblage (agrégation ou orchestration) de services de niveau N-1 ou N. A ce titre, SCA fournit deux niveaux de modèle :

- Un modèle d'implémentation : Construire des composants qui fournissent et consomment des services ;
- Un modèle d'assemblage : Construire une application métier à forte valeur ajoutée en liant entre eux un ensemble de composants.

SCA permet de décrire des services et leur assemblage indépendamment de toutes considérations techniques d'implémentation.

2.6.1.1 Le modèle d'implémentation

L'élément de base de SCA est le composant qui constitue l'unité élémentaire de construction Un composant est une instance configurée d'implémentation où une implémentation est un code source fournissant des fonctionnalités.

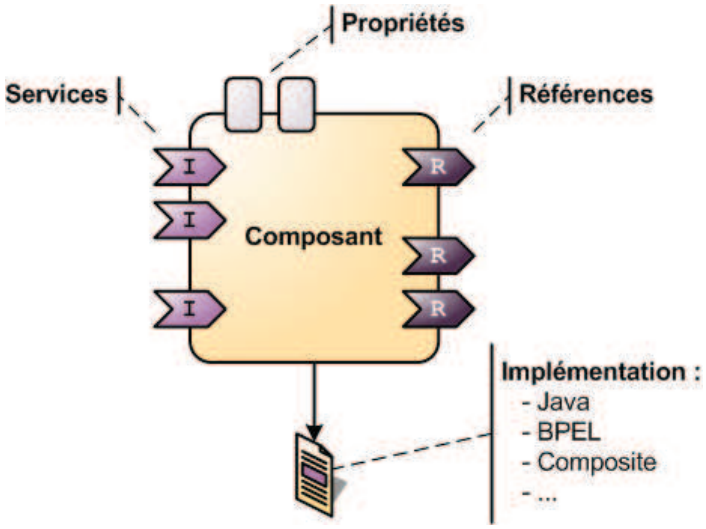


Figure 20. Modèle d'implémentation SCA

Ces fonctionnalités sont exposées en tant que services en vue de leur utilisation par d'autre composant. Les services sont décrits au travers d'interfaces qui constituent le contrat de service. Ces contrats sont implémentés par le composant.

L'implémentation peut s'appuyer sur des services fournis par d'autres composants dont elle dépend. Ces dépendances sont appelées références. Elles sont associées à des services qui peuvent être soit exposés par d'autre composants SCA soit exposé par des systèmes tiers (Web services, connecteurs JCA, …).

L'implémentation peut être paramétrable au travers de propriétés qui influencent le comportement d'une fonctionnalité. C'est le composant qui configure l'implémentation en fournissant des valeurs à ses propriétés et en liant les références aux services fournis par d'autres composants. Le système de références associé à celui des interfaces permet de réaliser un couplage lâche (ou faible) entre les composants : Un composant consommateur de services ne connaît des composants fournisseurs de services sur lesquelles il s'appuie que les interfaces (contrat de service) des services qu'il consomme.

2.6.1.2 Le modèle d'assemblage

Le deuxième élément définit par SCA est le composé qui est un assemblage de composants, services, références, propriétés et des liens qui existent entre ces éléments.

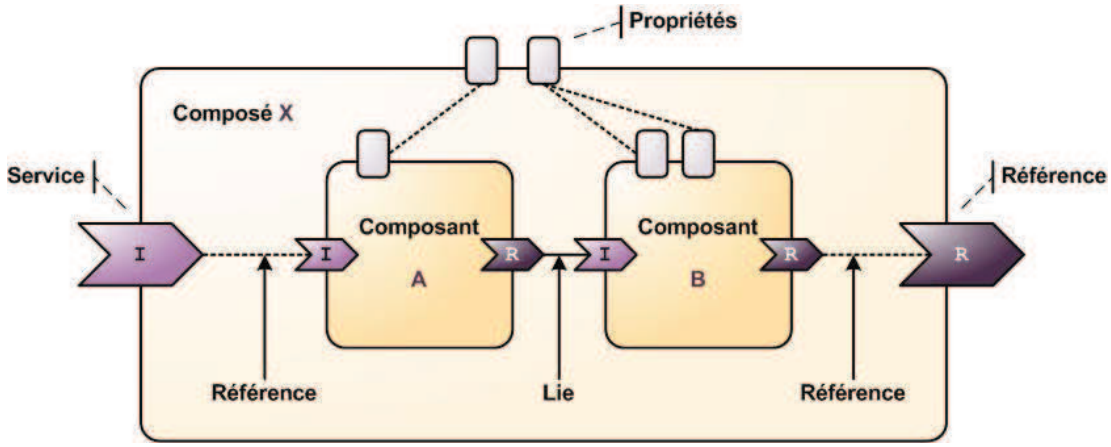


Figure 21. Modèle d'assemblage SCA

Un composé n'est donc rien d'autre qu'un composant de plus haut niveau que ceux qui le compose (Il fournit des services, dépends de références et a des propriétés). Un composé peut donc à son tour être référencé par d'autres composants et utilisé au sein d'autres composés.

L'utilisation première du composé peut être « détournée » pour regrouper un ensemble d'éléments non nécessairement liés mais qui constitue un ensemble fonctionnel cohérent.

Au plus haut niveau, les composés sont déployés dans des domaines SCA qui regroupe l'ensemble des services pour un système fonctionnel.

2.6.1.3 Implémentations open source

Il existe plusieurs implémentations open source des spécifications SCA. Les principales sont présentées dans cette section.

1. Le projet Apache Tuscany est une implémentation open source (licence Apache 2.0) des spécifications SCA. Tuscany permet la définition de nouvelles extensions pour de nouveaux types d'interface, d'implémentation et de méthodes de liaison, et repose sur le sérieux reconnu de la fondation Apache (déjà à l'origine de Tomcat, du serveur web Apache, Axis, Ant, ...).

2. Le projet collaboratif SCOrWare a pour ambition de fournir une implémentation en logiciels libre des spécifications SCA. Celle-ci comprend le runtime Frascati et l'outillage d'aide à la conception et au développement d'applications SCA. La plate-forme d'exécution Frascati est construite au dessus du modèle de composants Fractal du consortium OW2[3]. Elle offre des fonctionnalités avancées telles que la reconfiguration dynamique des assemblages SCA.

3. Le projet Fabric3 est une autre implémentation sous licence Apache. Il offre la possibilité de déployer et gérer des services dans des environnements distribués. Fabric3 est compatible avec de nombreux outils de développement et technologies populaires (JEE, Spring,...). Son architecture modulaire permet de s'ajuster aux technologies requises sur un projet. Il peut être utilisé pour construire, déployer et gérer des applications SCA au sein de divers middleware (serveur d'application JEE, conteneurs de Servlet, conteneurs OSGi).

[3] **OW2** est une association (loi 1901) internationale à but non lucratif dédiée au développement d'intergiciels libres de qualité industrielle. Elle regroupe des entreprises et des organismes de recherche de premier plan tels que l'INRIA, Bull, France Télécom, Thales Group ou RedHat

4. Le projet Newton est un framework OSGi distribué qui permet l'instanciation et la gestion d'applications SCA au sein d'environnements d'entreprise. Newton est capable de déployer et de gérer des assemblages de composants distribués par des ajustements dynamiques en fonction des erreurs et des changements de topologie des réseaux.

5. Eclipse SCA : Le projet SCA Tools, SCA Tools est une suite d'outils permettant de faciliter le développement d'applications SCA. C'est un projet faisant partie du projet SOA Tools Plateform (STP) de la fondation Eclipse. Ces outils sont issus du projet collaboratif SCOrWare.

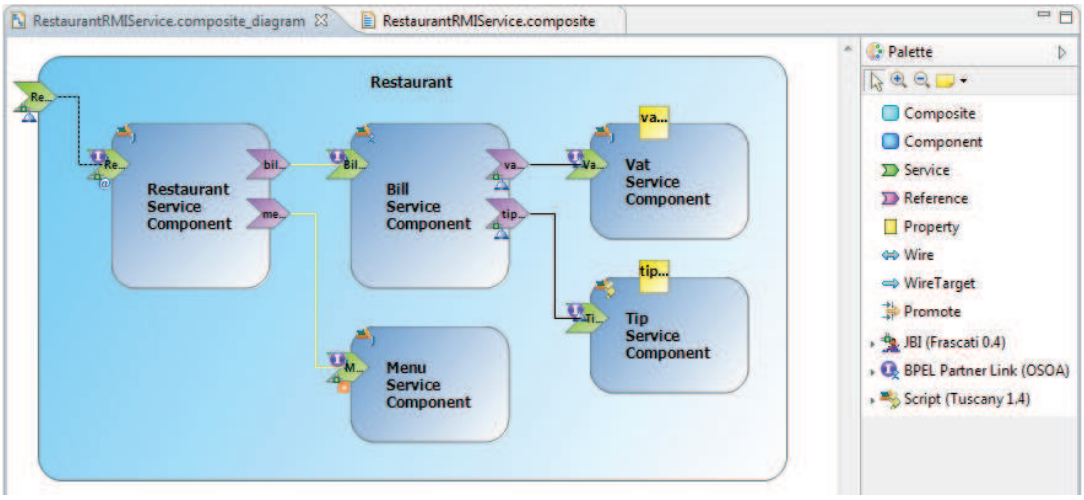


Figure 22. Diagramme SCA dans l'éditeur graphique de SCA Tool

2.6.1.4 Implémentations industrielles

Il existe aussi des implémentations industrielle, telles que :

1. IBM WebSphere Feature Pack for SOA qui permet d'ajouter la prise en charge des applications SCA au sein du serveur d'application IBM WebSphere.
2. Rogue Wave HydraSCA qui permet aux développeurs de construire des composants individuels puis de les lier au sein de « business process » agiles et de les exécuter de manière concurrente.

3. Oracle SALT (Service Architecture Leveraging Tuxedo) qui permet d'adapter le moniteur transactionnel Oracle Tuxedo au SOA. La dernière version (10gR3) de l'outil SALT supporte les spécifications SCA. Cet outil fait partie de la famille Oracle Fusion Middleware.

2.7 Vision académique de la composition de services web :

Plusieurs recherches sur la composition de services web ont été menées, afin de pallier aux insuffisances des langages de compositions vus précédemment :

- Eflow, est une plateforme pour la spécification, la création et la gestion de services composites en utilisant les méthodes de génération de Workflows statiques mais représentés en interne sous la forme d'un graphe qui peut être modifié dynamiquement lors de l'exécution. Ce graphe contient, en plus des noeuds représentant les services, des noeuds de décision et d'évènements qui permettent une plus grande robustesse du système : en cas de non disponibilité d'un service Web, celui-ci peut être automatiquement remplacé par un service Web équivalent.

- AO4BPEL (Aspect Oriented for BPEL) [50] et [51], portent sur l'élaboration d'un mécanisme de composition faisant intervenir des aspects rédigés sur la base du langage BPEL4WS. AO4BPEL introduit le concept de programmation par aspects dans les processus métier afin de pallier aux deux lacunes du langage BPEL4WS : d'une part les compositions manquent de modularité, et d'autre part de dynamicité des adaptations (les compositions demeurent figées du fait que le BPEL4WS spécifie des informations statiques). Les auteurs de ces travaux ont ainsi réalisé un modèle et une implémentation de langage d'aspect (AO4BPEL) basé sur le langage BPEL4WS. Un moteur BPEL4WS spécifique a également été conçu pour la gestion des aspects, ainsi que pour l'intégration de services middleware non fonctionnels usuels (sécurité, garantie de livraison, transaction). Le langage AO4BPEL, comme tout langage orienté aspect possède un modèle de points de jonction, un

langage de coupe et un langage d'action. Il a cependant la particularité d'être entièrement rédigé en XML. Selon la modélisation AO4BPEL, chaque activité est un point de jonction potentiel et il existe également des points de jonction liés aux événements d'envoi et de réception de messages SOAP. L'implémentation d'AO4BPEL (Figure 23) est effectuée par un moteur BPEL s'appuyant sur le moteur BPWS4J d'IBM.

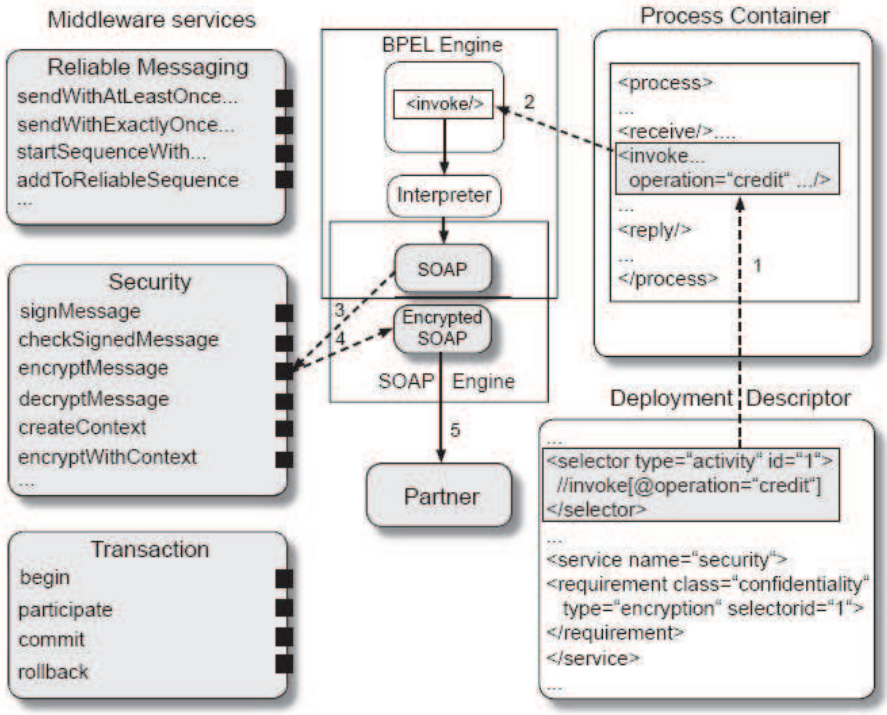


Figure 23. Architecture AO4BPEL

- YAWL(Yet Another Workflow Language), « Van der Aalst et al » ont définit vingt patterns pour évaluer les workflows, sur la base d'une analyse rigoureuse des systèmes de gestion de workflow, Ces patterns forment suivant leur vision, une mesure permettant d'évaluer les fonctionnalités des systèmes de gestion de workflows. Dans leur évaluation des langages de composition de services web, les auteurs ont utilisé également 6 patterns de communication présentés dans [52].

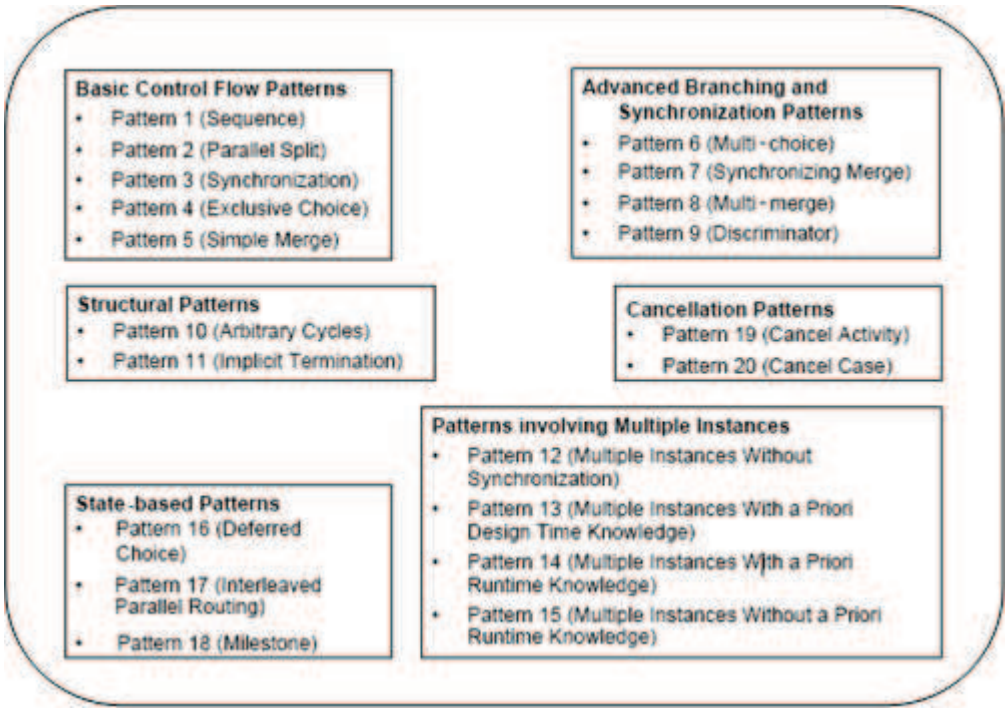


Figure 24. Les vingt patterns de Van der Alast

Ils ont constaté après que la représentation de certains modèles qui nécessitent des synchronisations complexes est très fastidieuse et ils ont proposé un nouveau langage de workflow appelé YAWL [53] qui se base sur les réseaux de Petri colorés et les 20 patterns définis. Ci après les symboles utilisés pour le langage YAWL

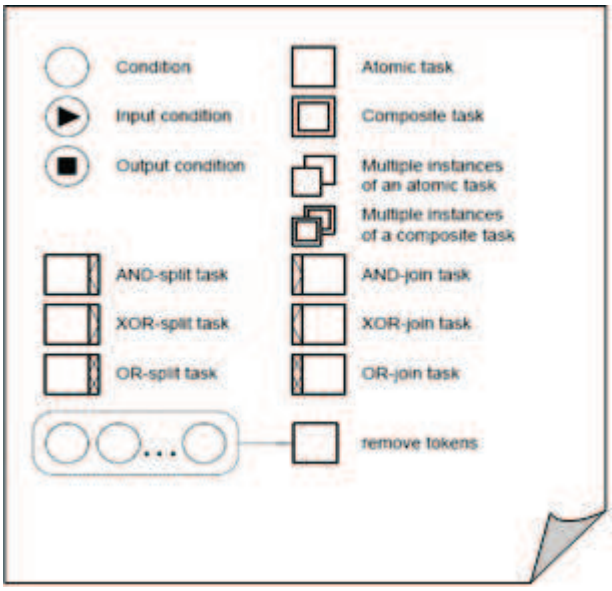


Figure 25. Les symboles de YAWL

La mise en œuvre de YAWL est « YAWL Editor » codé en Java. Cet outil est open source disponible en libre téléchargement[4] .

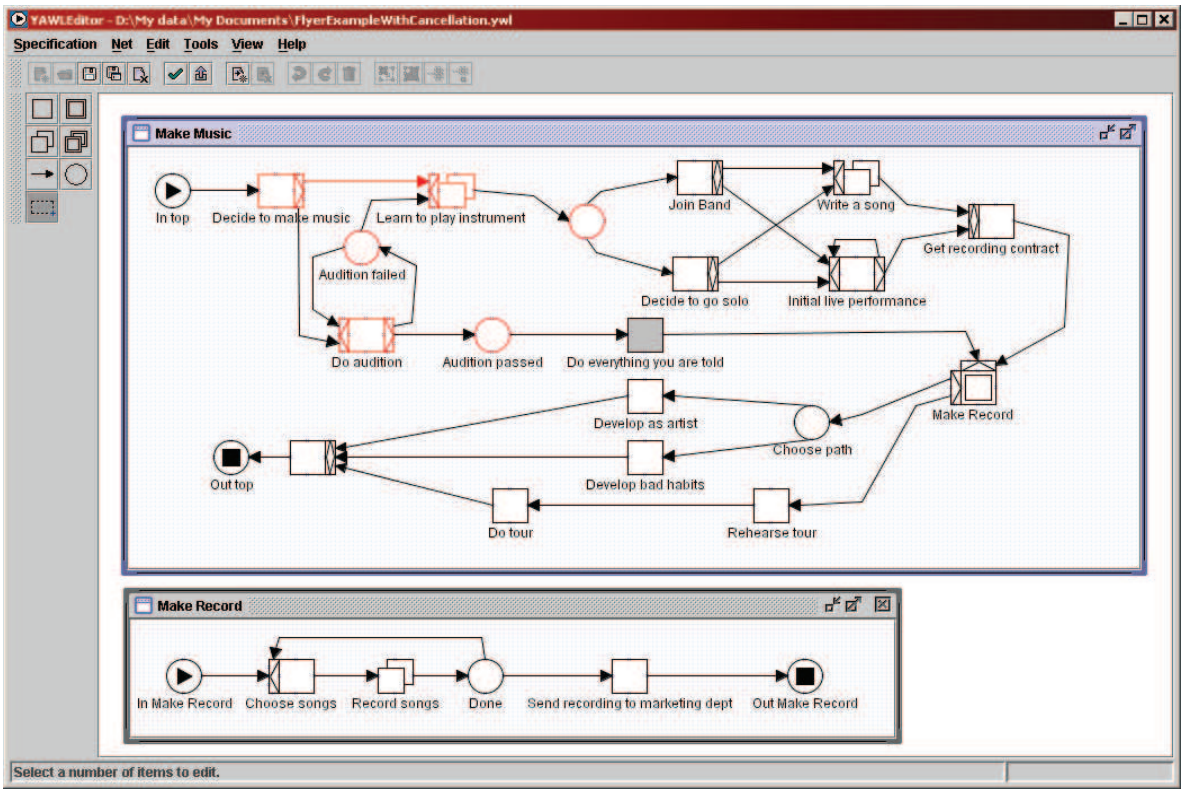


Figure 26. Interface graphique de YAWL Editor

- Les auteurs de [54] identifient deux solutions possibles pour composer dynamiquement des processus métier : remplacer un service Web dans un processus métier existant par un autre service ayant des fonctionnalités similaires, ou définir un nouveau workflow à partir des services Web disponibles. Les auteurs proposent d'utiliser les descriptions sémantiques des services Web pour pouvoir comparer les fonctionnalités en La solution proposée (représentée dans la figure 27) peut être résumée ainsi :

  1. Identifier les fonctionnalités requises
  2. Trouver les services adaptés grâce à leur description sémantique
  3. Créer ou modifier le workflow.
  4. Exécuter le workflow.
  5. Vérifier l'exécution du workflow.

[4] www_YAWL.jar :

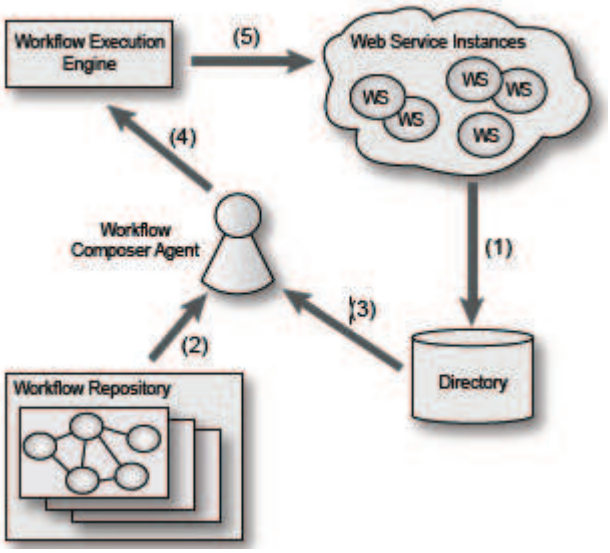


Figure 27. Solutions proposés dans [*54*]

- Approches formelles :
  - Preuve de théorème (Theorem proving). Waltinger [55] a élaboré une approche pour la composition de services par preuve de théorèmes. Cette approche est basée sur la déduction automatique et la synthèse de programmes. Les services disponibles et les requêtes utilisateur sont traduits dans un langage du premier ordre. Puis des preuves sont produites à partir d'un prouveur de théorèmes. La composition est obtenue à partir de preuves particulières.
  - Calcul situationnel (situation calculus) [56] les auteurs proposent d'adapter et d'étendre le langage Golog pour la construction automatique de services Web. Golog est un langage de programmation logique qui permet de faire du calcul situationnel (i.e., langage logique qui sert à représenter des changements ou évolutions en terme de situations, d'actions et d'objets). Le problème de la composition de services Web est abordé de la façon suivante : la requête de l'utilisateur et les contraintes des services sont représentées en terme de prédicats du premier ordre dans le langage de calcul situationnel. Les services sont transformés en actions (primitives ou complexes) dans le même langage. Puis, à l'aide de règles de déduction et de contraintes, des

modèles sont ainsi générés et sont instanciés à l'exécution à partir des préférences utilisateur. Ces recherches ont été étendues dans [57]: les prédicats nécessaires aux calculs situationnels sont tirés de DAML-S et représentés sous la forme de réseaux de Petri.

- Système multi-agent (SMA), Du fait de leur autonomie et leur hétérogénéité, les services Web peuvent être vus comme des agents. Ainsi les auteurs de [58] proposent le langage ASDL (Agent Service Description Language) qui a pour objectif de décrire le comportement externe des services Web. Une spécification ASDL décrit les messages compris par un service, ainsi que les protocoles d'interactions utilisés. Puis la composition est effectuée à l'aide du langage ASCL (Agent Service Composition Language) qui permet de décrire la logique avec laquelle un nouveau service est composé à partir de services existants. L'avantage de ces travaux est de mettre l'accent sur les interactions entre services au sein de la composition et d'adapter celle-ci en fonction des contraintes résultant des interactions. Cependant cette méthode n'est pas totalement dynamique. En effet, elle consiste à assembler et adapter au contexte d'exécution des patrons de composition définis a priori par le concepteur.
- Planification. Les services Web sont représentés par des opérateurs ayant des pré-conditions et produisant des effets. Puis un planificateur peut alors être utilisé pour produire un plan qui correspond à la composition des services Web. dans [59] les auteurs ont utilisé cette correspondance pour composer un ensemble de services Web de manière intuitive : les services Web sont choisis et composés uniquement à partir des types de données de leurs entrées et sorties.
- SMWM (Service Mediating Workflow Management) [60] ; Il s'agit d'une étude de l'université de Twente[5], qui tente d'améliorer l'association entre le workflow et des services Web en ajoutant un médiateur entre ces deux couches [61]. Le médiateur permet de dissocier une liaison directe entre le workflow et les services Web. Il est donc chargé d'établir dynamiquement l'association entre chaque activité et son service Web

[5] http://www.universiteittwente.nl/en

correspondant en exécution. Pour ce faire, il est basé sur deux informations :

1- Des contrats d'association qui spécifient la correspondance entre chaque activité et un ensemble de service qui sont capables de l'implémenter. Ces contrats sont établis lors de la phase de définition de modèle de procédé.
2- Des politiques de sélection qui permettent de choisir, à l'exécution, puis d'invoquer le service le plus approprié pour une activité, dans le cas où il y aurait plusieurs candidats possibles.

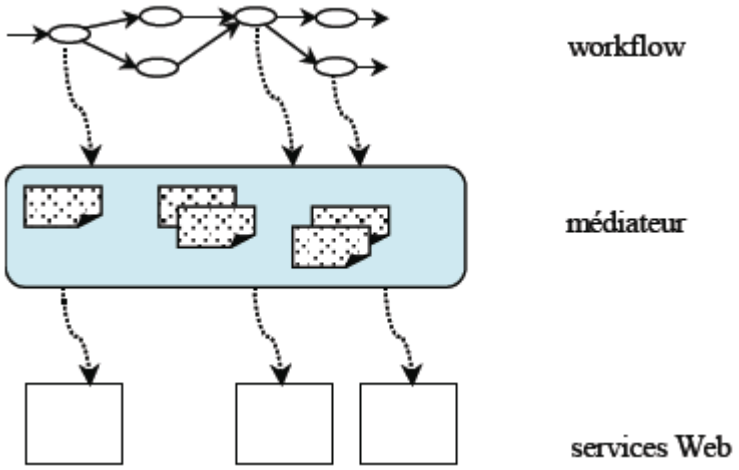


Figure 28. Médiateur dans SMWM

Grâce au médiateur, la définition des modèles de procédé ne contient plus d'information concernant des services utilisés. Ceci rend des *workflows* plus indépendant et plus flexible dans l'établissement de l'association avec ces services.

2.8 Conclusion

Dans ce chapitre nous avons présenté le problème de composition de services Web, en exposant les efforts qui ont été fait par l'industrie et le monde de recherches

La composition de services est principalement mise en œuvre selon deux techniques : l'orchestration et la chorégraphie.

Dans l'orchestration, un processus centralisé est défini pour coordonner les services web. Ceci permet de donner une vision globale de la coordination.

Dans la chorégraphie, la collaboration entre services web est décentralisée. Chaque service web gère lui seules ses interactions avec les autres services. Par conséquent, la chorégraphie ne fournit pas l'ensemble des interactions entre tous les services web. Nous ne pouvons donc pas avec cette technique obtenir une vision globale de la collaboration.

Plusieurs langages de composition ont vu le jour, certain ont adopté l'orchestration, d'autre la chorégraphie, mais dans les deux cas ils décrivent la collaboration entre services web sous forme de fichiers XML. Ces fichiers XML sont souvent très difficiles à lire et à manipuler. Il est même difficile de garantir la cohérence entre les différentes parties éparpillées sur les nombreux fichiers XML.

Des éditeurs graphiques ont été développés, afin de faciliter la tâche d'édition de fichiers de description de la composition. Cependant même avec les éditeurs graphiques le concepteur doit connaître les détails du langage de composition avec lequel il travaille puisque il doit intervenir avec des instructions de bas niveau (proche des langages de programmation)

Nous avons aussi constaté que la plupart de ces langages sont limités aux services Web et ne gèrent pas la composition entre toute sorte de participants

Il devient alors nécessaire et surtout intéressant de trouver une technique de composition qui puisse d'une part faciliter la composition et d'autre part assurer la composition entre les différents types de participants.

Pour cela nous proposons de remonter le raisonnement à un haut niveau d'abstraction, nous avons choisi l'architecture logicielle comme solution à la difficulté de composition. Nous verrons, dans le Chapitre III, les concepts de base de l'architecture logicielle nous présentons aussi le modèle de composants que nous avons opté pour résoudre cette problématique. Nous allons voir également, comment nous avons réussi à répondre à nos objectifs, en présentant notre environnement de développement. Nous expliquons cela avec des exemples dans le Chapitre IV.

CHAPITRE 3
L'ARCHITECTURE LOGICIELLE ET L'APPROCHE IASA

3.1- Introduction

Le logiciel et les techniques de développement de programmes informatiques ont dû faire face à des applications de plus en plus complexes. Pour maîtriser cette complexité, la notion d'architecture logicielle est apparue. Grâce à l'architecture logicielle, les applications d'aujourd'hui sont fabriquées à l'aide des méthodes permettant la construction d'applications exécutables par assemblage de composants logiciel. Dans ce chapitre nous allons présenter les concepts de bases de l'architecture logicielle ainsi que ses avantages. Nous nous familiarisons ensuite avec le vocabulaire de l'Approche Intégrée d'Architecture Logicielle (IASA) qui que nous avons choisi pour concrétiser nos objectifs.

3.2- L'architecture logicielle

L'architecture logicielle consiste à appliquer à la conception de logiciels des démarches similaires à celles de conception d'architecture de processeurs et d'ordinateurs. Elle se base sur des concepts similaires aux concepts d'architectures d'ordinateurs : les concepts de composants et de connecteurs. Les composants, peuvent provenir de sources variées et peuvent être réalisés dans des technologies diverses. Le raisonnement à divers niveaux d'abstraction[1] est une technique largement

[1] Abstraire la programmation peut être vu comme l'utilisation de mécanismes qui servent à réduire un programme d'un niveau de détail. Ces détails peuvent être liés à un paradigme de programmation utilisé et/ou une architecture matérielle. Il s'agit de se rapprocher du raisonnement humain plutôt que d'être proche d'un contexte d'exécution

utilisée en architecture logicielle pour isoler les aspects technologiques et réduire la complexité des logiciels. Par cette technique, l'architecture logicielle permet d'une part, une exploitation simple et efficace des diverses innovations technologiques en génie logiciel et d'autre part, une prise en charge de la construction de systèmes très complexes utilisant des grains de diverses tailles provenant de diverses sources [6]. L'architecture logicielle se base sur deux concepts fondamentaux : Le concept de composant et le concept de connecteur (Figure 29)

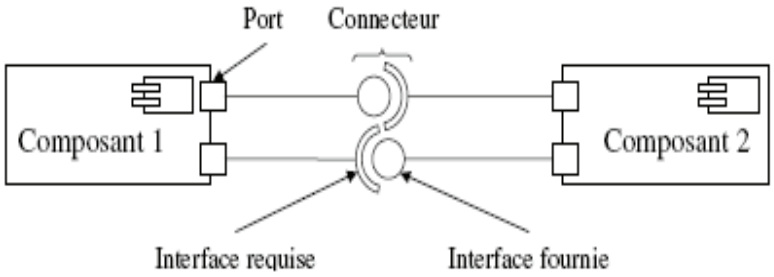


Figure 29. Concepts de base de l'architecture logicielle

Un composant logiciel est une entité responsable de la réalisation d'une (ou plusieurs) fonctionnalité(s) bien précise(s) dans une architecture à un certain niveau d'abstraction. C'est une entité qui fournit des fonctionnalités de calcul et de stockage. Il interagit avec les autres composants pour réaliser un ou plusieurs objectifs d'une architecture. Un composant peut être très simple, comme une fonction C, un objet d'une classe C++ ou complexe comme un serveur HTTP ou un SGBD.

Un composant est doté d'interfaces qui décrivent l'ensemble des services fournis et l'ensemble des besoins nécessaires à son fonctionnement et à la réalisation des services qu'il offre. Une interface est composée d'un certain nombre de points d'interaction appelés ports de communication dans WRIGHT [62] et C2 [63] ou « players » dans UNICON [64].

L'interaction entre composant est prise en charge par le concept de connecteur. Un connecteur lie une ou plusieurs interfaces représentant un service requis et une interface représentant un service fourni. Un connecteur possèdent deux vues ; Une vue abstraite dans laquelle sont spécifiées les interactions et une vue concrète représentant l'infrastructure permettant de transporter les éléments de l'interaction.

3.2.1 - Description des Architectures logicielles :

Les premiers travaux réalisés pour décrire les architectures logicielles ont aboutit à la naissance de langages de description d'architecture dit ADL. En accord avec la notion d'architecture logicielle, l'idée est de fournir une structure de haut niveau de l'application plutôt que l'implantation dans un code source spécifique [65]. Il faut préciser qu'il n'existe pas une définition unique des ADL (Architecture Description Language), mais en général on accepte qu'un langage de description d'architecture fournisse un modèle explicite de composants, de connecteurs et de leur configuration.

Depuis le début des années 90 un bon nombre d'ADLs ont vu le jour. En général, les divers ADL proposé sont orienté vers un domaine d'application particulier ou pour la résolution d'un problème bien spécifique à un niveau Architectural. Parmi les ADL les plus représentatifs correspondant à une première génération nous trouvons notamment :

- Darwin pour les systèmes d'envoi de messages distribués [66],
- Rapide pour la validation d'architecture par simulation [67],
- Wright pour la vérification formelle des propriétés d'une architecture logicielle [68].

3.3 - L'approche intégrée d'Architecture logicielle :

L'approche intégrée ou IASA (Integrated Approach to Software Architecture) est une approche définie dans le contexte d'une collaboration entre l'ESI[2] et le laboratoire

[2] ESI http://www.esi.dz/

LINA[3] de l'Université de Nantes. Cette approche permet une spécification très flexible et à un haut niveau d'abstraction d'architecture logicielle [6].

3.3.1 - Le modèle de composants de l'approche intégrée :

Le concept de composant est utilisé pour représenter n'importe quel élément rentrant dans la définition fonctionnelle d'une application. Autrement dit, toute fonctionnalité faisant partie de la logique d'une application est explicitement prise en charge par un composant. Les connecteurs sont exclus de cet ensemble, vu qu'ils sont neutres vis-à-vis des fonctionnalités d'une application.

Un composant est un type. La création d'instance est contrôlée par le type. C'est ainsi qu'il est possible de fixer le nombre total d'instances pouvant être crées. Nous pouvons distinguer deux types de composants : les composants primitifs et les composants composites. La structure interne d'un composant primitif est inaccessible. Par contre, celle d'un composite possède une organisation bien précise. Elle est composée de deux parties. Une première partie, appelée partie opérative, comprend les composants réalisant les fonctionnalités de l'architecture. La deuxième partie, appelée partie contrôle, contient des composants qui réalisent les opérations de contrôle global sur les autres composants et des composants qui supportent les aspects techniques de l'application.

L'instanciation d'un composant est réalisée dans le contexte du concept d'enveloppe. Une enveloppe permet d'isoler l'instance pure d'un composant de son environnement d'exploitation en fournissant à ce dernier les éléments nécessaires à l'exploitation de ses instances. L'enveloppe est l'endroit ou seront solutionnés les divers problèmes liés au déploiement de l'instance du composant et à la spécification de topologies très variées, notamment celle mettant en œuvre directement les points d'accès de port.

---

[3] Lina http://www.lina.univ-nantes.fr/

Dans la section qui suit, nous détaillons les éléments fondamentaux du modèle de composant de l'approche intégrée. Nous commencerons par les éléments modélisant la vue externe, à savoir le concept de point d'accès et le concept de ports. Nous présenterons ensuite le concept d'enveloppe et nous étudierons le modèle de la vue interne.

3.3.1.1 Modélisations de la vue externe :

3.3.1.1.1 Les points d'accès

Les points d'accès représentent dans notre modèle les concepts de base échangés entre deux ports de composants. Un point d'accès permet de localiser les diverses ressources fournies ou requises à travers un port et de renseigner sur les éléments intervenant dans la réalisation d'un comportement observable sur un port. Toute information, quelque soit sa nature (l'échange de données et le transfert du flux de contrôle) circule d'un composant à un autre à travers un point d'accès. Ainsi, un paramètre d'une méthode peut être associé à un point d'accès vu qu'il véhicule une information.

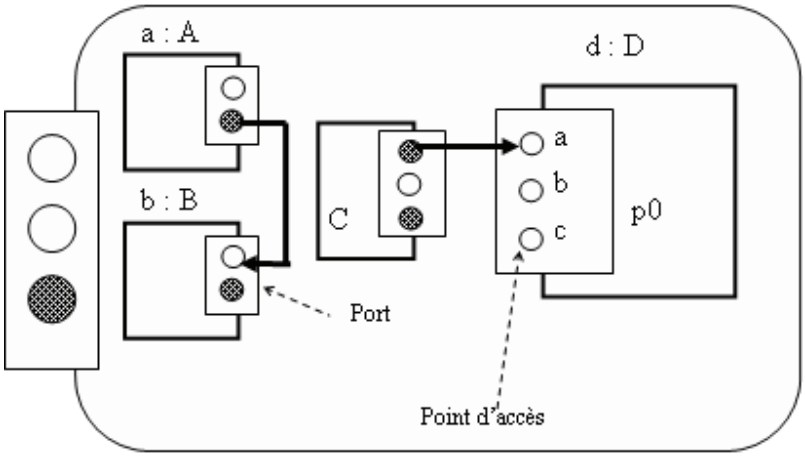


Figure 30. Un composant composite dans l'approche IASA

Les points d'accès sont tous représentés par le type de base IASA_AcessPoint. Le type AnyPoint possède un mode de communication bien précis, Il est doté de propriété générale aux points d'accès et d'opérations de base notamment celle destinées à des opérations réflexives. [62][69]. Un point d'accès est destiné soit au transfert de données (Data Oriented Access Point ou DOAP) ou émettre ou recevoir des actions (Action Oriented Access Point ou ACTOAP). Un ACTOAP indique la présence d'un service qui peut être initié à partir de ce point.

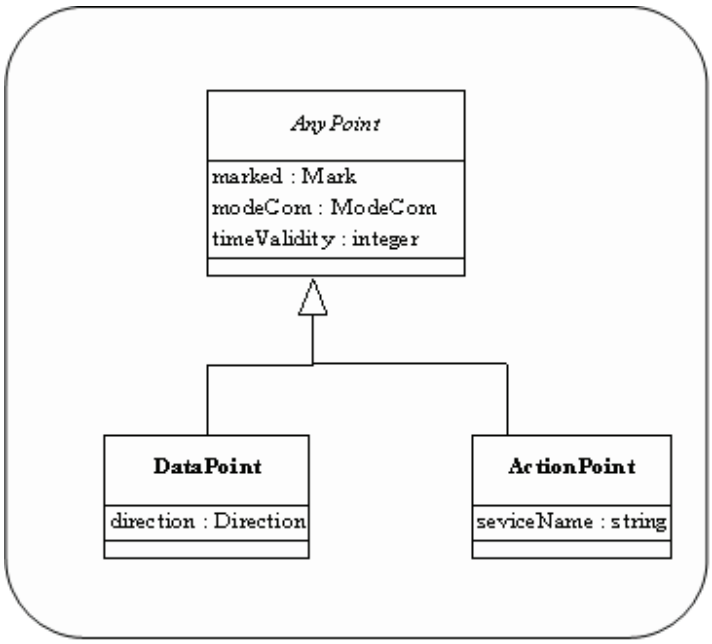


Figure 31. Diagramme de classes pour les points d'accès

Un point d'accès peut être marqué ou non marqué (marked, unmarked). Un point d'accès marqué est un point d'accès correctement connecté. Cette caractéristique est très utile dans le processus de validation d'une architecture et est exploitée dans un processus de conception du général vers le particulier avec validation progressive de l'architecture durant les différentes phases de conception, sans nécessité que les composants utilisés soient effectivement réalisés.

3.3.1.1.1.2 Les points d'accès aux données ou DOAP (Data Oriented Access Point) :

Un DOAP est utilisé pour spécifier un transfert explicite de données. L'architecte peut utiliser un ensemble de DOAP prédéfini ou définir son propre DOAP. Les DOAP prédéfinis englobent les types de données primitifs (entier, réel, caractère, booléen) ainsi que des types spécifiques à l'approche IASA, tels que les types de données renseignant sur l'état structurel d'un composant composite. Dans le contexte actuel où le langage Java est utilisé comme cible dans les diverses opérations de validation des modèles de l'approche intégrée, les types primitifs associés aux DOAP correspondent aux types primitifs Java. A titre d'exemple, IntDOAP, ByteDOAP, CharDOAP et BooleanDOAP sont successivement associés aux types primitifs int, byte, char et boolean. Les points d'accès SubCmpSetDOAP, ConSetDOAP, DConSetDOAP, OpPartPortSetDOAP, et CtrlPartPortSetDOAP sont associés à des types de données renseignant sur l'état structurel d'un composite (i.e. listes de composants, de connecteurs, de ports internes etc..).

La définition de nouveau DOAP spécifique doit suivre un style de nommage et une méthodologie de définition bien précise. Le style de nommage apparaît clairement dans les exemples précédents. Le nom du nouveau type commence par le nom du type associé au point d'accès et se termine par le terme DOAP.

Un DOAP est doté d'attributs indiquant le sens des opérations de transfert de données (Figure 31). Trois valeurs sont possibles pour spécifier le sens des données : in, out et inout. L'attribut in indique la nécessité de pourvoir le DOAP d'une donnée. L'attribut out indique que le DOAP est une source de données. L'attribut inout spécifie que la donnée peut être transférée dans les deux directions. Comme ceci est le cas dans plusieurs approches [70], les variables globales (mémoire partagée) sont considérées comme des implémentations possibles du concept de DOAP possédant l'attribut inout. Dans notre cas, un DOAP inout spécifie que le concept associé possède une existence réelle et n'est pas réduit à une référence. Ainsi si deux DOAP inout de deux composant sont connectés, il y'aura alors deux copies de la donnée associée aux

DOAP, chacune localisée dans son composant. Si deux DOAP inout sont connectés, les deux données associées aux points d'accès auront toujours les mêmes valeurs.

La connexion de DOAP suit les règles suivantes : Un DOAPin (respectivement DOAPout) ne peut se connecter qu'à un DOAPout (respectivement DOAPin) ou DOAPinout. Un DOAPinout est connectable à un DOAPin, DOAPout et DOAPinout. Lorsqu'un DOAP est correctement connecté, il est alors marqué (positionnement de l'attribut de marquage du DOAP à la valeur marked).

Les DOAP peuvent être explicitement spécifiés avec des valeurs d'initialisation (Figure 32). Un point DOAPin initialisé est un point marqué qui peut être non connecté. Les DOAPin avec initialiseurs sont utilisés pour spécifier des valeurs par défaut pour des DOAP qui, éventuellement, ne serait pas connectés. Si le DOAPin avec initialiseur est connecté la valeur de l'initialiseur n'aura aucun sens. Un DOAPout avec initialiseur est un point représentant une valeur constante qui ne peut pas être altérée durant l'exécution.

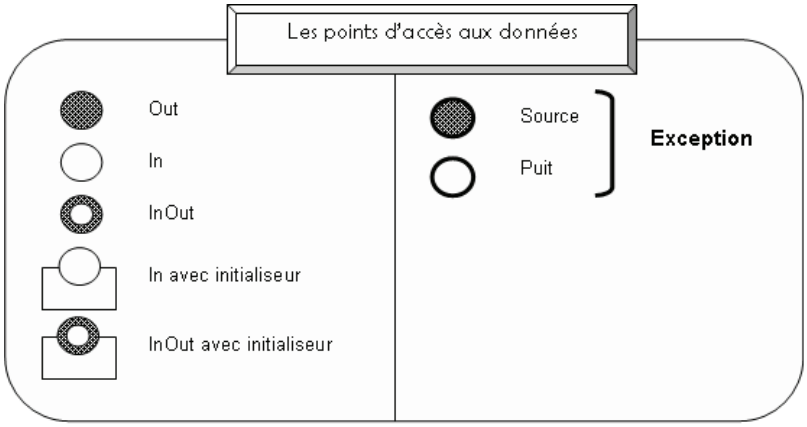


Figure 32. Représentation graphique des points d'accès

3.3.1.1.1.2 Les points d'accès de services ou ACTOAP (Action Oriented Access Point)

Un ACTOAP indique qu'un service peut être initié à partir de ce point. Dans la version actuelle d'IASA, un ACTOAP correspond uniquement à un seul service et un service permet de réaliser plusieurs actions. Généralement le nombre d'actions que

peut prendre en charge un service, est assez restreint et concerne un aspect bien précis d'un domaine d'application. C'est cette idée qui est à la base de la définition de la notion de contexte d'action au niveau du langage 3ADL.

Vis-à-vis d'un service, un ACTOAP ne peut être que fournisseur ou client. Un ACTOAP à travers lequel un service est appelé un ServerACTOAP (ou ACTOAPS). Un ACTOAP qui spécifie un besoin de service est un ClientACTOP (ou ACTOAPC).

3.3.1.1.2 Les ports :

Un port est un regroupement de points d'accès, étroitement associés dans le contexte de la réalisation d'un objectif commun. Tous les ports sont représentés par le type AnyPort. Un port possède un attribut renseignant sur le nom de l'instance. Le port représente un espace de nom pour les points d'accès. Chaque point d'accès est identifié de manière unique dans le contexte d'un port. Le composant représente un espace de nom pour un port. Ce dernier est identifié de manière unique dans un composant.

Les ports modélisent la vue externe d'un composant. C'est seulement à travers les ports qu'un composant est manipulable. Les ports divulguent les ressources (services et données) requises et fournies d'un composant ainsi que les aspects comportementaux du composant, observables sur ces ports. Les comportements sont décrits dans le langage d'action 3ADL [71] qui est une des composantes essentielles de l'approche IASA.

Un port est une entité à part, pouvant être ajouté ou supprimé à un composant. Il peut en outre être modifié par l'ajout ou la suppression de point d'accès, notamment les points d'accès de données. Cette situation est souvent rencontrée avec les composants représentant des opérateurs et les composants d'initialisation.

3.3.1.1.3 L'enveloppe

L'instanciation de tout composant se fait à travers le concept d'enveloppe. Le concept d'enveloppe a été introduit pour permettre d'atteindre les objectifs suivants [6] :

- L'isolation totale du type de composant du monde externe.
- Offrir le support nécessaire à la réalisation de topologies qu'il n'est pas possible de réaliser dans le contexte de ports typés par les interfaces, comme ceci est le cas dans les divers ADL et UML.
- Offrir le support nécessaire au déploiement des instances d'un composant. Selon son environnement d'existence, une instance sera enveloppée par l'enveloppe adéquate (Une instance s'habille adéquatement pour une situation particulière).
- Offrir le support nécessaire à la réalisation des opérations de validation.
- La transformation de n'importe quel composant (COTS[4], anciennes applications) provenant de n'importe quelle source, en un composant prêt à être exploité dans les opérations d'assemblage selon le modèle de composant de l'approche intégrée.

[4] Composant sur étagère : Component On The Shelf

3.3.1.2 Organisation de la vue interne

La vue interne est organisée en deux parties (Figure 34). Une partie opérative et une partie contrôle.

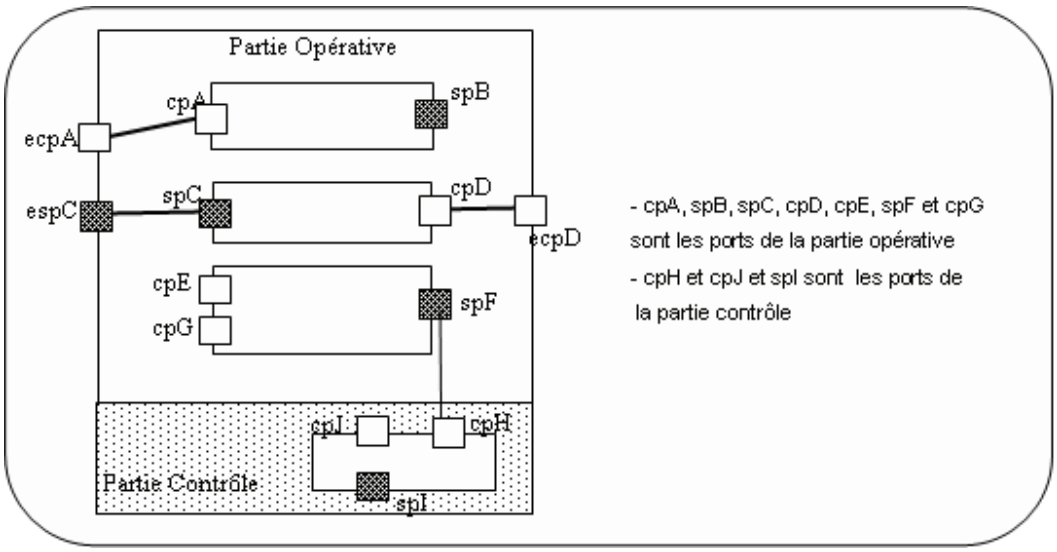


Figure 34. Vue interne d'un composant composite

3.3.1.2.1 La partie opérative

Elle contient les composants représentants par leurs interconnexions la logique générale du composite à un moment bien précis. Les instances de composants et connecteurs sont soit statiques soit dynamiques. Une instance statique est définie en tant que telle à la spécification. Elle est toujours présente dans la partie opérative et ne peut en être supprimée. Une instance dynamique peut disparaître et apparaître durant tout le cycle de vie de l'instance de son composite. Elle peut ainsi être créée, s'auto détruire ou être supprimée.

En plus des instances de types publiques de composants, la partie opérative peut comporter l'instanciation de type de composants internes. L'instanciation de ces types internes n'est possible qu'au niveau de la partie opérative du composant dans lequel les types internes ont été définis. Les types internes ont été introduits pour représenter des aspects très spécifiques à un composant. Les interfaces homme machine sont un exemple d'applications où il y a une mise en œuvre intense du concept de types internes de composant.

3.3.1.2.2 La partie contrôle

La partie contrôle réalise les diverses opérations de contrôle sur les composants de la partie opérative, tels que la gestion du flux de contrôle des divers services (arrêt, lancement en séquence ou en parallèle), le contrôle de l'évolution structurelle, la gestion des exceptions, l'exportation des états du composant et la génération de log. La partie contrôle est composé d'au moins un composant, appelé le contrôleur. Ce dernier est un composant dit comportemental. Sa spécification est faite complètement en langage d'action 3ADL. Un composant comportemental est un type primitif dans le sens ou il ne dispose pas de structure interne accessible.

3.3.1.3 Déploiement des composants

Le modèle de composant que nous avons défini permet une haute flexibilité dans la spécification des propriétés de déploiement des composants. Ces propriétés sont utilisées pour produire le code, et éventuellement les descripteurs de déploiement effectif nécessaires. Ces codes et descripteurs sont utilisés pour charger les composants dans leurs environnements d'exécution. Le déploiement adressé dans IASA, ne concerne pas la méthode de chargement et de lancement des composants dans leurs environnements mais la production du code et descripteurs qui seront utilisés par les outils de déploiement effectif des composants dans leurs environnements d'exécution.

La spécification du déploiement consiste à définir pour un type de composant des cas de déploiement et des plans de déploiement. Les cas de déploiement renseigne sur les formes réelles que le composant pourra avoir une fois en exécution (i.e. tache, processus). Le plan de déploiement indique comment un cas de déploiement est appliqué (i.e. tache dans un processus fonctionnant sur la machine locale).

3.3.1.4. Les cas de déploiement :

Le cas de déploiement spécifie l'état du composant à l'exécution (i.e. tache principale, web service, processus). L'approche IASA a défini plusieurs cas de déploiement direct parmi lesquels nous citons à titre d'illustration: PROCESS, THREAD, APPLET, SERVLET, EJB[5], et JAVASCRIPT. Il est clair que cette liste est extensible pour la prise en charge de cas de déploiement divers. A titre d'exemple, dans le cas d'une conception mixte hard soft, le cas de déploiement pourrait être un circuit VLSI dans lequel le composant logiciel serait déployé. Dans notre cas, il serait nécessaire d'introduire un cas de déploiement qui indique que la nature du composant à déployer est un service web.

Chaque cas de déploiement doit être spécifié avec son environnement. A titre d'exemple, pour le cas PROCESS, il est nécessaire d'indiquer le type du système d'exploitation et l'adresse Internet de la machine.

En plus de ces cas de déploiement, IASA définit 3 autres cas de déploiement à travers lesquels est spécifiée la nature globale du flux de contrôle. Ces cas sont FULLY_CONTROLLED_FLOW, CONCURRENT_FLOW, FREE_FLOW. Ces cas additionnels peuvent être utilisés pour la synthèse de l'architecture finale, prête à l'exécution. Dans cette architecture, les connecteurs qui ne sont pas compatible avec ces cas de déploiement seront inhibés durant l'exécution de l'architecture et aucune communication ne sera véhiculée par les connecteurs inhibés.

3.3.2 - l'ADL de l'approche intégrée :

La première version de l'ADL de l'approche IASA, appelé SEAL (Simple and eXtensible Architecture Language) [7] a été définie afin de démontrer le concept de composant exécutable à un haut niveau d'abstraction. Cependant l'approche IASA

[5] Le choix de ces cas est du à l'utilisation du langage JAVA qui dispose d'une grande variété de technologies d'implémentation

nécessite un langage plus complet pour la spécification des diverses caractéristiques du modèle. C'est pour combler les défaillances de SEAL et pour permettre d'exploiter le modèle IASA de manière efficace que nous avons contribué à la définition du nouveau langage pour l'approche IASA et qui a été appelé 3ADL (Architecture, Aspect and Action Description Language). De plus nous avons défini la forme XML de ce langage que nous avons appelé x3ADL (eXtensible Architecture, Aspect and Action Description Language). x3ADL est défini pour être fortement extensible, flexible et adaptable afin de servir aux expérimentations au fur et à mesure de l'avancée des recherches. Le langage x3ADL rejoint la famille des langages de description d'architecture basés sur le langage XML, tel que XArch [72], xADL [73], xAcme [74], xOLAN [75].

3.3.2.1 Description des différentes balises X3ADL :

Un document x3ADL est hiérarchisé en plusieurs niveaux, le premier niveau est illustré dans comme suit:

```
<?xml version="1.0" encoding="UTF-8"?>
<Component name="Nom_composant"  deployedAs="deployment_case">

      <!--Définition du composant-->
  <Inputs>
      <!—les input du composant-->
  </Inputs>

  <Outputs>
      <!—les output du composant-->
  </Outputs>

  <Ports>
      <!--Définition des ports du composant-->
  </Ports>

  <Connectors>
       <!--Définition des types de connecteurs-->
  </Connectors>

  <OperativePart>
       <!--  définition de la partie opérative-->
  </OperativePart>

  <ControlPart>
       <!--   définition de la partie contrôle-->
  </ControlPart>

  <Properties>
```

```
    <!—Spécification des propriétés non fonctionnelles (déploiement pour
    L'instant -->
  </Properties>

</Component>
```

3.3.2.1.1La balise <Inputs> :

Cette balise indique les entrées du composant composite c'est-à-dire les DOAP ayant le sens IN (ou DOAPin) du composant composite.

<Inputs> est composé donc de plusieurs balises <AccessPoint>

```
 <Inputs>
    < Acesspoint name="x" type="intDataPoint" />
    < Acesspoint name="y" type="intDataPoint" />
 </Inputs>
```

3.3.2.1.2 La balise <Outputs> :

Cette balise indique les sorties du composant composite, c'est-à-dire les DOAPout du composant composite.

Comme la balise<Inputs>, <Outputs> est composé de plusieurs balises <AccessPoint>

```
   <Outputs>
     < Acesspoint name="result" type="FloatDataPoint" />
   </Outputs>
```

3.3.2.1.3 La balise <Ports> :

Cette balise est une descendante directe de la balise <Component>. Elle englobe un ensemble de balises <port>

```
<Ports>

   <port><!—Définition du premier type port --> </port>
   <port><!—Définition du deuxième type port --> </port>
    ……
   <port><!—Définition du nième type port --> </port>

</Ports>
```

Chaque balise <port> définie un nouveau type de port en indiquant ses points d'accès <Accesspoint> définit un point d'accès, en spécifiant un ensemble d'attributs ; le type du point d'accès (attribut type), le nom de l'instance (attribut name), le sens de communication (attribut sens qui peut prendre les valeur IN, OUT ou INOUT), la

synchronisation (attribut synchrounious **yes** si le point d'accès est synchrone **no** sinon) et enfin le temps de validité d'une action ou d'une donnée (attribut temp, il prend la valeur 0 lorsque le temps de validité est infini).

```
<port name="Typeport1">

  <!—Définition des points d'accès -->
   <Acesspoint type="intDataPoint" name="SMainAp" synchronious="no"/>

   <Acesspoint type="ClientActionPoint" name="CMainAp" synchronious="yes"
    temp="0"/>

   <Acesspoint type="IntDataPoint" name="pMainStatus" sens="OUT"
    synchronious="yes" temp="0"/>


</port>
```

3.3.2.1.4La balise <Connectors> :

La balise Connectors regroupe un ensemble de balises connector, dans lesquelles les différents types de connecteurs sont définis

```
<Connectors>

   <connector><!—Définition du premier type de connecteur --> </connector>
   <connector><!—Définition du deuxième type de connecteur --> </connector>
    ……
   <connector><!—Définition du nième type de connecteur --> </connector>

</Connectors>
```

Chaque connecteur est identifié par son nom et est défini par ses rôles, son architecture et son comportement (respectivement les balises <Roles>,<ConnectorArchitecture> et <Behaviour>

```
<connector name="CON">
     <Roles>
              <!—Déclaration des rôles  -->
     </Roles>
     <ConnectorArchitecture>
              <!—Déclaration du type de connecteur en se basant sur des
              types de connecteurs existants -->
     </ConnectorArchitecture>
     <Behaviour>
              <!—Définition du comportement-->
     </Behaviour>
</connector>
```

Lorsqu'il s'agit d'un connecteur prédéfini la balise connecteur est juste identifié par son nom : `<connector name="SOAPTOSOAP"/>`

3.3.2.1.5 La balise <OperativePart>:

Dans la partie opérative les composants primitifs entrants dans la composition sont indiqués dans la balise Components

```
<OperativePart>
  <Components >
   <import name="add-0" type="ADD" />
   <import name="sub-1" type="SUB" />
   <import name="mul-2" type="MUL" />
   <import name="sqr-3" type="SQR" />
  </Components >
</OperativePart>
```

3.3.2.1.6 La balise <ControlPart> :

La partie contrôle est décrite par trois balises principales

- La balise <DelegateConnectors> : le connecteur de délégation se charge de faire la correspondance entre les points d'accès du composants composite et les points d'accès des composants qui le compose.
- La balise <OPControler> : le contrôleur de la partie opérative gère les connexions entre les divers composants
- La balise <DataFlow> : le flux de données est enregistré dans cette balise

```
<ControlPart>
 <DelegateConnectors>
    <map var="x" to-cmp="add-0" in="param0" />
    <map var="y" to-cmp="add-0" in="param1" />
 </DelegateConnectors>

 <OPControler name="FOpPartCtrler">
   <Connecions>
     <connect cmp="add0.ret" To_cmp="mul2.param0" using="SOAP_TO_SOAP" />
   </Connecions>
 </OPControler>

 <DataFlot>
  <fire name="add-0" />
  <fire name="mul-2" />
 </DataFlot>
</ControlPart>
```

3.3.2.1.7 La balise <Properties> :

Cette balise définit les éléments de l'architecture et décrit le déploiement des composants via les balise <architecture> et <deployment> respectivement ; <architecture> donne les informations générales sur la machine comme son nom, son système d'exploitation.

<deployment> donne les informations sur le déploiement du composant.

```
<Properties>
   <architecture>
      <!—Définition des éléments d'architecture-->
      <Environement name="local">
         <Machine>localhost</Machine>
         <OS>UNIX</OS>
      </Environement>
      <Environement name="192.168.0.1">
         <Machine>192.168.0.1.test.dz</Machine>
         <OS>UNIX</OS>
      </Environement>
   </architecture>

   <Deployment>
      <!—Description du déploiement de composants  -->
      <deploymentMap name="map1" flow = "CONCCURENT_FLOW">
         <deploy component="X25GC" As="PROCESS" In="local" />
         <deploy component="x25" As="MAIN_THREAD" In="COMPOSITE" />
         <deploy component="controler" As= PROCESS In="192.168.0.1" />
      </deploymentMap>
   </Deployment>
</Properties>
```

3.4 - Conclusion :

L'architecture logicielle est devenue une activité de conception explicite dans le processus de développement. Elle modularise une application sous forme d'une configuration de composants et de connecteurs. L'architecture logicielle permet d'identifier des entités de connexion appelées connecteurs et de décrire de manière précise les manières de communication entre composants Dans ce chapitre nous avons mis l'accent sur l'architecture logicielle et nous avons présenté l'approche intégrée qui

constitue le cœur de notre contribution. Le chapitre 4, sera dédié à la représentation de notre approche de composition de services Web selon IASA.

CHAPITRE 4
LA COMPOSITION DES SERVICES WEB SELON L'APPROCHE IASA

4.1 Introduction

Dans ce chapitre nous présentons les concepts et mécanismes mis en œuvre dans IASA dans un processus de composition de service Web. Ce processus commence par une spécification abstraite de la composition et s'achève par un processus de transformation de la vue abstraite en une vue d'implémentation dans une technologie de réalisation bien précise. Au niveau abstrait nous avons introduit un modèle de composant IASA spécifique pour la représentation des services Web à un haut niveau d'abstraction. Au niveau du processus de transformation, nous avons enrichis IASA par trois nouveau cas de déploiements spécifiques aux services Web. Concernant la production de la vue implémentation, l'approche IASA se base exclusivement sur la technologie JAVA. Dans ce contexte, nous avons enrichis le processus de transformation d'IASA par un ensemble de règles de transformation permettant produire des services Web réalisé totalement en JAVA à partir d'une description abstraite en x3ADL.

4.2 Représentation des composants « Services Web » dans IASA

Dans IASA, tout composant IASA, quelle qu'il soit pourrait être déployé tant que service Web. Cette capacité est principalement du au fait que tout composant IASA est réalisé à l'aide de composant primitif représenté par des POJOs[1] dont l'objectif est de représenter uniquement et purement le métier pour lequel le composant est destiné. Les fonctions techniques nécessitées par un composant lui seront associé grâce à l'approche de conception par aspects supporté par IASA. Cependant, après un travail expérimental sur les services Web et le problème de leur composition, nous nous sommes rendus compte que le problème devenait très complexe si l'on souhaitait traiter n'importe quel composant comme service Web.

Ainsi, afin de réduire la complexité du problème à traiter et pouvoir réaliser la composition de services Web de manière efficace, nous avons définis dans le

[1] Plain Old Java Object

contexte d'IASA un modèle de composant spécifique pour représenter un service Web. La figure 35 présente la notation graphique de ce modèle. La spécificité concerne uniquement la vue externe du composant et ne remet pas en cause la vue interne ni les capacités d'IASA à manipuler la vue externe d'un composant notamment par l'opération de manipulation individuelle de points d'accès et l'injection des aspects. La vue externe d'un composant représentant un service web simple ou composite comporte deux ports : Un port de service et un port de données. Le port de service qui est du type *IASAServerPort* [6] comporte en plus du point de service, des points d'accès aux données ayant le sens IN (*DOAPin*). Le port de données est le port à travers lequel le service Web renvoie la réponse. Il contient un point d'accès aux données ayant le sens OUT (DOAPout).

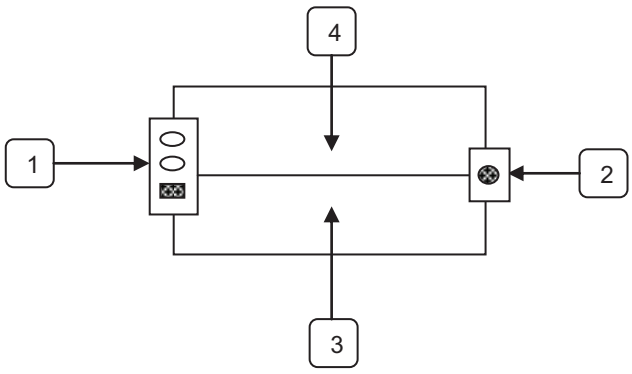


Figure 35. Architecture d'un service Web composite dans IASASTUDIO

4.3 Spécification du comportement du contrôleur :

Le contrôleur est le composant de base dans la spécification de n'importe quel composant composite. Dans IASA ce composant doit être obligatoirement instancié dans la partie contrôle. Il représente une sorte de procédure principale du composant comme l'est la fonction main dans un programme C ou JAVA.

Le contrôleur est dit un composant comportemental. Dans la version actuelle d'IASA il ne peut pas être à son tour un composant composite. Il possède un comportement défini en x3ADL. En règle général, le comportement spécifié concerne l'initialisation du composite, la spécification du flot de contrôle global du

composite et la gestion du dynamisme de l'architecture du composite (ajout suppression de composant, de connecteurs etc..).

Dans IASA le flot de contrôle pourrait être totalement centralisé, totalement décentralisé ou mixte. La centralisation du contrôle se fait par sa spécification au niveau du composant de contrôle de la partie contrôle. Cette approche correspond fortement à la composition par orchestration.

La décentralisation est mise en œuvre par le concept d'enveloppe associée aux diverses instances de composant de la partie opérative. Cette deuxième approche s'apparente beaucoup à la composition par chorégraphie.

Dans la spécification mixte, le flot de contrôle est géré simultanément par le concept d'enveloppe présent sur toutes les instances de composant et par le composant de contrôle.

Dans notre cas, nous nous sommes limités à la spécification centralisée du flot de contrôle et qui correspond à la technique d'orchestration. Nous verrons par la suite que cette forme de spécification est associée à un nouveau cas de déploiement que nous avons introduit dans IASA et qui s'appelle ORCHESTRATED_SOA. Ainsi, dans le contexte d'une composition de service Web par orchestration, le contrôleur invoquera les composants services Web un par un à travers leur port de service et récupère à la fin de chaque activation le résultat à partir du port de données. La forme générale en x3ADL de l'activation et la réception du résultat est comme suit :

```
<fire service= « NomDuPortDeService » result = « nomDuPortDeService/>
```

Cependant, comme nous nous basons sur un modèle spécifique de composant IASA pour les services web (Un seul port de service et un seul port de résultats, nous utiliserons la forme simplifiée suivante :

```
<fire service= « NomDuComoosantDeService »\>
```

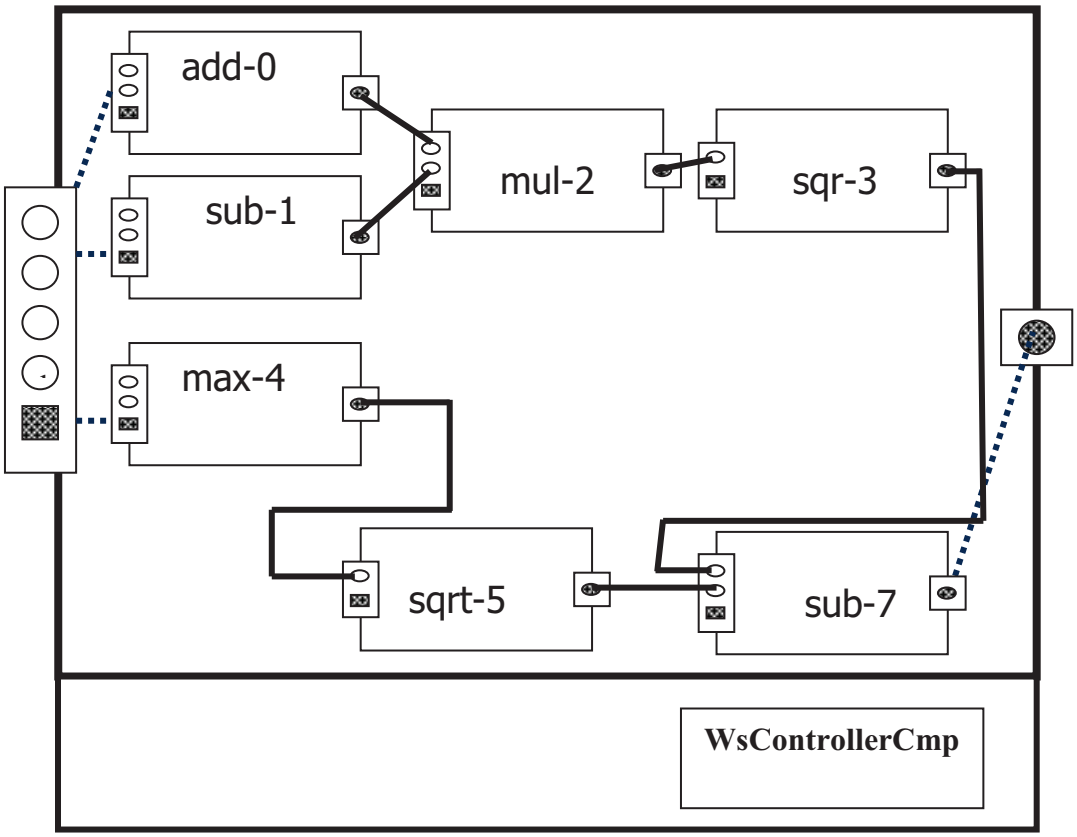


Figure 36. Un service web composite selon IASA

Le code suivant montre un exemple de spécification simplifiée x3ADL du flot de contrôle pour le composite de la figure 36 :

```
<DataFlot>
  <fire  name="add-0" />
  <fire  name="sub-1" />
  <fire name="max-4" />
  <fire  name="mul-2" />
  <fire  name="sqr-3" />
  <fire  name="sqrt-5" />
  <fire  name="round-6" />
  <fire  name="sub-7" />
</DataFlot>
```

4.4 Spécification du déploiement

L'approche IASA définit plusieurs cas de déploiement à travers lesquels apparait clairement l'utilisation intensive de la technologie Java comme cible dans un

processus de transformation d'une description x3ADL vers une technologie d'implémentation. Parmi ces cas de déploiement il y a ceux qui indiquent la nature exacte du composant lors de sa mise en œuvre et des cas qui indiquent la nature du flot de contrôle à appliquer sur l'architecture. Ce dernier se base principalement sur le flot de contrôle définit au niveau du contrôleur instancié au niveau de la partie contrôle.

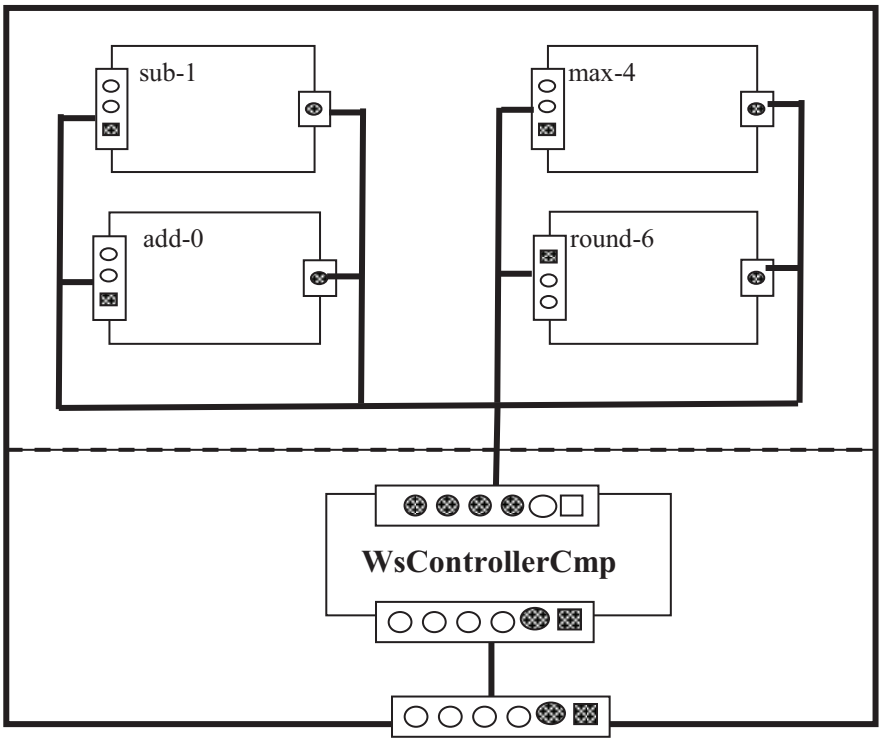
Parmi les cas de déploiement à travers lesquels la nature du composant est indiquée, nous retrouvons le cas PROCESS, THREAD, EJB, SERVLET, APPLET et JAVABEAN. Pour la précision de la nature du flot de contrôle dans une architecture, IASA a définit les trois cas de déploiement suivant : FULLY_CONTROLLED_FLOW, CONCURRENT_FLOW, FREE_FLOW. Actuellement les deux derniers cas de déploiement doivent être utilisés avec beaucoup de précaution car ils ne sont pas encore complètement gérés par IASA[2]. Ils font l'objet d'une recherche dans le cadre d'un mémoire de doctorat.

Pour supporter de manière efficace la mise en œuvre des services Web dans le contexte d'une composition pure de service Web ou l'utilisation de ceux dans des compositions hétérogènes, nous avons introduit trois nouveaux cas suivant : WS, ORCHESTRATED_SOA et CONCURENT_SOA. Le cas WS indique que le composant doit être déployé en tant que service Web. Le second et le troisième permettent d'agir au niveau de la gestion du flot de contrôle. Le premier impose une composition par orchestration. Comme pour le cas FULLY_CONTROLLED_FLOW, le cas ORCHESTRATED_SOA indique que le flot de contrôle doit être totalement sous le contrôle du composant de contrôle. En plus, le contrôleur doit réaliser certaines opérations spécifiques aux Architectures Orientées Service tels que la consultation d'un serveur UDDI.

Le deuxième permet de mettre en œuvre la composition par orchestration. La spécifité du cas CONCURENT_SOA par rapport au cas CONCURRENT_FLOW est représenté par las spécificités des Architectures Orientées Services.

[2] Ces deux cas de déploiements sont en cours de traitement dans un travail de recherche dans le cadre d'un Doctorat à l'ESI intitulé « Architecture Exécutable » et en préparation par Mr Saadi Abdelfettah sous la direction de Mr Henni et Mr Bennouar

Dans les cas de déploiement FULLY_CONTROLLED_FLOW et ORCHESTRATED_SOA, le contrôleur peut remettre en cause la spécification d'une topologie lors de l'opération de transformation d'une spécification x3ADL vers une spécification dans une technologie d'implémentation, afin que celle-ci soit exempt de toute opération de toute activité concurrente non indiqué explicitement au niveau du contrôleur. Dans les cas CONCURRENT_FLOW, FREE_FLOW et ORCHESTRATED_SOA les composants peuvent imposer leur flot de contrôle et évoluer indépendamment de la logique du contrôleur. Cependant, cette liberté est limitée pour le cas ORCHESTRATED_SOA qui doit respecter les exigences imposé par la technique de composition par orchestration.



La figure 37 Architecture effective du flot ORCHESTRATED_SOA (vue partielle)

Dans notre travail nous nous sommes limités à l'exploitation du cas de déploiement ORCHESTRATED_SOA

4.5 Transformation d'une description IASA en une spécification service Web :

Si ce n'est le modèle spécifique particulier au service Web et que nous avons introduit tout à fait au début de ce chapitre, la composition de services Web n'a rien de différents de la composition de composants pour d'autres technologies tels que les EJB ou les CCM ou tout simplement le langage JAVA.

La différence fondamentale apparaît lors de la spécification des propriétés de déploiement des composants. C'est à ce moment la que IASA attache un composant à une technique d'implémentation et une nature bien précise du composant dans cette technologie.

L'approche IASA permet la spécification libre de topologies à un haut niveau d'abstraction, totalement indépendante des divers mécanismes logiciels, notamment le mécanisme d'appel de procédure. Le processus de transformation de la vue abstraite en une vue d'implémentation dépend de plusieurs facteurs tels que la mise en œuvre ou non de l'orienté aspect, la topologie spécifiée, le mode de communication des points d'accès (synchrone, asynchrone), la technologie d'implémentation ciblée, les cas de déploiement associé à chaque instance, et les standards d'interconnexion utilisés.

En règle générale, le processus de transformation d'une spécification x3ADL vers une spécification dans une technologie d'implémentation bien précise est composé de trois phases :

1. Le tissage des aspects
2. La normalisation
3. La production de la vue d'implémentation

Le tissage d'aspects se base principalement sur le concept d'enveloppe. C'est au niveau de celle-ci que les types de ports de composant sont transformés par ajout de nouveau point d'accès appelé ASPOAP. La transformation de ces ports entraîne la création de nouveau type interne de ports. Le résultat de cette phase est une

description x3ADL dans laquelle les opérations d'injection d'aspects sont totalement résolues.

La phase normalisation transforme une description x3ADL en une description basée sur les concepts ordinaires de port et d'interface. Nous rappelons que le port IASA, contrairement aux ports basés sur le concept d'interface, permet à un haut niveau d'abstraction, la manipulation individuelle de tout élément structurel ou comportemental le définissant. Ceci n'est pas le cas dans les autres approches d'architecture logicielle ou le port, appelé aussi interface, est un concept atomique, ne permettant pas la manipulation de ses éléments constitutifs.

Le résultat de la phase normalisation est une description qui permet une synthèse automatique et claire du niveau d'implémentation. A titre d'exemple, la normalisation permettra d'obtenir une description basée sur les ports et interface UML, une description basée sur les interfaces Java ou une description utilisant les ports d'ArchJava [Référence] A titre d'exemple, la vue externe du composant add-0 de la figure 36 serait représentée par l'interface Java suivante :

```
package iasa.component.javaws.primitive;
import iasa.ports.*;
public interface AdderPort extends IASAServerPort {
    public int add(int a, int b);
}
```

La phase de normalisation s'achève par la détermination de la topologie finale de l'application. La topologie produite par la phase de normalisation détermine les connecteurs qui seront effectivement mis en œuvre lors du fonctionnement de l'application. Cette phase dépend essentiellement du flot de contrôle spécifié au niveau du composant de contrôle et du cas de déploiement. Le flot de contrôle spécifié permettra de déterminer les connecteurs qui seront réellement mis en œuvre. Les voies de communications qui ne seront pas utilisée seront supprimées et les voies imposées par le flot de contrôle seront établis. A titre d'exemple, lorsque le cas de déploiement est FULLY_CONTROLLED_FLOW et que le flot de contrôle spécifié au niveau du contrôleur est un flot synchrone adressant uniquement et explicitement les ports principaux de toutes les instances de composant de la partie

opérative (pas de concurrence dans le flot de contrôle), alors l'architecture produite par la phase de normalisation est une architecture dans laquelle toutes les connexions inter composant seront inhibées. De plus, si les connecteurs entre contrôleur et ports principaux de composant n'ont pas été spécifiés, ces derniers seront établis lors du processus de production d'une architecture ordinaire. Le cas ORCHESTRATED_SOA est spécifique par rapport au cas FULLY_CONTROLLED_FLOW par le fait qu'il impose un flot synchrone adressant uniquement et explicitement les ports principaux de toutes les instances de composant de la partie opérative comme indique sur la figure 36.

La troisième phase du processus de transformation utilise l'architecture régulière produite par la phase de normalisation (Figure 37). Cette architecture est définie exclusivement à base d'interface ou port ordinaire (exemples : Interface Java, port ArchJava). L'objectif principal de cette phase est la production de la vue d'implémentation de l'enveloppe. C'est au niveau de cette phase, que sont installés dans l'enveloppe les adaptateurs pour solutionner le problème d'incompatibilité d'interface et le problème de distance entre ports interconnectés. Lorsque les ports interconnectés sont totalement compatibles (exemple : port dotés d'opération ayant exactement la même signature, port standard tels que FTP, HTTP, SOAP), cette phase est réduite à la production d'une enveloppe dite transparente dont l'objectif principal est l'interception de toute information ou actions.

4.6 Les règles de transformation :

La phase de normalisation est la plus importante dans le processus de génération de la vue d'implémentation. Elle met en œuvre un ensemble de règles de transformation pour produire une architecture régulière à base d'interface ordinaires. Deux types de règles sont mise en œuvre dans cette étapes : Les règles générales et les règles spécifique à chaque technologie d'implémentation. Les règles générales indiquent comment passer d'une description de port IASA à une description en termes d'interfaces. Les règles générales ne dépendent d'aucune technologie d'implémentation. La spécification du transporteur d'un DOAP qu'il soit utilisé indépendamment ou dans le contexte d'une action est une de ces règles. Ainsi, à titre d'exemple, un DOAP peut être cité dans plus d'une action. Dans ce cas l'action

citée représente le transporteur de l'information fournie ou requise par le DOAP. Lorsque le DOAP est interconnecté individuellement, il doit être pourvu de transporteur spécifique représenté par les méthodes get ou set dépendamment de la manière dont le connecteur est tiré entre les DOAP interconnectés. Une étude poussée de ces règles de transformation est présentée dans [6].

Les règles spécifiques indiquent les techniques de génération de l'enveloppe dans la technologie d'implémentation choisie. Dans IASA le composant pur est protégé par l'enveloppe et n'est pas touché dans une opération d'instanciation. Ainsi à titre d'exemple si un composant est déployé comme une Servlet [Reference], l'enveloppe sera une classe qui étend l'interface HTTPServlet et qui dispose d'une référence sur l'objet représentant le composant instancié. Toutes les informations provenant des requêtes HTTP ou qui seront mise dans une réponse HTTP seront intercepté par l'enveloppe et aiguillé de ou vers l'instance enveloppée. Dans le cas d'un composant EJB3, l'enveloppe sera représentée par une classe dotée d'un ensemble d'annotation Java. Les travaux présentés dans [Conf] et [Conf] présentent un ensemble de règles de transformation spécifique au langage d'ArchJava et aux pages JSP.

4.6.1 Les règles de transformation dans les cas des services Web :

Notre objectif principal est de montrer comment une approche Architecture Logicielle pouvait être utilisée pour résoudre le problème de la composition des services web. Nous avons opté pour l'approche IASA, vu que celle-ci offre plus de possibilité et de flexibilité que les autres approches d'architecture logicielle [Reference Elsevier 2010]. Cependant, dans les études de cas que nous présentons dans ce chapitre, nous n'avons pas utilisé toute les capacités de l'approche IASA, notamment l'orienté aspect, la spécification libre de topologie dans laquelle il est possible d'établir des connecteurs entre point d'accès de ports distincts (nous n'avions pas eu besoin d'utiliser les points d'accès individuellement dans une spécification d'interconnexion) et l'exploitation du cas de déploiement CONCURENT_SOA mettre en œuvre une composition par chorégraphie.

Pour des raisons de validation de notre approche, nous nous sommes suffit d'une représentation standard de port (les points d'accès ne sont pas manipulés indépendamment). De plus comme indiqué précédemment nous avions utilisé l'orchestration comme méthode de composition. L'architecture que nous utilisons dans le contexte d'IASA est une architecture ordinaire dans laquelle les composants sont instanciés dans une enveloppe qui se chargera de fournir les ports standards d'interaction, notamment les ports supportant le protocole SOAP. Les règles de transformation sont réduites ainsi aux règles spécifiques à la technologie d'implémentation qui est représentée dans notre cas par les services web. Le connecteur utilisé pour interconnecter les composants qui seront déployé comme service Web est un connecteur standard représentant le protocole SOAP.

Les règles de transformation générale qui concerne la composition de web service par orchestration seront fortement influencées par ce processus de composition qui est fortement centralisé. Ainsi les connexions qui seront effectivement actives seront celles liant le contrôleur aux divers composants qui seront déployés en tant que Services Web.

Les règles de transformations spécifiques seront celle permettant de générer des services web écrit en Java, vu que pour l'instant, le langage Java représente la seule cible technologique que l'approche IASA utilise. Les services web construit à base du langage java utiliserons l'API JAX-RPC, les annotations pour services Web et le Framework axis 2.

Les règles de transformations mise en œuvre durant la phase de normalisation se résument aux points suivants :

1er Cas : Les composants à déployer comme service Web sont des composants IASA Primitifs représentés par des POJOS.

Les composants représentés par des POJOS peuvent être déployés en tant que Web service ou dans une autre nature telle qu'EJB, SERVLET ou APPLET.

1. Tous les aspects permettant de transformer un composant quelconque, notamment ceux qui sont représenté par des POJO, en services Web feront

partie de l'enveloppe. Ce sera le cas des fichiers WSDL et les Stub pour le port client de service Web.

2. Les connecteurs utilisés entre client de service Web et service Web seront basé sur JAX-RPC.
3. Chaque enveloppe est représentée par un POJO.
4. L'enveloppe doit implémenter la même interface que le composant qu'elle enveloppe. L'implémentation au niveau de l'enveloppe aura un sens d'interception de tout appel entrant ou sortant du composant enveloppé. Dans le cas de la composition par orchestration, l'interception ne concernera en réalité que les appels entrant. Les appels sortant seraient interceptés dans le cas de la composition par chorégraphie.
5. Le constructeur de l'enveloppe doit instancier les POJO représentant le composant enveloppé
6. L'enveloppe doit disposer des références de Stub des autres services Web qu'elle mettra en œuvre dans le cas d'une composition par chorégraphie.
7. L'envelope du controleur doit disposer d'une interface qui est une extension de toutes les interfaces des services web utilisée pour réaliser la composition. C'est par une référebce de cette interface que l'envelope du controleur attreindra toutes les interfaces des services web qu'il orchestre.

L'exemple suivant montre le composant primitif réalisant la fonction d'addition, l'interface qu'il implémente, l'enveloppe représentée par un POJO et les annotations Java et le fichier WSDL décrivant le service Web d'addition.

```
package iasa.component.javaws.primitive;

public class AdderCmp implements AdderPort{
    public int add(int a, int b){
        return a + b;
    }
}
```

```
package iasa.component.javaws.primitive;
import iasa.ports.*;
public interface AdderPort extends IASAServerPort {
    public int add(int a, int b);
}
```

```
package iasa.impl.ws.envelope;
import iasa.component.javaws.primitive.*;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public class AdderImpl {
    AdderPort adder;

    AdderImpl(){
        adder = new AdderCmp();
    }

    @WebMethod(operationName="add" )
    public int add(int a, int b){
    return adder.add(a, b);
    }
}
```

2ème cas : Les composants à déployer en tant que services Web ne sont pas des POJOS mais des composants services web complètement définis :

Ces composants ne peuvent être associés qu'au cas de déploiement WS (Web Service). Du point de vue de l'approche IASA, ces composants considérés comme primitifs, sont soit internes soit externes. Les composant internes résident dans la librairie des composants primitifs d'IASA et peuvent être déployés en plusieurs exemplaires, selon l'architecture spécifiée. Les web services déjà déployés sur serveur sont considérés comme des composants externes. Ces derniers sont représentés dans une architecture par des composants de liaison. Dans le cas ou la conception par aspect, l'exploitation individuelle des points d'accès et la composition par chorégraphie ne sont pas mis en œuvre, le processus de transformation ne concernera que la partie cliente de ces Services Web.

4.7 Production du contrôleur de l'orchestration

Le contrôleur représente en fait le cœur de la composition par orchestration. La production de l'orchestrateur pourra se faire selon l'une des deux méthodes suivantes :

- Le contrôleur est représenté par un composant primitif IASA (un POJO) pouvant être déployé comme n'importe quel autre composant, notamment en tant que Web service. S'il doit être déployé en tant que Service Web, il doit

être conforme au modèle spécifique représentant les services Web. L'orchestration est alors définie dans un fichier XML déployé avec le contrôleur et qu'exploitera ce dernier pour lancer les divers web services selon le processus définis dans le fichier XML.

- Le contrôleur est représenté par un POJO généré à partir de la description du flot de contrôle spécifiée pour le contrôleur et dans lequel l'orchestration est une fois pour toute gravée dans le code POJO du contrôleur.

Dans notre cas, nous avons opté pour la deuxième solution, vu que pour l'instant elle est amplement suffisante pour illustrer notre objectif qui est celui de l'exploitation d'une approche Architecture Logicielle pour la composition des services web.

La production de l'orchestrateur dépendra du cas de déploiement qui lui est associé. Nous considérons que le contrôleur est conforme au modèle de composant IASA représentant les services Web. Dans les deux cas de figure, le POJO représentant le contrôleur est le même. Seule la constitution de son enveloppe qui est différente.

Si le contrôleur n'est pas déployé en tant que service web, il ne représentera pas un service Web composite. Dans ce cas bien précis, l'enveloppe associé au contrôleur ne contiendra que les fichiers WSDL et les Stubs des services web que lancera tour à tour le contrôleur. Ceci est une composition hétérogène.

Lorsque le contrôleur est déployé en tant que service web, il représentera dans ce cas un service web composite. Le processus de production du contrôleur générera une enveloppe qui sera dotées des fichiers WSDL et les stubs des services Web orchestré par le contrôleur ainsi qu'un WSDL décrivant le service web représenté par le contrôleur. L'exemple ci-dessous montre partiellement le POJO représentant le code d'appel du contrôleur :

```
public static int  controller(int a,int b,int c,int d) throws
RemoteException
    {
        ControllerServiceStub stub = new ControllerServiceStub();
        ControllerServiceStub.contr operation = new
        ControllerServiceStub.contr();
```

```
        operation.setParam0(a);
        operation.setParam1(b);
        operation.setParam2(c);
        operation.setParam3(d);
        ControllerServiceStub. ContrResponse r = stub.cont(operation);
        return (int)r.get_return();

    }
```

Ainsi que le fichier WSDL décrivant le composant service web composite présenté dans la figure 36.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ControllerService"
targetNamespace="http://ControllerService.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://ControllerService.wsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:msgs="http://www.example.org/messages">
    <wsdl:types>
      <xsd:schema xmlns:ns="http://ws.apache.org/axis2"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://www.example.org/messages">
         <xsd:element name="ControllerServiceRequest">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element minOccurs="0" name="param0"
type="xsd:int"/>
                  <xsd:element minOccurs="0" name="param1"
type="xsd:int"/>
                  <!--<xsd:element minOccurs="0" name="param2"
type="xsd:int"/>-->
                  <!--<xsd:element minOccurs="0" name="param3"
type="xsd:int"/>-->
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="ControllerServiceResponse">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element minOccurs="0" name="return"
type="xsd:int"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:schema>
   </wsdl:types>
    <message name="ControllerServiceOperationRequest">
        <part name="ControllerServiceRequest"
element="msgs:ControllerServiceRequest"/>
    </message>
    <message name="ControllerServiceOperationResponse">
```

```
        <part name="ControllerServiceResponse"
element="msgs:ControllerServiceResponse"/>
    </message>
    <portType name="ControllerServicePortType">
        <operation name="ControllerServiceOperation">
            <input name="input1"
message="tns:ControllerServiceOperationRequest"/>
            <output name="output1"
message="tns:ControllerServiceOperationResponse"/>
        </operation>
    </portType>
    <binding name="ControllerServiceBinding"
type="tns:ControllerServicePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="ControllerServiceOperation">
            <soap:operation/>
            <input name="input1">
                <soap:body use="literal"/>
            </input>
            <output name="output1">
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="ControllerServiceService">
        <port name="ControllerServicePort"
binding="tns:ControllerServiceBinding">
            <soap:address
location="http://localhost:8080/axis2/ControllerServicePort"/>
        </port>
    </service>

</definitions>
```

4.8 Conclusion

Dans ce chapitre nous avons présenté les grandes lignes qui montrent comment la composition de service Web peut être prise en charge par l'approche IASA. Nous n'avions en réalité exploité qu'une partie des capacités de l'approche IASA et nous nous sommes limités à une spécification ordinaire d'architecture logicielle. Vu la complexité du problème de composition de services Web nous étions amené à définir un modèle spécifique pour la représentation des services Web et pour lequel les cas de déploiements introduits pourraient être appliqué. Dans le chapitre qui suit, pour une fin de validation de notre approche, nous mettrons en œuvre les concepts présentés dans ce chapitre dans le contexte d'un outil que nous avions réalisé pour

traiter spécifiquement le problème de composition de service Web et que nous avons appelé IASASTUDIO.

CHAPITRE 5
IASASTUDIO : UN OUTIL POUR LA VALIDATION DE LA COMPOSITION DE SERVICES WEB SELON L'APPROCHE IASA

5.1 Introduction

Afin de pouvoir mettre en œuvre l'approche IASA dans le contexte de la résolution du problème de composition de services Web et de valider cette approche par des études de cas, nous avons réalisé un environnement de développement d'applications selon l'approche IASA. C'est ainsi que dans ce chapitre, nous commencerons par la présentation de cet environnement de développement que nous avons appelé IASASTUDIO. Nous présentons tout d'abord l'architecture générale de IASASTUDIO et nous nous intéresserons ensuite aux aspects conception et réalisation de IASASTUDIO. Dans le contexte de la validation de notre approche de composition de services Web, nous utiliserons IASASTUDIO dans une étude de cas sur la composition des Web service selon IASA. Nous terminerons ce chapitre par une étude comparative avec le langage BPEL.

5.2 Présentation d'IASASTUDIO :

IASASTUDIO est l'environnement de développement qui génère des composants composites selon l'approche IASA.

La version actuelle d'IASASTUDIO permet de résoudre des formules mathématiques en composant des fonctions mathématiques de base. Chacune de ces fonctions peut représenter un appel à un service Web ou un appel à une fonction ordinaire.

Lorsque la formule est construite seulement par composition des services Web, elle représente dans ce cas un «service Web composite ». Par contre, si l'une des fonctions composant la formule n'est pas un service web, le résultat serait dans ce cas un « composant IASA composite » (figure 38). IASASTUDIO permet de

concevoir les applications à un haut niveau d'abstraction, et génère automatiquement le code x3ADL.

A chaque fois qu'un événement se produit (création d'un composant, connexion de deux composants, etc.), le code correspondant s'ajoute. Le résultat final est enregistré dans un fichier x3ADL décrivant le projet.

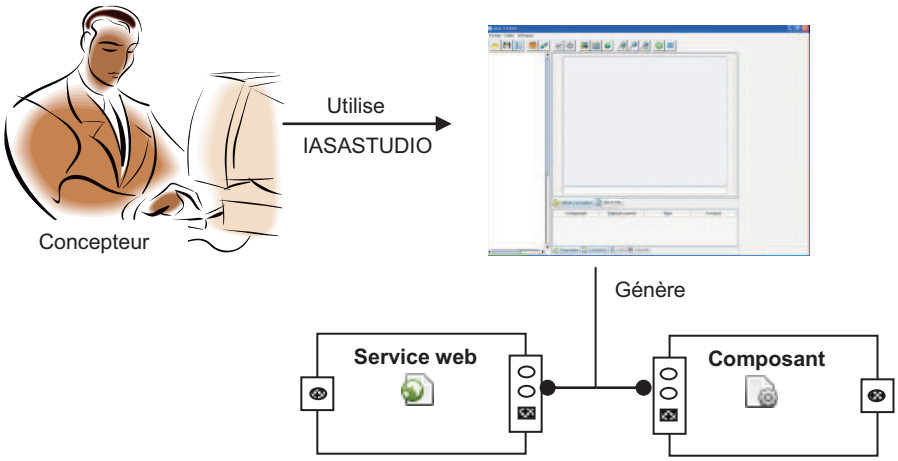


Figure 38. Présentation d'IASASTUDIO

5.2.1 Conception d'IASASTUDIO :

IASASTUDIO est une application totalement codée en java, elle est articulée autours de trois API

1) JGraph : JGraph[1] est un logiciel libre sous licence LGPL écrit en java en s'appuyant sur le composant JTree de SWING, il a été proposé par Gaudenz Alder en 2000 à l'université ETH Zurich en Suisse et diffusé comme logiciel libre en Mai 2002, JGraph a permet de tracer des graphes et des organigrammes Nous avons utilisé JGraph pour tracer les différents composants et les interconnexions entre eux.

[1] http://www.jgraph.com/

2) JDOM (Java Based Document Object Model): JDOM[2] est conçue spécialement pour simplifier l'écriture des documents XML en Java. Nous l'avons utilisé pour écrire le document x3ADL correspondant à chaque projet.

3) Le Framework AXIS (Apache eXtensible Interaction System): Axis[3] est à la fois un environnement d'hébergement de services Web, et un outil complet de développement pour la création de services développé par la fondation Apache . Pour fonctionner, Axis doit être hébergé au sein d'un serveur web. Nous avons choisi dans notre cas le serveur Tomcat[4] 5.5. Dans notre cas Le framework AXIS est utilisé pour générer les Stub client, afin de pouvoir consommer les services web.

5.2.2 La bibliothèque IASASTUDIO :

IASASTUDIO est muni d'une liste de composants primitifs prédéfini,
ADD – SUB – MUL – SQR – SQRT – MIN – MAX – ROUND
Parmi ces composants nous distinguons ceux ayant à leur charge la réalisation de fonctions mathématiques de base telle que l'addition (AdderCmp) la multiplication (MulCmp). Plusieurs composants primitifs tels qu'AdderCmp, MulCmp et SQRCmp possèdent une vue d'implémentation permettant leur déploiement en tant que services Web. Pour notre étude de cas qui servira à la validation de notre approche nous utiliserons les trois composants AdderCmp, MulCmp et SQRCmp. ces trois ont été téléchargés d'internet sous format .aar (Axis Archive). Les fichiers WSDL de ces trois composants sont reportés en Annexe de ce mémoire.

5.2.3 Représentation des composants composite dans IASASTUDIO

Dans IASA un projet est un composant composite. Une composition de services web est réalisée par un composant composite. Un composant composite représenta un service Web composite est conforme au modèle IASA représentant les services

[2] http://www.jdom.org/
[3] http://xml.apache.org/axis
[4] http://tomcat.apache.org

web. La vue externe du modèle IASA représentant un service web est caractérisé par les éléments suivants (figure 39) (voir chapitre 4):

- Un port représentant le service. Il est constitué de points d'accès aux données ayant le sens IN
- Un port de données représentant le résultat. Il contient un point d'accès aux données ayant le sens OUT. Celui-ci retourne la réponse du composant.

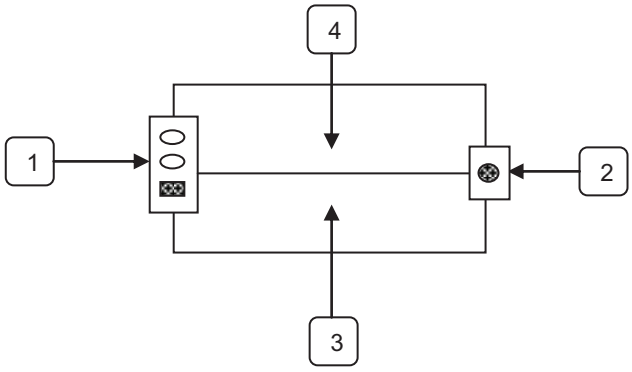


Figure 39. Architecture d'un service Web composite dans IASASTUDIO

5.2.4 Interface graphique d'IASASTUDIO :

La fenêtre principale d'IASASTUDIO (Figure 40) est composée de plusieurs zones dédiées à faciliter la spécification d'une architecture logicielle. Les zones principales sont :

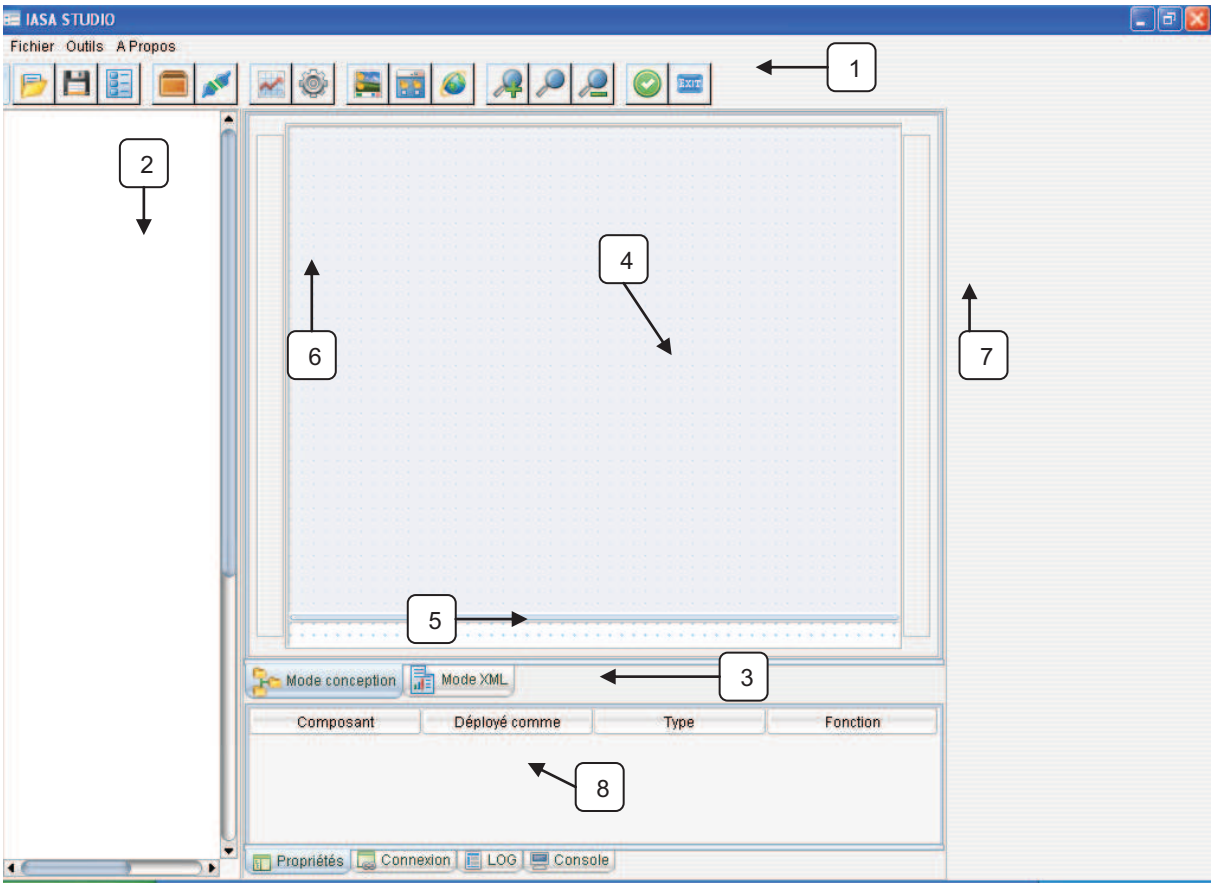


Figure 40. Fenêtre principale d'IASASTUDIO

Zone 1 : La barre d'outils : La barre d'outils d'IASASTUDIO (figure 41) est constituée d'un ensemble de bouton permettant entre autres la création de nouveau projet (bouton 1), la production du fichier x3ADL du projet (bouton 2), l'ajout des port externe du projet (bouton 3), l'importation de composant de la bibliothèque de composants (bouton 4), la connexion des ports de composant (bouton 5) et la spécification du déploiement des instances de composant (bouton 9)

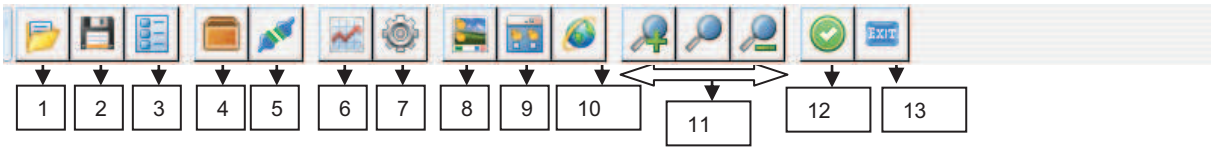


Figure 41. La barre d'outils d'IASASTUDIO

Zone 2 : L'arbre de composition : cette zone affiche les noms des différents composants qui forment le projet (composant composite)

Zone 3 : Onglet (Mode conception / Mode XML) : cet onglet permet de basculer entre le mode conception représenté par le graphe du composant, et le mode XML représenté par le fichier x3ADL.

Zone 4 : La partie opérative : affiche les composants qui forment la partie opérative du projet, ainsi que les liaisons entre ces derniers.

Zone 5 : La partie contrôle : cette partie contient le contrôleur du composant en cours de réalisation dans le contexte du projet ouvert.

Zone 6 : Le port de services cette zone affiche les ports et point d'accès à travers lesquels le composant représenté par le projet spécifie les services offerts

Zone 7 : Le port de réponse : Ce port sert à rendre la réponse du service activé

Zone 8 : Onglets (Propriétés / Connexion / LOG / Console) : l'onglet « propriétés » affiche les noms des composants, le type, et l'état de déploiement. L'onglet Connexion affiche les différents connexions établis entre composants. L'onglet Log affiche les événements importants comme date et heure de création de projet. L'onglet Console affiche le résultat d'exécution pas à pas du projet.

5.3 Mise en œuvre d'IASASTUDIO pour la réalisation d'un composant composite :

Dans ce qui suit, nous allons montrer comment réaliser un composant composite en utilisant IASASTUDIO. Le composant composite que nous projetons de réaliser implémente une fonction f décrite par la formule mathématique suivante.

$$f(x,y,a,b) = [(x+y)\times(x-y)]^2 - \left\lfloor \sqrt{Max(a,b)} \right\rfloor$$

5.3.1 Représentation IASA de la fonction f

Selon l'approche IASA la formule f sera représentée par un composant composite. Les éléments de définition des composants sont des composants primitifs représentant les divers opérateurs mathématiques. Soit FCmp le composant IASA réalisant la formule f. FCmp possède un port de service et un port de données. Le port de service est constitué d'un point d'accès de service (ACTOAP) et de quatre points d'accès aux données (DOAPin) notés x, y, a, b et ayant tous le sens IN. Le port de donnée de FCmp est doté d'un seul point d'accès aux données (DOAPout) ayant le sens out et à travers lequel le composant FCmp renvoie le résultat de la formule f.

Les composants primitifs utilisés pour réaliser le composant FCmp, se chargent chacun de la réalisation des fonctions élémentaires suivantes :

```
Addition de x et y……...(1) réalisée par un composant du type AdderCmp
Sosutarction de x et de y)……...(2) réalisée par un composant du type
SubstractCmp
Multiplication de r(1) et  r(2)…..(3) réalisée par un composant du type
MultiplierCmp
Le carré de r(3)…..……(4) réalisé par un composant du type SquareCmp
Maximum entre a et b……..(5) réalisée par un composant du type MaximumCmp
Racine carrée de (r(5))……..(6) réalisée par un composant du type
SquareRootCmp
Partie entière de r(6)……..(7) réalisée par un composant du type RoundCmp
Soustraction de r(4) et r(7)……(8) réalisée par un composant du type
SubstractCmp
Où r(n) représente le résultat obtenu à l'étape n.
```

-Enchaînement des opérations dans la formule f -

Deux types de composant primitifs ont été utilisés : des composants services web téléchargés d'Internet et intégré à la bibliothèque d'IASA et des POJOs qu'il faudrait transformer en Service Web lors du déploiement.
La réalisation du composant FCmp fait ressortir l'utilisation de huit composants primitifs comme la montre la vue interne du composant FCmp en figure 42

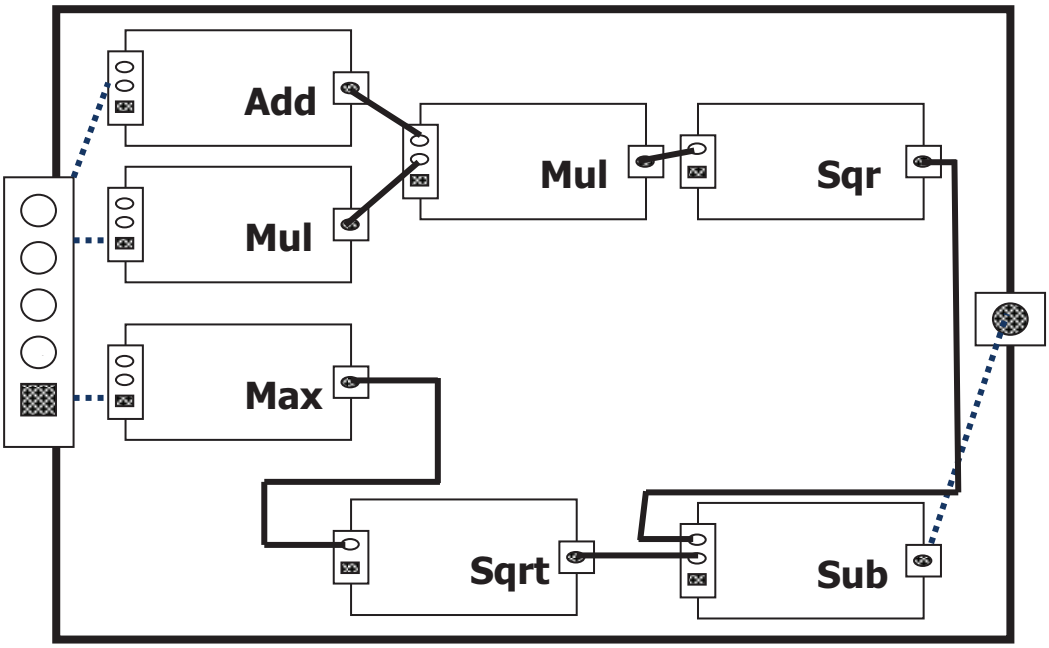


Figure 42. Architecture du composant FCmp

5.3.2 Spécification de FCmp dans IASASTUDIO

La spécification de toute application dans IASASTUDOIO se fait par la définition d'un nouveau composant composite. Cette définition correspond au niveau d'IASASTUDIO à l'ouverture d'un projet dont le nom est celui du composant composite. Dans notre cas ce serait FCmp. La figure 43 montre la fenêtre affichée par IASASTUDIO suite à la demande d'ouverture d'un nouveau composant composite.

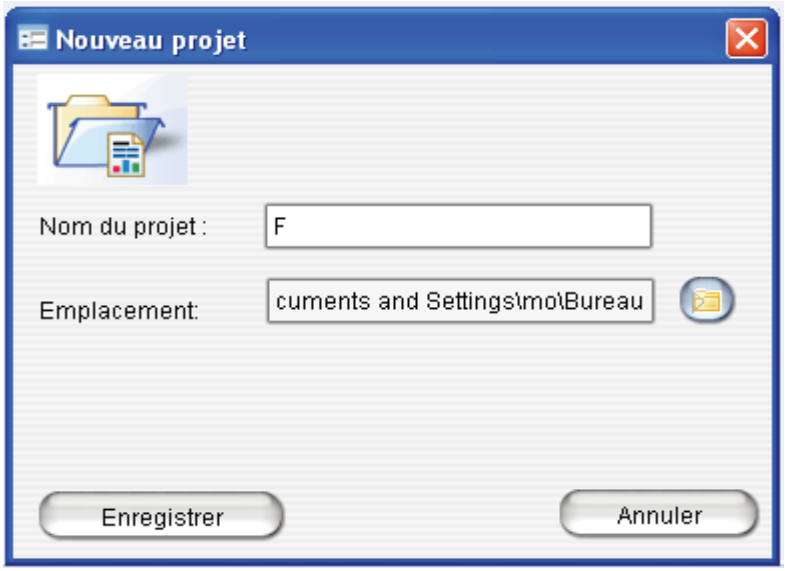


Figure 43. Nouveau projet dans IASASTUDIO

Une fois le projet crée, les éléments suivant sont alors instancié (figure 44):

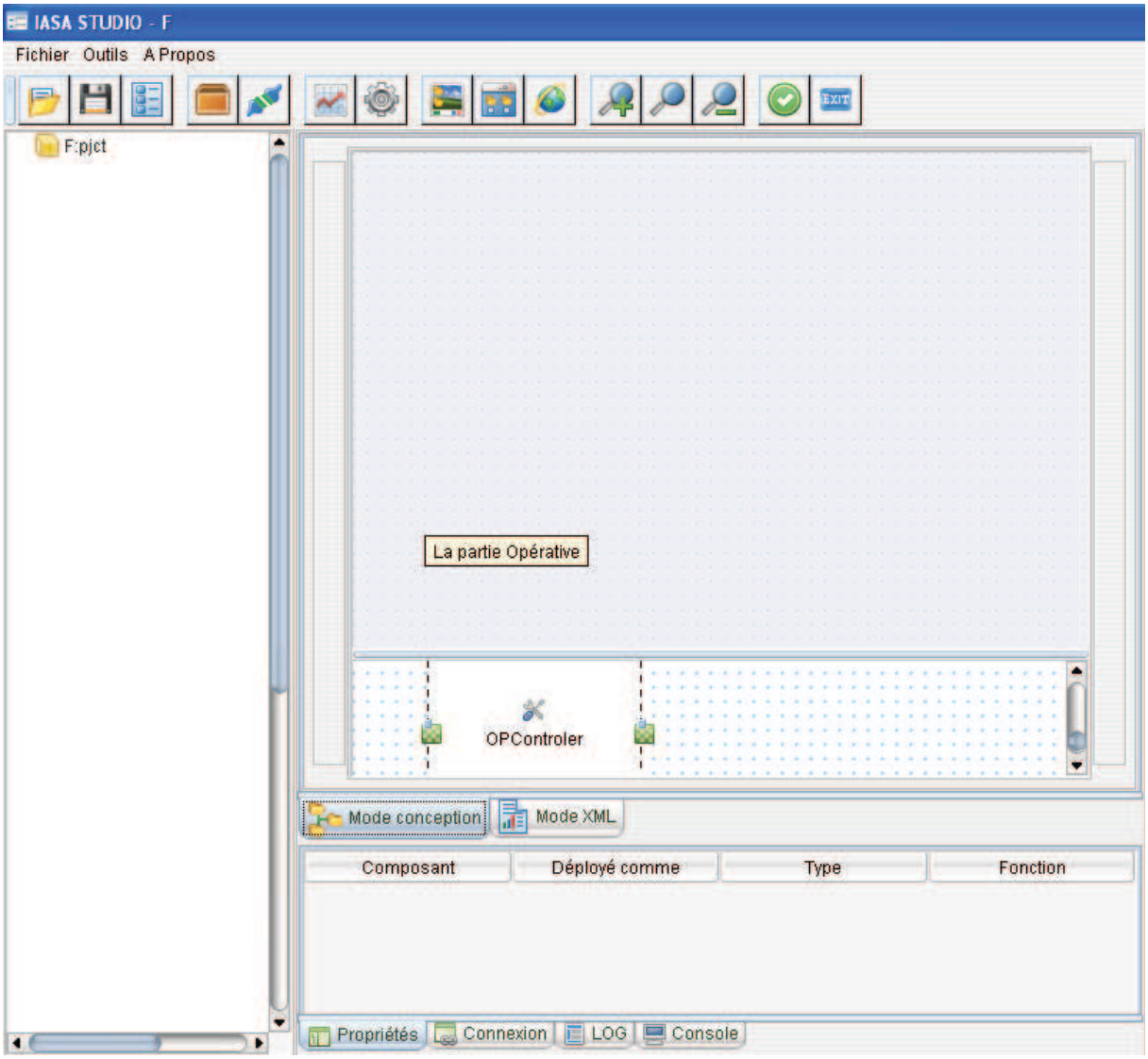


Figure 44. Conception du composant F dans IASASTUDIO

- Un arbre de composition dont la structure est affiché dans la zone gauche de la fenêtre de conception
- Un composant de contrôle est instancié dans la partie contrôle du composant composite.
- Le fichier x3ADL décrivant le nouveau composant composite

5.3.3 Définition de la vue externe de FCmp

Elle consiste à définir la structure des ports du composant FCmp. : Le premier port servant à l'invocation du service est représentée par un point d'accès orienté

action (ACTOAP) et de 4 points d'accès orientée donnée (DOAP) portant **x, y, a**, et **b** (Figure 45). Le port est représenté par un seul point d'accès dont le nom est **result.** Tous les points d'accès des deux ports sont du type intDOAP .La vue externe complète du composant FCmp est illustrée par la figure 46

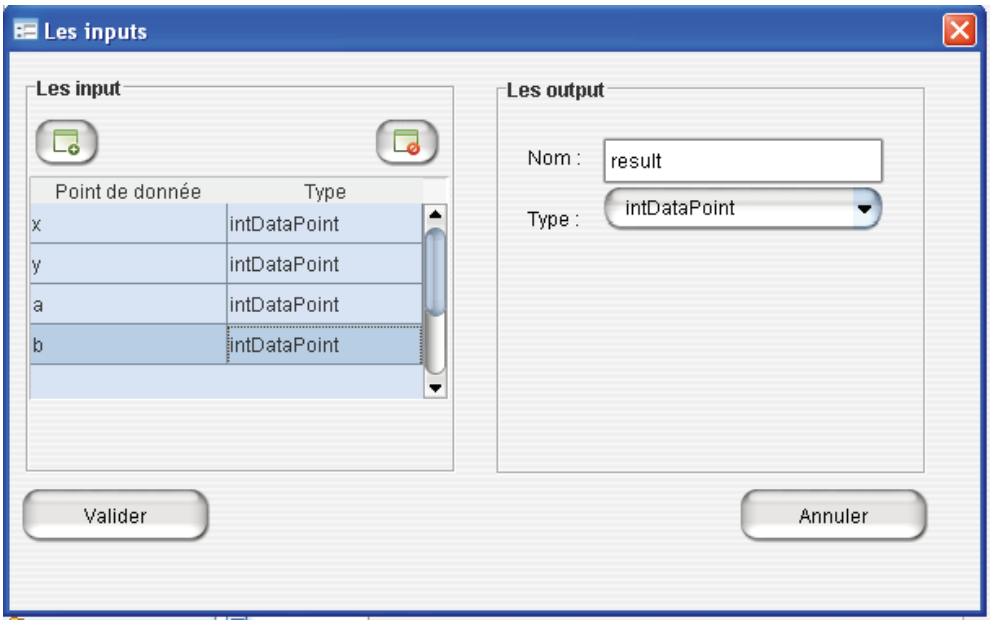


Figure 45. Les points de donnés du composant F

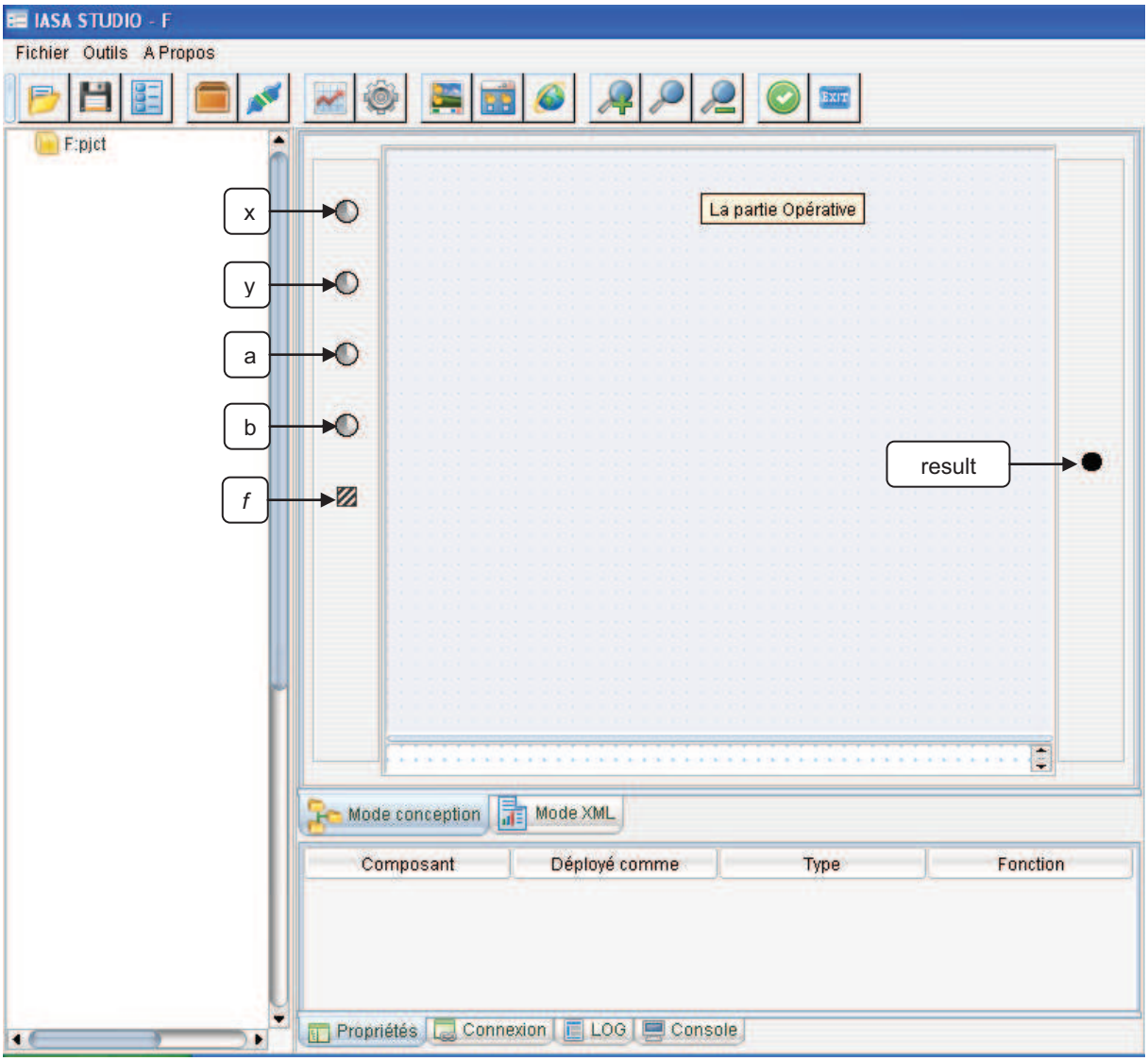


Figure 46. Vue externe de FCmp

5.3.4 Définition de la vue interne

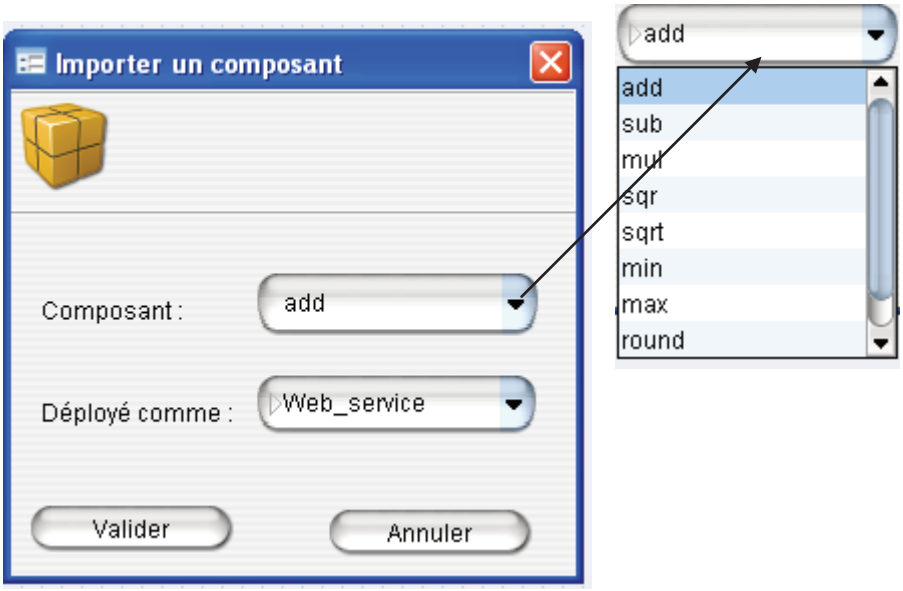


Figure 47. Importation d'un composant dans IASASTUDIO

Les composant définissant de la vue interne du composant FCmp dont l'architecture a été présentée dans la figure 42, sont tous des types primitifs complètement définies dans l'approche IASA qu'il est possible d'importer pour la définition de tout composant composite (Figure 47).

L'instanciation d'un type de composant nécessite la spécification d'un nom d'instance. IASASTUDIO affecte par défaut des noms aux instances selon un style particulier. Ces noms sont redéfinissables par le concepteur.
La figure 48 montre la partie de la fenêtre de conception chargée de montrer la hiérarchie de composition de FCmp

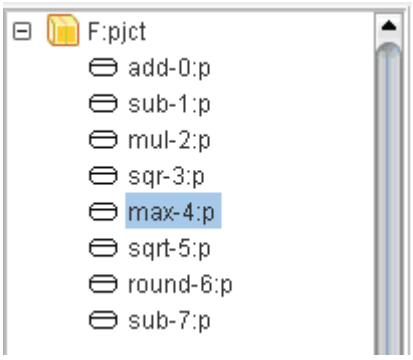


Figure 48. Arborescence de F dans IASASTUDIO

5.3.5 Réalisation des interconnexions :

Dans l'approche IASA la composition de Web services est réalisée par l'opération d'interconnexion des composants entre eux et par la connexion des ports de composant internes aux ports externes du composant composite. Dans le premier type de connexion les connecteurs sont appelés des connecteurs d'assemblage. Dans le deuxième cas nous utilisons les connecteurs dits de délégation

5.3.5.1 Spécification des connecteurs de délégation :

L'objectif est d'indiquer quels sont les ports des composants internes qui seront exporté pour devenir des ports de la vue externe du composant FCmp. Dans IASASTUDIO cette opération est appelée une opération de mapping comme indiquée par la figure 49 :

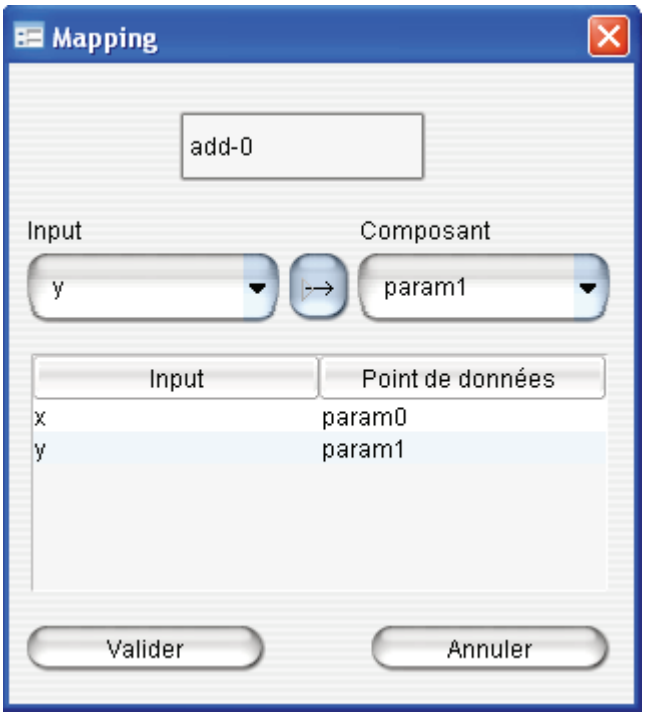


Figure 49. Mapping des variables

5.3.5.2 Interconnexion des composants de la vue interne

Cette opération est réalisée en utilisant des connecteurs IASA explicite. Dans notre situation le connecteur à utiliser correspond à un protocole standard. Dans

IASA les connecteurs standard sont des connecteurs primitifs qui doivent être défini au niveau de la librairie des connecteurs. Comme contribution de notre part, nous avions défini complètement les connecteurs supportant le protocole SOAP. C'est ce type de connecteurs qui est utilisé pour interconnecter les ports des divers services web comme ceci est illustré par la figure 50. La réalisation complète de la composition en utilisant IASASTUODO est illustrée par la figure 51 :

Figure 50. Spécification de Connexions des composants dans IASASTUDIO

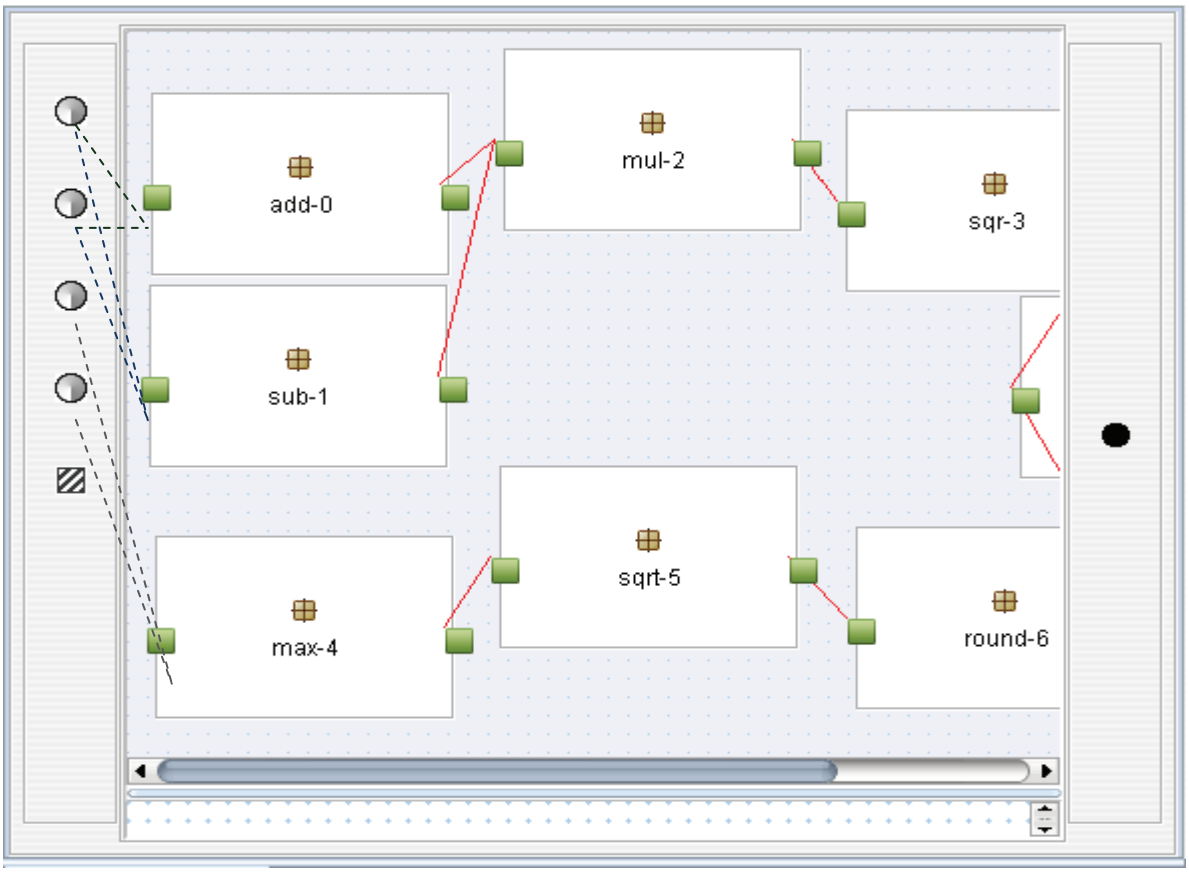


Figure 51. Vue finale du composant F dans IASASTUDIO

5.4 Spécification du flot de contrôle

Après avoir défini les diverses connexions, il est nécessaire d'indiquer le flot de contrôle du composant composite. Cette spécification se fait dans IASA en remplissant la propriété comportement du composant comme indiqué sur la figure 52 :

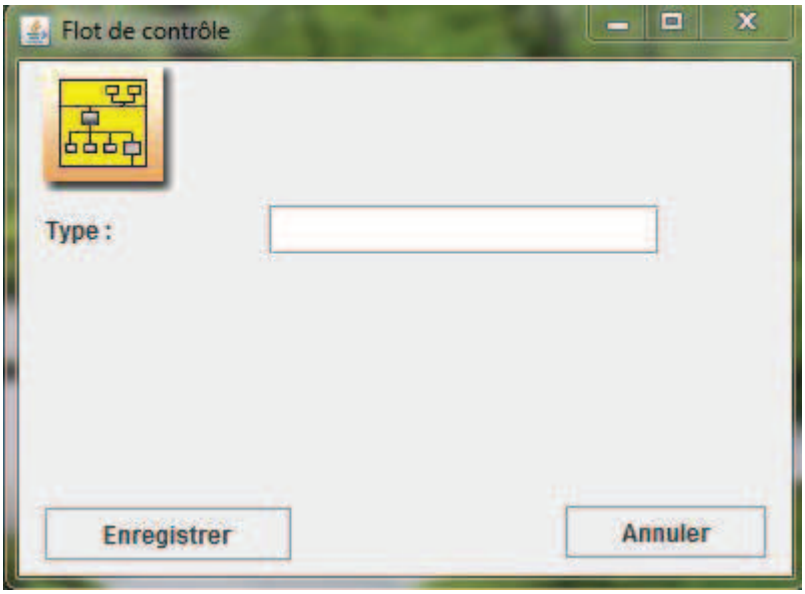


Figure 52. Spécification du flot de contrôle dans IASASTUDIO

5.4.1 Vérification de l'exécution du flot de contrôle

Cette opération se fait par un clic droit sur le composant de contrôle (figure 53)

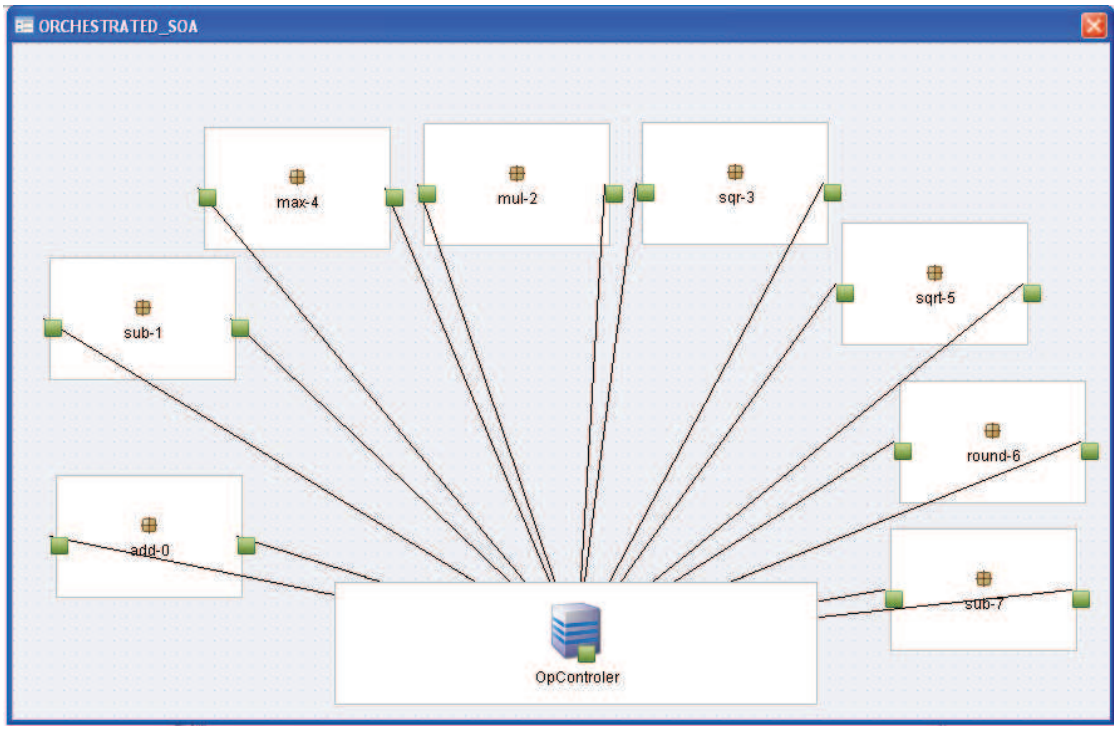


Figure 53. L'exécution du flux de contrôle dans IASASTUDIO

Le résultat d'exécution du composant Fcmp est présenté dans la figure 54 :

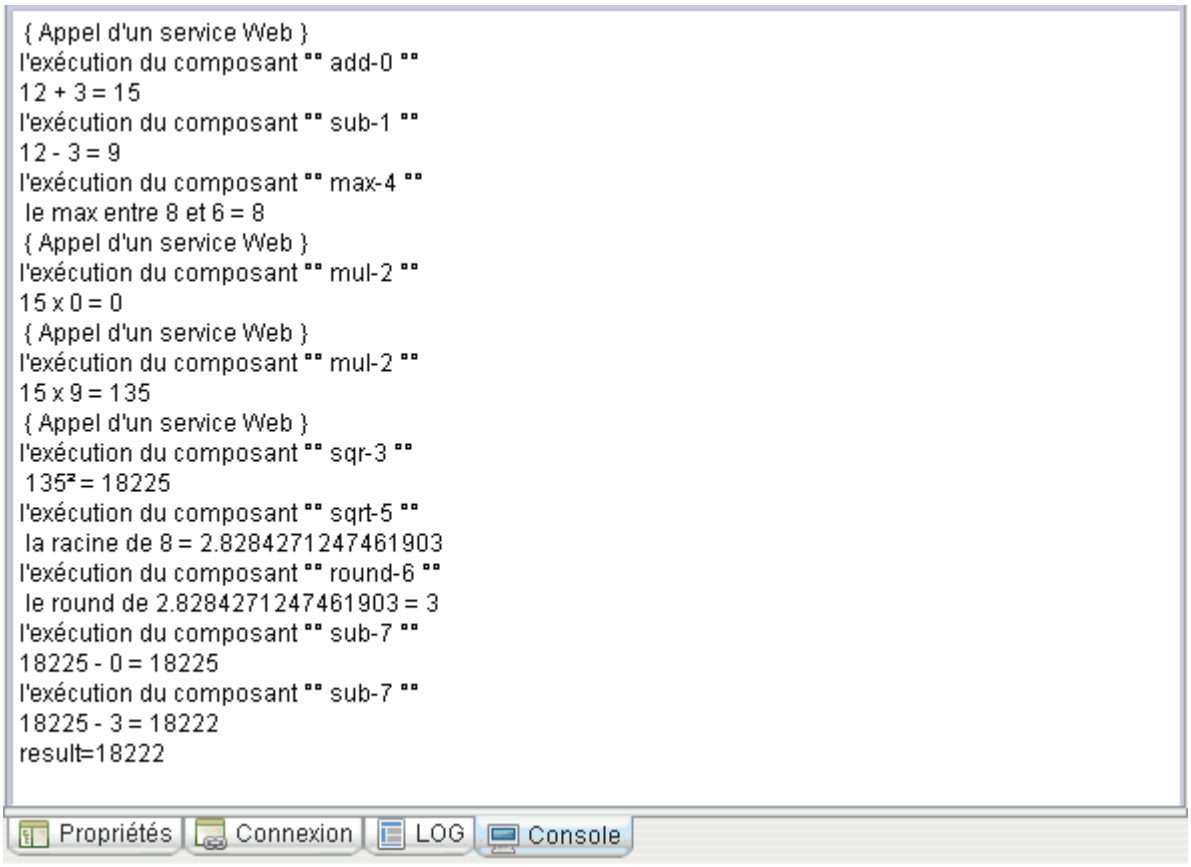


Figure 54. Exécution pas à pas de Fcmp dans IASASTUDIO

5.5 Production de la vue implémentation

Le composant réalisant le service Web est un simple objet Java qui consomme le service Web en question. La génération de ce dernier est assurée par l'enveloppe. Ainsi, le vrai service Web sera représenté par l'enveloppe. Prenons l'exemple du composant addition du composant FCmp représenté par le fichier WSDL suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:ns0="http://ws.apache.org/axis2"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://ws.apache.org/axis2">
   <wsdl:types>
      <xs:schema xmlns:ns="http://ws.apache.org/axis2"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://ws.apache.org/axis2">
```

```
         <xs:element name="add">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="param0"
type="xs:int"/>
                  <xs:element minOccurs="0" name="param1"
type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="addResponse">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="return"
type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:schema>
   </wsdl:types>
   <wsdl:message name="addRequest">
      <wsdl:part name="parameters" element="ns0:add"/>
   </wsdl:message>
   <wsdl:message name="addResponse">
      <wsdl:part name="parameters" element="ns0:addResponse"/>
   </wsdl:message>
   <wsdl:portType name="AdderServicePortType">
      <wsdl:operation name="add">
         <wsdl:input message="ns0:addRequest"
wsaw:Action="urn:add"/>
         <wsdl:output message="ns0:addResponse"
wsaw:Action="urn:addResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="AdderServiceSOAP11Binding"
type="ns0:AdderServicePortType">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
      <wsdl:operation name="add">
         <soap:operation soapAction="urn:add" style="document"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:binding name="AdderServiceSOAP12Binding"
type="ns0:AdderServicePortType">
      <soap12:binding
transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
      <wsdl:operation name="add">
         <soap12:operation soapAction="urn:add" style="document"/>
         <wsdl:input>
            <soap12:body use="literal"/>
```

```
         </wsdl:input>
         <wsdl:output>
            <soap12:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:binding name="AdderServiceHttpBinding"
type="ns0:AdderServicePortType">
      <http:binding verb="POST"/>
      <wsdl:operation name="add">
         <http:operation location="AdderService/add"/>
         <wsdl:input>
            <mime:content type="text/xml" part="add"/>
         </wsdl:input>
         <wsdl:output>
            <mime:content type="text/xml" part="add"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="AdderService">
      <wsdl:port name="AdderServiceSOAP11port_http"
binding="ns0:AdderServiceSOAP11Binding">
         <soap:address
location="http://localhost:8080/ode/processes/AdderService"/>
      </wsdl:port>
      <wsdl:port name="AdderServiceSOAP12port_http"
binding="ns0:AdderServiceSOAP12Binding">
         <soap12:address
location="http://localhost:8080/ode/processes/AdderService"/>
      </wsdl:port>
      <wsdl:port name="AdderServiceHttpport"
binding="ns0:AdderServiceHttpBinding">
         <http:address
location="http://localhost:8080/ode/processes/AdderService"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

A partir de ce WSDL, l'enveloppe génère les classes stub, en exécutant WSDL2JAVA de Apache Axis2

Le stub généré à partir du fichier WSDL sera identifié par URI avec le nom de package associé et il sera placé dans le chemin PATH, l'enveloppe se charge après de coder d'appel au service Web :

```
    public static int  add(int a,int b) throws RemoteException
    {
        AdderServiceStub stub = new AdderServiceStub();
        AdderServiceStub.Add operation = new AdderServiceStub.Add();
        operation.setParam0(a);
        operation.setParam1(b);
        AdderServiceStub.AddResponse r = stub.add(operation);
        return (int)r.get_return();
```

```
    }
```

5.5.1 Les phases de la génération de la vue implémentation

La génération de la vue implémentation par IASASTUDIO se base sur le fichier x3ADL. Une fois appuyé sur le bouton exécuter, le contrôleur de la partie opérative jouera le rôle du chef d'orchestre applique les règles de transformation suivantes à partir du fichier x3ADL vers le langage Java :

1- retire une liste de composants du fichier x3ADL, où chaque composant est une instance de la classe composant.

   Chaque instance de la classe composant est identifiée par un nom unique, un mode de déploiement, une liste des composants qui le précèdent et une liste des composants qui le succèdent. Chaque composant possède deux ports, un port de données et un port de service. Le port est une instance de la classe port. La classe port a comme attributs, le nom du port qui l'identifie d'une manière unique, ainsi que son type.

```
package IASASTUDIO_LIB;
import java.util.ArrayList;
public class composant {
String nom;  // le nom_du_composant
String type; // ce champ indique si le composant est primitif ou composite
String mode_dpl; // cet attribut indique comment le composant est déployé (
Web service ou processus )
port port_de_service; // le port de service du composant
port port_de_donnes; // le port de données
ArrayList<composant> precedent = new ArrayList<composant>(); //liste des
composants à exécuter avant ce composant
ArrayList<composant> suivant = new ArrayList<composant>();  // liste des
composants qui utilisent le résultat de l'exécution de ce composant
public composant(){}
}
```

Chaque instance de ports contient une liste de points d'accès

```
package IASASTUDIO_LIB;
import java.util.ArrayList;
public class port {
String nom; // nom_du_port
String type;//type_du_port
ArrayList<pt_acces> liste_des_acces_pt = new ArrayList<pt_acces>();//liste
des points d'accès qui forment le port
public port(){}
}
```

Les points d'accès sont des instances la classe point-d'accès. Chaque point est caractérisé par un identifiant unique qui est son nom, un type, un sens et une valeur.

```
package IASASTUDIO_LIB;
public class pt_acces {
String nom;  // nom_du_pt_d'accès
String type; // son type (intDataPoint,ServerActionPoint ..etc)
String sens; // le sens IN,OUT ou INOUT
int valeur;// la valeur stockée dans le point d'accès
public pt_acces(){}
}
```

Les valeurs des points d'accès dans IASASTUDIO sens IN sont des valeurs entière, si jamais le composant exécuté retourne un résultat réel, il faut connecter ce composant avec le composant Round afin d'avoir un résultat entier.

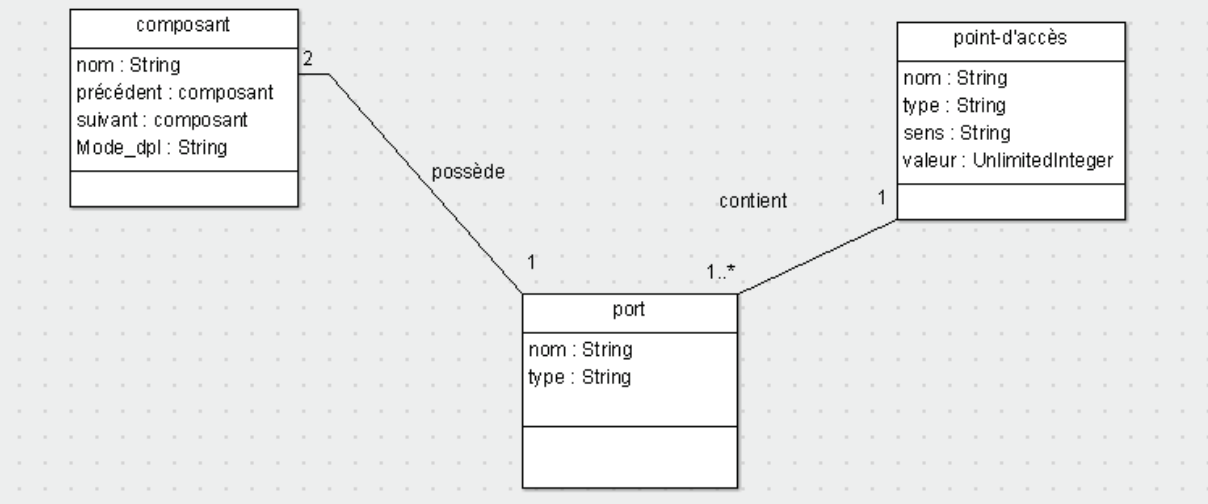


Figure 55. Diagramme de classe représentant un composant composite dans IASASTUDIO

2- Cherche les composants déclencheurs, un composant déclencheur est un composant qui possède un connecteur de délégation, ces derniers sont enregistrés dans la balise <DelegateConnectors>.

```
<DelegateConnectors>
 <map var="x" to-cmp="add-0" in="param0" />
 <map var="y" to-cmp="add-0" in="param1" />
 <map var="x" to-cmp="sub-1" in="param0" />
```

```
      <map var="y" to-cmp="sub-1" in="param1" />
      <map var="a" to-cmp="max-4" in="param0" />
      <map var="b" to-cmp="max-4" in="param1" />
     </DelegateConnectors>
```

3- retire le flux d'exécution et l'enregistre dans une liste, cette information est récupérée de la balise <DataFlow>du fichier x3ADL.
4- Exécute le code métier des composants déclencheurs, si le composant représente une fonction ordinaire, l'exécution fait appel à cette fonction. Par contre s'il s'agit d'un service Web, l'exécution fait appel à un service Web.
5- Exécute le reste des composants.

Ces étapes peuvent être présentées par l'Algorithme suivant :

```
Component current_cmp ;
List<Component> List_of_cmp;
int Resultat;

Lire la balise DataFlow
//Récupérer la liste des composants (remplir les champs, name,last,next)
    //
-Lire la balise Deployment
//faire une recherche par nom de composant afin de remplir le champ
deplyed_as

-Exécuter les composants déclencheurs
 //un composant déclencheur est un composant sans précédent

         Current_cmp= List_of_cmp.First ;
         While (List_of_cmp != null)
            {
               if(current_cmp.last = « »)
                 {
                      invoke(current_cmp);
                 }
               //passer aux suivant
               current.cmp = List_of_cmp.next;

            }

- Une fois terminée invoquer le reste des composants


         While (List_of_cmp! = null)
            {
               if(current_cmp.last != « »)
                 {
                      invoke(current_cmp);
                 }
               //passer aux suivant
               current.cmp = List_of_cmp.next;

            }
La fonction invoke :
void invoke (Component comp)
{
    if(comp.deployed_as = «web_service » )
```

```
            résultat = {appel du web service correspondant}

    Else // deployer comme une fonction ordinaire
            résultat = {simple appel de fonction}


    -passer le résultat de l'exécution au composant next
}
```

5.6 Etude comparative IASA VS BPEL :

Cette étude comparative nous montrerons comment réaliser en IASA et en BPEL le service Web composite réalisant la fonction [(x+y) * (x*y)] ². Cette composition utilise trois services web : Le premier se charge de réaliser une addition, le deuxième réalise la multiplication et le troisième réalise le carré

Il est important de noter que dans la plupart des rapports de recherche, communications, publications et thèses, les exemples cités pour montrer la faisabilité des diverses approches sont des exemples très simples. Souvent c'est l'exemple standard de réservation qui revient le plus souvent. De plus cet exemple est souvent considéré de manière assez simpliste. Dans notre cas nous avions utilisé une fonction mathématique qui pourrait être réellement concrétisée même dans un environnement local.

5.6.1 Composition du service Web avec BPEL

La réalisation d'une composition en BPEL nécessite la définition de trois fichiers : Le premier fichier, nommé FunctionProcess.bpel se charge de définir le processus d'orchestration. Le deuxième fichier nommé FunctionProcessService.wsdl contient la description du service décrit par le processus BPEL défini dans FunctionProcess.bpel. Le troisième fichier nommé deploy.xml représente le fichier de déploiement du service web composite. Ce fichier dépend fortement du moteur BPEL. Dans notre cas nous avons utilisé le moteur Apache ODE[5].

Ces trois fichiers seront enregistrés dans un dossier nommé «FunctionProcess », ce dernier doit être placé dans le dossier « processes » d'ODE.

[5] Apache ODE (Orchestration Director Engine) : un moteur open source qui exécute les processus métier écrient en langage BPEL4WS

Afin de montrer la complexité du processus de composition avec BPEL, nous présentons dans ce qui suit les éléments de définitions des trois fichiers cité auparavant.

- Définition du processus : le fichier FunctionProcess.bpel :

Dans le processus BPEL, lorsque les deux paramètres d'entrée sont reçus par le processus, les services seront invoqués selon le diagramme d'activité présenté dans la (figure 56).

$$\text{Function} = [(x+y)^{(2)} * (x*y)]^{2\,(3)}$$

(1) (1)

Les opérations (x + y) et (x * y) invoquées en parallèle. Ensuite, la multiplication, et enfin le carrés.

BPEL permet l'envoi et la réception d'informations entre les activités de processus qui peuvent être accomplies par les services web (figure 4.17), ces activités sont résumées dans le tableau suivant

Activité	Rôle
ReceiveRequest	Cette activité attend la requête pour déclancher le processus BPEL.
Assign1	Après ReceiveRequest, le message reçu est stocké dans une variable qui doit être spécifiée dans l'activité ReceiveRequest. Cette variable contient deux entiers. Selon la fonction mathématique que nous utilisons, ces deux entiers devraient être transmis séparément pour les deux activités InvokeAdderService et InvokeMultiplierService1. afin d'invoquer les services Web correspondants.
Flow1	Cette activité regroupe InvokeAdderService et InvokeMultiplierService1. Ceci signifie que ces deux activités se produisent en parallèle.
InvokeAdderService	Cette activité invoque le service Web AdderService.
InvokeMultiplierService1	Cette activité invoque le service Web MultiplierService

Assign2	Après l'exécution des activités InvokeAdderService et InvokeMultiplierService1.les deux valeurs retournées seront affectés comme des variables d'entrées d'InvokeMultiplierService2
InvokeMultiplierService2	invoque le service MultiplierService Web service
Assign3	affecte le résultat d'InvokeMultiplierService2 à la variable d'entrée d'InvokeSquareService.
InvokeSquareService	Invoque le service Web SquareService
Assign4	affecte le résultat d'InvokeSquareService à Reply1.
Reply1	envoie le résultat d'InvokeSquareService au client via FunctionProcessService.

Tableau 9 - Les activités BPEL de FUNCTION

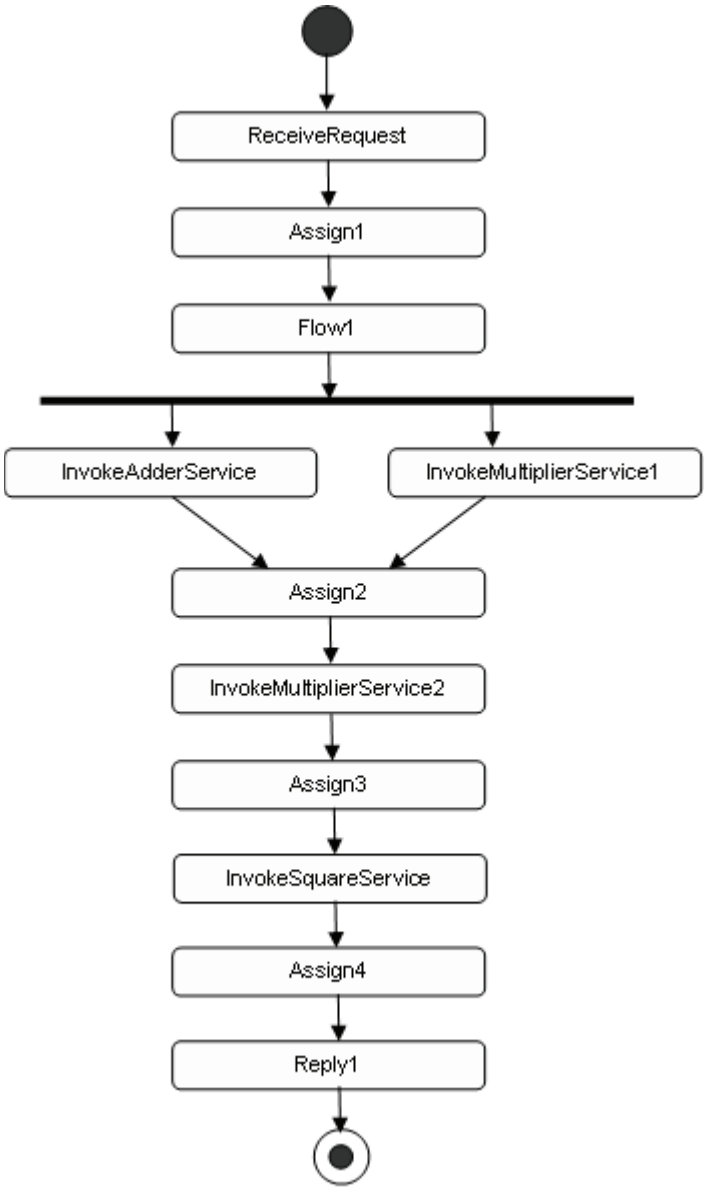


Figure 56. Diagramme d'activités de FunctionProcess

Il faut suivre rigoureusement la syntaxe BPEL décrite dans le chapitre 2, pour écrire le fichier FunctionProcess.bpel (voir Annexe A).

- FunctionProcessService.wsdl :

Ce fichier sert à décrire le service web représenté par la formule FUNCTION, ce nouveau service web est décrit dans le fichier FunctionProcessService.wsdl (Annexe A)

- Descripteur de déploiement

Après la création du processus BPEL la dernière étape consiste à déployer ce dernier dans un moteur BPEL. Cette phase dépend du moteur utilisé. Dans notre cas nous avons utilisé Apache ODE. Pour déployer notre processus sur ODE, nous avons besoin de créer un descripteur de déploiement deploy.xml. Sans deploy.xml, ODE ne peut pas comprendre le processus BPEL (voir Annexe A)Une fois ces trois fichiers sont prêts, nous pouvons tester avec SoapUI (figure 57)

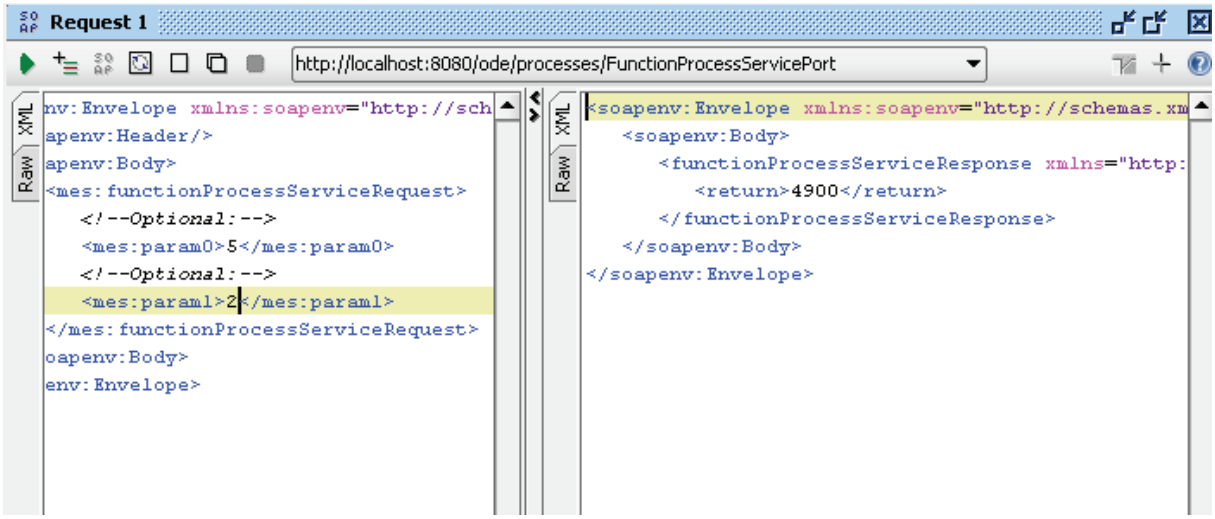


Figure 57. Exécution de FunctionProcessService dans SoapUI

5.6.2 Composition de FUNCTION avec IASASTUDIO

Pour traiter cet exemple avec IASASTUDIO il suffit d'importer les composants nécessaires déployer en tant que services web, de la même manière comme nous l'avons fait avec la formule f (figure 58).

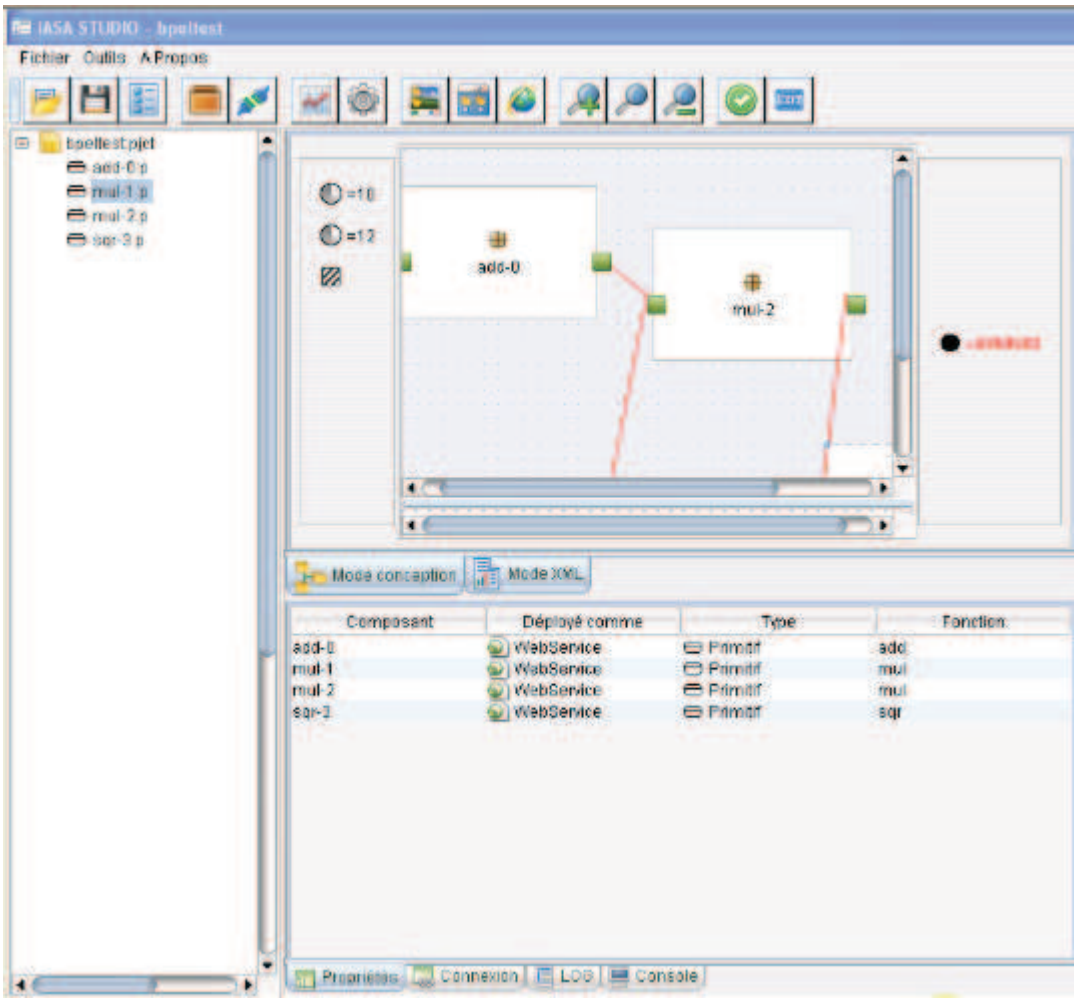


Figure 58. Exécution de FUNCTION dans IASASTUDIO

Avec BPEL la composition est très délicate, il faut savoir écrire le processus BPEL, le descripteur de service web et bien évidement le fichier de déploiement. Si le concepteur commis une erreur, la composition ne peut pas fonctionner correctement, nous avons testé aussi le plugin Eclipse BPEL DESIGNER dédié à la simplification de l'écriture des processus BPEL, mais malheureusement il faut que l'utilisateur intervient avec des bouts de code dans chaque activités BPEL, de même il ne garantie aucune cohérence syntaxique.

Par contre IASASTUDIO, se charge de l'écriture automatique du fichier x3ADL, la composition est très proche du modèle mental, ce qui engendre une composition efficace sans difficultés.

5.7 Utilisation des fichiers x3ADL en dehors d'IASASTUDIO dans des programmes java :

IASASTUDIO ne génère pas des descripteurs WSDL, cette décision était prise parce qu'IASASTUDIO peut aussi générer des composants hétérogènes qui ne sont pas des services Web.

Comme le fichier x3ADL est le seule fichier représentant la composition dans notre approche, nous avons développé une bibliothèque pour la lecture des fichiers x3ADL, cette bibliothèque est un fichier jar nommé IASACL.
IASACL implémente un composant contrôleur qui jouera le rôle du contrôleur de la partie opérative donc orchestrateur.

Nous allons découvrir à travers cet exemple comment utiliser un fichier x3ADL dans une application totalement indépendante, nous allons prendre l'exemple du fichier MyProject.xml ce dernier exécute la formule $V=(x+y)^2$.

Dans le projet Java il faut importer ces trois bibliothèques : La bibliothèque jdom.jar, la bibliothèque d'Apache Axis et notre bibliothèque IASACL.jar. L'utilisateur doit ajouter la ligne suivante à la classe qui requiert la composition,

```
Consomer.consume(PATH) ;
```

dans notre cas cette classe s'appelle AppelWebService
Dans l'environnement Eclipse nous avons créé le projet TestMyProject(Figure 59), la classe «AppelWebservice » fait appel au MyProject.xml

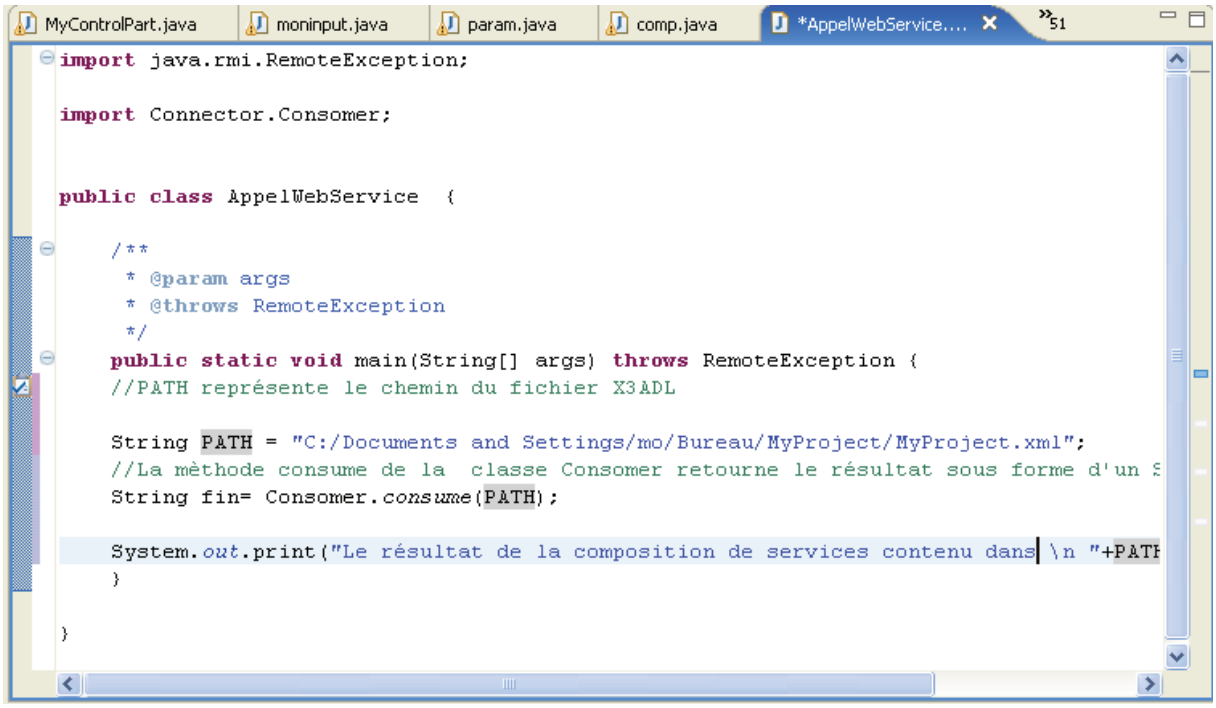


Figure 59. Classe AppelWebService dans Eclipse

Après l'exécution il faut que l'utilisateur donne les valeurs initiales

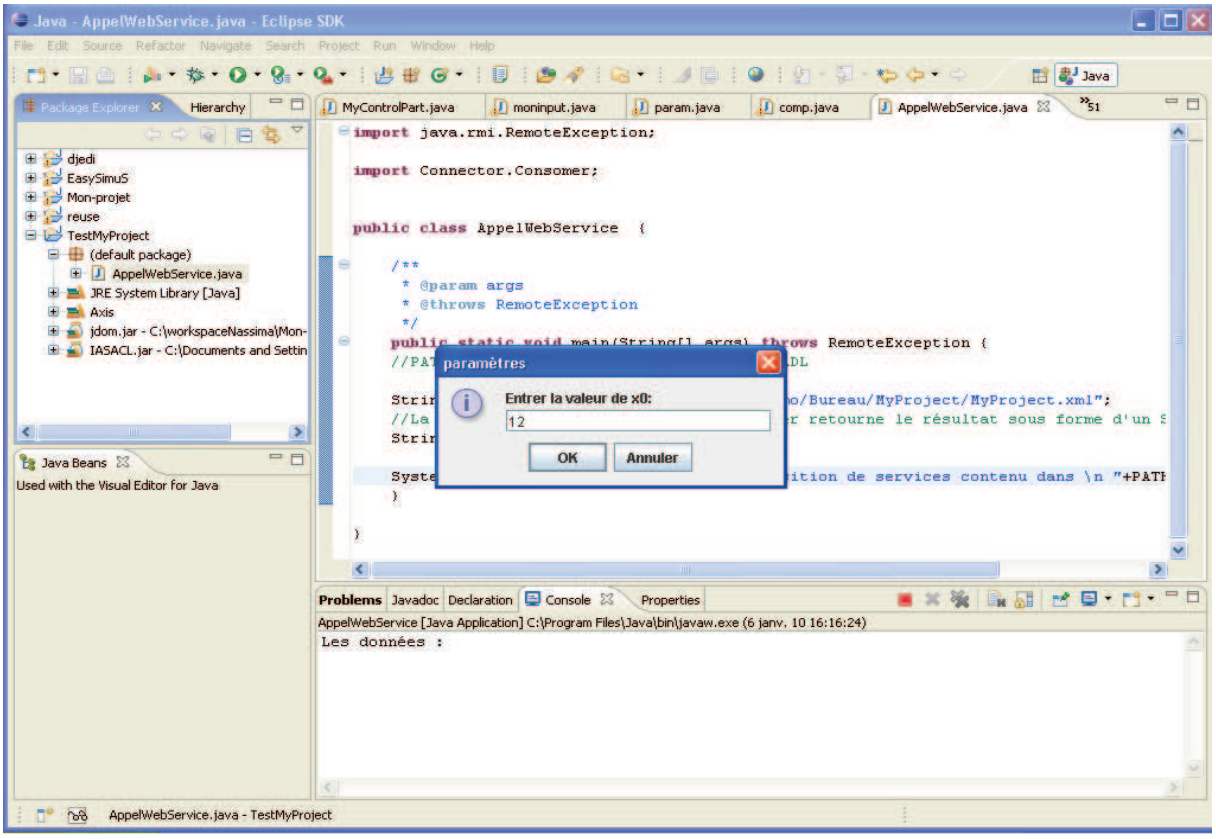


Figure 60. Exécution de la classe AppelWebService dans Eclipse

Le résultat de l'exécution est présenté dans la figure suivante :

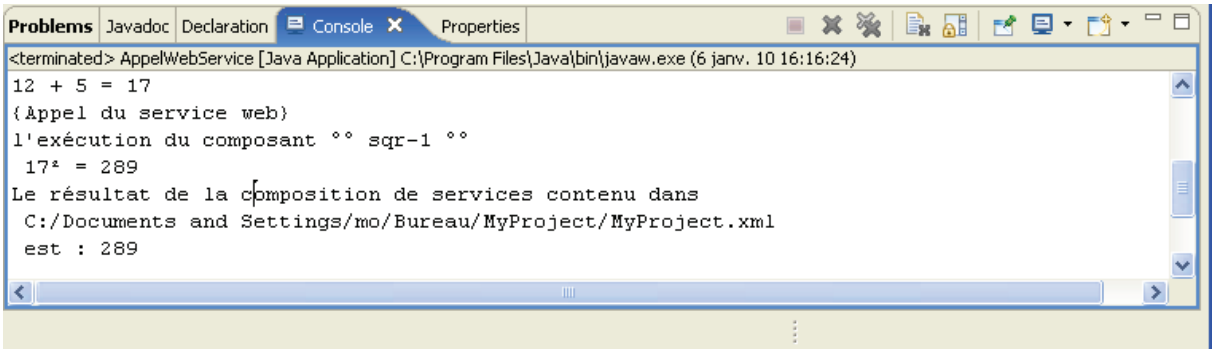


Figure 61. Trace d'exécution AppelWebService dans Eclipse

5.8 Conclusion

Dans ce chapitre nous avons présenté l'environnement IASASTUDIO que nous avons développé pour la composition de services web selon l'approche intégrée.

Nous avons vu à travers un exemple comment notre environnement simplifie la composition au concepteur, ce dernier s'occupe juste d'interconnecter les composants adéquatement et non pas des détails techniques.

Le résultat de la composition peut être un service Web comme il peut être un composant hétérogène. De plus IASASTUDIO génère automatiquement un seul fichier lors de la composition, chose qui n'est pas assuré par BPEL.

CONCLUSION

Le présent mémoire a été le fruit de plusieurs efforts de recherches sur les services Web et l'architecture logicielle dans le but de proposer une solution à la problématique de composition de services Web.

Nous avons présenté la problématique de composition de service Web, les langages de composition existants ainsi que quelques travaux de recherches liés au problème de la composition de services web. Après l'étude théorique que nous avons mené, nous avons constaté, que la composition de services Web avec les langages de composition actuels est une tâche très délicate. Le nombre de concepts, mécanisme et artefact à maitriser et à mettre en œuvre est très important et très décourageant. C'est ce qui explique d'ailleurs le faible intérêt pour les services Web dans le développement de logiciels centré sur le Web.

Afin d'abstraire les divers mécanismes de mise en œuvre des services web et d'en libérer les développeurs, nous avons proposé dans ce mémoire de porter la résolution du problème de composition des web services à un niveau d'abstraction élevé représenté par l'espace Architecture Logicielle. Cette approche n'aurait pas été possible si ce n'est l'apparition dans le domaine du génie logicielle de la discipline à Architecture logicielle. Celle-ci a pour objectif majeur de promouvoir la conception de système logiciel par assemblage de composants logiciels. L'Architecture Logicielle est un domaine dans lequel pourrait s'inscrire aisément les Architectures Orientées Services.

Les modèles de composants, de connecteurs et les langages de descriptions d'architecture logicielles (ADL) représentent les éléments fondamentaux des approches architectures logicielles. Plusieurs modèles de composants, de connecteurs et d'ADL ont été proposés. En général, les modèles et langages proposés traitent souvent un domaine particulier ou traitent un problème particulier au niveau architecture. Parmi ces approches, se distingue l'approche IASA (Integrated Approach to Software Architecture). Cette approche accentue le niveau d'abstraction en permettant des spécification qui sont totalement indépendante de

tout mécanisme logiciel. Le but de cette approche est la saisie directe des modèles mentaux d'architecture logicielle et leur formalisation.

L'approche IASA n'a pas été seulement choisi à cause de sa capacité de spécifier des Architecture logicielle à un haut niveau d'abstraction indépendant des mécanismes logiciels, mais pour son modèle de composant qui s'adapte efficacement pour supporter les deux grandes approches de composition de services Web à savoir l'approche par Orchestration et l'approche par Chorégraphie.

Pour le support d'une approche par orchestration, l'approche IASA utilise un modèle de composant composite dans lequel le contrôle globale de l'architecture est supporté par un composant explicite, appelé contrôleur. Ceci n'est pas le cas dans les autres approches d'Architecture logicielle qui font abstraction de ce composant. De plus l'approche IASA dispose du cas de déploiement CONTROLLED_FLOW dans lequel le contrôle de flux est forcé à suivre celui défini par le contrôleur.

Pour le support de la chorégraphie, l'approche IASA dispose de deux éléments fondamentaux : Le concept d'enveloppe qui est utilisé pour instancier les composant et les cas de déploiement CONCURRENT_FLOW et FREE_FLOW.

Pour la validation de notre approche, nous n'avions pas utilisé toute la puissance de l'approche IASA, notamment l'orienté aspects et la spécification libre de topologie de composant, ni l'exploitation du concept d'enveloppe dans la gestion du flux (cas de déploiement CONCURRENT_FLOW et FREE_FLOW). De plus vu la complexité des approches de composition de services Web, nous nous sommes limités dans notre travail à la composition par Orchestration de service Web.

Notre contribution dans ce travail peut se résumer par les points suivants :

- La définition d'un modèle de composant spécifique pour le support des Services Web.
- L'introduction dans IASA de trois nouveaux cas de déploiement : le cas WS (Web Service), le cas ORCHESTRATED_SOA qui est un cas spécifique de CONTROLLED_FLOW, et le cas CONCURRENT_SOA C qui est un cas spécifique de CONCCURENT_FLOW.

- La contribution à la définition du langage 3ADL et la définition complète de x3ADL qui est la forme XML de 3ADL.
- Un processus de transformation d'une spécification abstraite décrite en x3ADL vers une vue d'implémentation représenté par la technologie java pour les services web (Java-WS).

Cette contribution a été validé par

- La mise en place d'un outil de spécification graphique d'architecture logicielle. Cet outil permet de générer en temps réel la description x3ADL. Nous avons appelé IASASTUDIO cet outil.
- La réalisation d'exemples simples avec IASASTUDIO. A travers ces exemples, nous montrons comment il est facile de réaliser des compositions pures de services web ou des compositions hétérogènes.

Le travail que nous venons de réaliser a en fait ouvert les grandes portes vers un nombre important de perspectives dans le domaine de la composition de services Web selon l'approche architecture logicielle. Les perspectives qui semblent très intéressantes peuvent se résumer aux points suivants :

- L'introduction de l'approche de composition par chorégraphie. Nous avons introduit comme prévision à cette perspective le cas de déploiement CONCCURENT_SOA. Les travaux qui seraient menés pour cette perspective concernent d'une part la transformation d'une architecture IASA concurrente vers une architecture ordinaire concurrente mettant en œuvre la chorégraphie. D'autre part il est nécessaire de définir les règles du processus de transformation d'une description X3ADL vers une vue implémentation.
- La mise en œuvre de l'approche de conception par aspects dans une composition de service web.

Enfin, nous notons que ce travail a été concrétisé par une communication accepté pour présentation au 3ème Symposium International sur les Services Web (WSs2010 Symposium, Dubai, U.A.E). La communication est intitulée: "Composing web services with the integtarated Approach to software architecture.

ANNEXE A

AdderService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="AdderService"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="http://AdderService.wsdl" xmlns:plink="http://docs.oasis-
open.org/wsbpel/2.0/plnktype" xmlns:ns0="http://AdderService.wsdl"
xmlns:ns2="http://ws.apache.org/axis2">
   <wsdl:types>
      <xs:schema xmlns:ns="http://ws.apache.org/axis2"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://ws.apache.org/axis2">
         <xs:element name="add">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="param0" type="xs:int"/>
                  <xs:element minOccurs="0" name="param1" type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="addResponse">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="return" type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:schema>
   </wsdl:types>
   <wsdl:message name="addRequest">
      <wsdl:part name="parameters" element="ns2:add"/>
   </wsdl:message>
   <wsdl:message name="addResponse">
      <wsdl:part name="parameters" element="ns2:addResponse"/>
   </wsdl:message>
   <wsdl:portType name="AdderServicePortType">
      <wsdl:operation name="add">
         <wsdl:input message="ns0:addRequest" wsaw:Action="urn:add"/>
         <wsdl:output message="ns0:addResponse"
wsaw:Action="urn:addResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="AdderServiceSOAP11Binding"
type="ns0:AdderServicePortType">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
      <wsdl:operation name="add">
         <soap:operation soapAction="urn:add" style="document"/>
         <wsdl:input>
```

```
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:binding name="AdderServiceSOAP12Binding"
type="ns0:AdderServicePortType">
      <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
      <wsdl:operation name="add">
         <soap12:operation soapAction="urn:add" style="document"/>
         <wsdl:input>
            <soap12:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap12:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:binding name="AdderServiceHttpBinding"
type="ns0:AdderServicePortType">
      <http:binding verb="POST"/>
      <wsdl:operation name="add">
         <http:operation location="AdderService/add"/>
         <wsdl:input>
            <mime:content type="text/xml" part="add"/>
         </wsdl:input>
         <wsdl:output>
            <mime:content type="text/xml" part="add"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="AdderService">
      <wsdl:port name="AdderServiceSOAP11port_http"
binding="ns0:AdderServiceSOAP11Binding">
         <soap:address
location="http://localhost:8080/ode/processes/AdderService"/>
      </wsdl:port>
      <wsdl:port name="AdderServiceSOAP12port_http"
binding="ns0:AdderServiceSOAP12Binding">
         <soap12:address
location="http://localhost:8080/ode/processes/AdderService"/>
      </wsdl:port>
      <wsdl:port name="AdderServiceHttpport"
binding="ns0:AdderServiceHttpBinding">
         <http:address
location="http://localhost:8080/ode/processes/AdderService"/>
      </wsdl:port>
   </wsdl:service>
    <plink:partnerLinkType name="AdderServicePartnerlinkType">
        <plink:role name="adderRole" portType="ns0:AdderServicePortType"/>
    </plink:partnerLinkType>
</wsdl:definitions>
```

MultiplierService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="MultiplierService"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="http://MultiplierService.wsdl"
xmlns:plink="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
xmlns:ns0="http://MultiplierService.wsdl"
xmlns:ns2="http://ws.apache.org/axis2">
   <wsdl:types>
      <xs:schema xmlns:ns="http://ws.apache.org/axis2"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://ws.apache.org/axis2">
         <xs:element name="multiply">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="param0" type="xs:int"/>
                  <xs:element minOccurs="0" name="param1" type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="multiplyResponse">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="return" type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:schema>
   </wsdl:types>
   <wsdl:message name="multiplyRequest">
      <wsdl:part name="parameters" element="ns2:multiply"/>
   </wsdl:message>
   <wsdl:message name="multiplyResponse">
      <wsdl:part name="parameters" element="ns2:multiplyResponse"/>
   </wsdl:message>
   <wsdl:portType name="MultiplierServicePortType">
      <wsdl:operation name="multiply">
         <wsdl:input message="ns0:multiplyRequest"
wsaw:Action="urn:multiply"/>
         <wsdl:output message="ns0:multiplyResponse"
wsaw:Action="urn:multiplyResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="MultiplierServiceSOAP11Binding"
type="ns0:MultiplierServicePortType">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
      <wsdl:operation name="multiply">
         <soap:operation soapAction="urn:multiply" style="document"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
```

```
   <wsdl:binding name="MultiplierServiceSOAP12Binding"
type="ns0:MultiplierServicePortType">
      <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
      <wsdl:operation name="multiply">
         <soap12:operation soapAction="urn:multiply" style="document"/>
         <wsdl:input>
            <soap12:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap12:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:binding name="MultiplierServiceHttpBinding"
type="ns0:MultiplierServicePortType">
      <http:binding verb="POST"/>
      <wsdl:operation name="multiply">
         <http:operation location="MultiplierService/multiply"/>
         <wsdl:input>
            <mime:content type="text/xml" part="multiply"/>
         </wsdl:input>
         <wsdl:output>
            <mime:content type="text/xml" part="multiply"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="MultiplierService">
      <wsdl:port name="MultiplierServiceSOAP11port_http"
binding="ns0:MultiplierServiceSOAP11Binding">
         <soap:address
location="http://localhost:8080/ode/processes/MultiplierService"/>
      </wsdl:port>
      <wsdl:port name="MultiplierServiceSOAP12port_http"
binding="ns0:MultiplierServiceSOAP12Binding">
         <soap12:address
location="http://localhost:8080/ode/processes/MultiplierService"/>
      </wsdl:port>
      <wsdl:port name="MultiplierServiceHttpport"
binding="ns0:MultiplierServiceHttpBinding">
         <http:address
location="http://localhost:8080/ode/processes/MultiplierService"/>
      </wsdl:port>
   </wsdl:service>
    <plink:partnerLinkType name="MultiplierServicePartnerlinkType">
        <plink:role name="multiplierRole"
portType="ns0:MultiplierServicePortType"/>
    </plink:partnerLinkType>
</wsdl:definitions>
```

SquareService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="SquareService"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:ns1="http://org.apache.axis2/xsd"
```

```
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="http://SquareService.wsdl" xmlns:plink="http://docs.oasis-
open.org/wsbpel/2.0/plnktype" xmlns:ns0="http://SquareService.wsdl"
xmlns:ns2="http://ws.apache.org/axis2">
   <wsdl:types>
      <xs:schema xmlns:ns="http://ws.apache.org/axis2"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://ws.apache.org/axis2">
         <xs:element name="square">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="param0" type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="squareResponse">
            <xs:complexType>
               <xs:sequence>
                  <xs:element minOccurs="0" name="return" type="xs:int"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:schema>
   </wsdl:types>
   <wsdl:message name="squareRequest">
      <wsdl:part name="parameters" element="ns2:square"/>
   </wsdl:message>
   <wsdl:message name="squareResponse">
      <wsdl:part name="parameters" element="ns2:squareResponse"/>
   </wsdl:message>
   <wsdl:portType name="SquareServicePortType">
      <wsdl:operation name="square">
         <wsdl:input message="ns0:squareRequest" wsaw:Action="urn:square"/>
         <wsdl:output message="ns0:squareResponse"
wsaw:Action="urn:squareResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="SquareServiceSOAP11Binding"
type="ns0:SquareServicePortType">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
      <wsdl:operation name="square">
         <soap:operation soapAction="urn:square" style="document"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:binding name="SquareServiceSOAP12Binding"
type="ns0:SquareServicePortType">
      <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
      <wsdl:operation name="square">
         <soap12:operation soapAction="urn:square" style="document"/>
         <wsdl:input>
            <soap12:body use="literal"/>
```

```
         </wsdl:input>
         <wsdl:output>
            <soap12:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:binding name="SquareServiceHttpBinding"
type="ns0:SquareServicePortType">
      <http:binding verb="POST"/>
      <wsdl:operation name="square">
         <http:operation location="SquareService/square"/>
         <wsdl:input>
            <mime:content type="text/xml" part="square"/>
         </wsdl:input>
         <wsdl:output>
            <mime:content type="text/xml" part="square"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="SquareService">
      <wsdl:port name="SquareServiceSOAP11port_http"
binding="ns0:SquareServiceSOAP11Binding">
         <soap:address
location="http://localhost:8080/ode/processes/SquareService"/>
      </wsdl:port>
      <wsdl:port name="SquareServiceSOAP12port_http"
binding="ns0:SquareServiceSOAP12Binding">
         <soap12:address
location="http://localhost:8080/ode/processes/SquareService"/>
      </wsdl:port>
      <wsdl:port name="SquareServiceHttpport"
binding="ns0:SquareServiceHttpBinding">
         <http:address
location="http://localhost:8080/ode/processes/SquareService"/>
      </wsdl:port>
   </wsdl:service>
    <plink:partnerLinkType name="SquarePartnerlinkType">
        <plink:role name="squareRole"
portType="ns0:SquareServicePortType"/>
    </plink:partnerLinkType>
</wsdl:definitions>
```

FunctionProcessService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ControllerService"
targetNamespace="http://ControllerService.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://ControllerService.wsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:msgs="http://www.example.org/messages">
    <wsdl:types>
      <xsd:schema xmlns:ns="http://ws.apache.org/axis2"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://www.example.org/messages">
         <xsd:element name="ControllerServiceRequest">
            <xsd:complexType>
```

```
               <xsd:sequence>
                  <xsd:element minOccurs="0" name="param0" type="xsd:int"/>
                  <xsd:element minOccurs="0" name="param1" type="xsd:int"/>
                  <!--<xsd:element minOccurs="0" name="param2"
type="xsd:int"/>-->
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="ControllerServiceResponse">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element minOccurs="0" name="return" type="xsd:int"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:schema>
   </wsdl:types>
    <message name="ControllerServiceOperationRequest">
        <part name="ControllerServiceRequest"
element="msgs:ControllerServiceRequest"/>
    </message>
    <message name="ControllerServiceOperationResponse">
        <part name="ControllerServiceResponse"
element="msgs:ControllerServiceResponse"/>
    </message>
    <portType name="ControllerServicePortType">
        <operation name="ControllerServiceOperation">
            <input name="input1"
message="tns:ControllerServiceOperationRequest"/>
            <output name="output1"
message="tns:ControllerServiceOperationResponse"/>
        </operation>
    </portType>
    <binding name="ControllerServiceBinding"
type="tns:ControllerServicePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="ControllerServiceOperation">
            <soap:operation/>
            <input name="input1">
                <soap:body use="literal"/>
            </input>
            <output name="output1">
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="ControllerServiceService">
        <port name="ControllerServicePort"
binding="tns:ControllerServiceBinding">
            <soap:address
location="http://localhost:8080/axis2/ControllerServicePort"/>
        </port>
    </service>

</definitions>
```

FunctionProcessService.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="FunctionProcess"
    targetNamespace="http://FunctionProcess.bpel"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Tr
ace"

xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/E
ditor"
    xmlns:tns="http://FunctionProcess.bpel"
xmlns:ns0="http://AdderService.wsdl"
xmlns:ns1="http://MultiplierService.wsdl"
xmlns:ns2="http://SquareService.wsdl"
xmlns:ns3="http://www.example.org/messages"
xmlns:ns4="http://ws.apache.org/axis2">
    <import namespace="http://FunctionProcessService.wsdl"
location="FunctionProcessService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://AdderService.wsdl"
location="AdderService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://MultiplierService.wsdl"
location="MultiplierService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://SquareService.wsdl"
location="SquareService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <partnerLinks>
        <partnerLink name="SquarePartnerLink"
xmlns:ns2="http://SquareService.wsdl"
partnerLinkType="ns2:SquareServicePartnerlinkType"
partnerRole="squareRole"/>
        <partnerLink name="MultiplierPartnerLink"
xmlns:ns1="http://MultiplierService.wsdl"
partnerLinkType="ns1:MultiplierServicePartnerlinkType"
partnerRole="multiplierRole"/>
        <partnerLink name="AdderPartnerLink"
xmlns:ns0="http://AdderService.wsdl"
partnerLinkType="ns0:AdderServicePartnerlinkType" partnerRole="adderRole"/>
        <partnerLink name="FunctionProcessPartnerLink"
xmlns:tns="http://FunctionProcessService.wsdl"
partnerLinkType="tns:FunctionProcessService"
myRole="FunctionProcessServicePortTypeRole"/>
    </partnerLinks>
    <variables>
        <variable name="FunctionProcessServiceOperationOutput"
xmlns:tns="http://FunctionProcessService.wsdl"
messageType="tns:FunctionProcessServiceOperationResponse"/>
        <variable name="MultiplyOutput1"
messageType="ns1:multiplyResponse"/>
        <variable name="MultiplyInput1" messageType="ns1:multiplyRequest"/>
        <variable name="FunctionProcessServiceOperationInput"
xmlns:tns="http://FunctionProcessService.wsdl"
messageType="tns:FunctionProcessServiceOperationRequest"/>
```

```
        <variable name="MultiplyOutput"
xmlns:ns1="http://MultiplierService.wsdl"
messageType="ns1:multiplyResponse"/>
        <variable name="MultiplyInput"
xmlns:ns1="http://MultiplierService.wsdl"
messageType="ns1:multiplyRequest"/>
        <variable name="SquareOutput" xmlns:ns2="http://SquareService.wsdl"
messageType="ns2:squareResponse"/>
        <variable name="SquareInput" xmlns:ns2="http://SquareService.wsdl"
messageType="ns2:squareRequest"/>
        <variable name="AddOutput" xmlns:ns0="http://AdderService.wsdl"
messageType="ns0:addResponse"/>
        <variable name="AddInput" xmlns:ns0="http://AdderService.wsdl"
messageType="ns0:addRequest"/>
    </variables>
    <sequence>
        <receive name="ReceiveRequest" createInstance="yes"
partnerLink="FunctionProcessPartnerLink"
operation="FunctionProcessServiceOperation"
xmlns:tns="http://FunctionProcessService.wsdl"
portType="tns:FunctionProcessServicePortType"
variable="FunctionProcessServiceOperationInput"/>
        <assign name="Assign1">
          <copy>
                <from>
                    <literal xml:space="preserve">
                  <tns:add xmlns:tns="http://ws.apache.org/axis2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                        <tns:param0></tns:param0>
                        <tns:param1></tns:param1>
                  </tns:add>
                </literal>
                </from>
                <to part="parameters" variable="AddInput">
                </to>
            </copy>
          <copy>
                <from>
                    <literal xml:space="preserve">
                  <tns:multiply xmlns:tns="http://ws.apache.org/axis2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                        <tns:param0></tns:param0>
                        <tns:param1></tns:param1>
                  </tns:multiply>
                </literal>
                </from>
                <to part="parameters" variable="MultiplyInput">
                </to>
            </copy>

            <copy>

<from>$FunctionProcessServiceOperationInput.functionProcessServiceRequest/n
s3:param0</from>
                <to>$AddInput.parameters/ns4:param0</to>
            </copy>
            <copy>

<from>$FunctionProcessServiceOperationInput.functionProcessServiceRequest/n
s3:param1</from>
                <to>$AddInput.parameters/ns4:param1</to>
```

```
            </copy>
            <copy>

<from>$FunctionProcessServiceOperationInput.functionProcessServiceRequest/n
s3:param0</from>
                <to>$MultiplyInput.parameters/ns4:param0</to>
            </copy>
            <copy>

<from>$FunctionProcessServiceOperationInput.functionProcessServiceRequest/n
s3:param1</from>
                <to>$MultiplyInput.parameters/ns4:param1</to>
            </copy>
        </assign>
        <flow name="Flow1">
            <invoke name="InvokeAdderService"
partnerLink="AdderPartnerLink" operation="add"
portType="ns0:AdderServicePortType" inputVariable="AddInput"
outputVariable="AddOutput"/>
            <invoke name="InvokeMultiplierService1"
partnerLink="MultiplierPartnerLink" operation="multiply"
portType="ns1:MultiplierServicePortType" inputVariable="MultiplyInput"
outputVariable="MultiplyOutput"/>
        </flow>
        <assign name="Assign2">
          <copy>
                <from>
                    <literal xml:space="preserve">
                  <tns:multiply xmlns:tns="http://ws.apache.org/axis2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                        <tns:param0></tns:param0>
                        <tns:param1></tns:param1>
                  </tns:multiply>
                </literal>
                </from>
                <to part="parameters" variable="MultiplyInput1">
                </to>
            </copy>
            <copy>
                <from>$AddOutput.parameters/ns4:return</from>
                <to>$MultiplyInput1.parameters/ns4:param0</to>
            </copy>
            <copy>
                <from>$MultiplyOutput.parameters/ns4:return</from>
                <to>$MultiplyInput1.parameters/ns4:param1</to>
            </copy>
        </assign>
        <invoke name="InvokeMultiplierService2"
partnerLink="MultiplierPartnerLink" operation="multiply"
portType="ns1:MultiplierServicePortType" inputVariable="MultiplyInput1"
outputVariable="MultiplyOutput1"/>
        <assign name="Assign3">
            <copy>
                <from>
                    <literal xml:space="preserve">
                  <tns:square xmlns:tns="http://ws.apache.org/axis2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                        <tns:param0></tns:param0>
                  </tns:square>
                </literal>
                </from>
```

```
                <to part="parameters" variable="SquareInput">
                </to>
            </copy>
          <copy>
                <from>$MultiplyOutput1.parameters/ns4:return</from>
                <to>$SquareInput.parameters/ns4:param0</to>
            </copy>
        </assign>
        <invoke name="InvokeSquareService" partnerLink="SquarePartnerLink"
operation="square" xmlns:ns2="http://SquareService.wsdl"
portType="ns2:SquareServicePortType" inputVariable="SquareInput"
outputVariable="SquareOutput"/>
        <assign name="Assign4">
            <copy>
                <from>
                    <literal xml:space="preserve">
                  <tns:functionProcessServiceResponse
xmlns:tns="http://www.example.org/messages"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                        <tns:return></tns:return>
                  </tns:functionProcessServiceResponse>
                </literal>
                </from>
                <to part="functionProcessServiceResponse"
variable="FunctionProcessServiceOperationOutput">
                </to>
            </copy>
          <copy>
                <from>$SquareOutput.parameters/ns4:return</from>

<to>$FunctionProcessServiceOperationOutput.functionProcessServiceResponse/n
s3:return</to>
            </copy>
        </assign>
        <reply name="Reply1" partnerLink="FunctionProcessPartnerLink"
operation="FunctionProcessServiceOperation"
xmlns:tns="http://FunctionProcessService.wsdl"
portType="tns:FunctionProcessServicePortType"
variable="FunctionProcessServiceOperationOutput"/>
    </sequence>
</process>
```

Deploy.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
xmlns:FunctionProcess.bpel="http://FunctionProcess.bpel"
xmlns:FunctionProcessService.wsdl="http://FunctionProcessService.wsdl"
xmlns:AdderService.wsdl="http://AdderService.wsdl"
xmlns:MultiplierService.wsdl="http://MultiplierService.wsdl"
xmlns:SquareService.wsdl="http://SquareService.wsdl">

  <process name="FunctionProcess.bpel:FunctionProcess">
    <process-events generate="all"/>
    <provide partnerLink="FunctionProcessPartnerLink">
      <service
name="FunctionProcessService.wsdl:FunctionProcessServiceService"
port="FunctionProcessServicePort"/>
    </provide>
    <invoke partnerLink="SquarePartnerLink">
```

```
      <service name="SquareService.wsdl:SquareService"
port="SquareServiceSOAP11port_http"/>
    </invoke>
    <invoke partnerLink="MultiplierPartnerLink">
      <service name="MultiplierService.wsdl:MultiplierService"
port="MultiplierServiceSOAP11port_http"/>
    </invoke>
    <invoke partnerLink="AdderPartnerLink">
      <service name="AdderService.wsdl:AdderService"
port="AdderServiceSOAP11port_http"/>
    </invoke>
  </process>

</deploy>
```

REFERENCES

1. OMG: Unified Modeling Language: Infrastructure, version 2.0, 3rd revised submission to OMG RFP ad/00-09-01, January 2003.
2. D. Garlan: Software Architecture. Encyclopedia of Software Engineering, John Wiley & Sons, Inc. 2001.
3. C Hofmeister, R. L. Nord, and D. Soni. Describing software architecture with UML. In Proceeding of the First Working IFIP Conference on Software,2001.
4. John D. Poole. "Model-Driven Architecture: Vision, Standards And Emerging Technologies". ECOOP 2001, Workshop on Metamodeling and Adaptive Object Models, April 2001.
5. N. R. Mehta, N. Medvidovic and S. Phadke, Towards Taxonomy of Software Connectors. ICSE '00, Limerick, Ireland, May 2000
6. D.Bennouar : UNE APPROCHE INTEGREE POUR L'ARCHITECTURE LOGICIELLE, thèse de doctorat, 2009, ESI (Algérie).
7. Saadi Abdelfetah : Un langage d'action pour la spécification et la validation du comportement d'une architecture logicielle, mémoire de magistère, Département d'Informatique, USDB, 2008 (Algérie)
8. Frédéric POURRAZ, Diapason: une approche formelle et centrée architecture pour la composition évolutive de services Web, thèse de doctorat, UNIVERSITE DE SAVOIE (France), Décembre 2007.
9. Michael PAPAZOGLOU, Web services : Principales and technology ; Prentice-Hall 2007.
10. Thomas Erl, "SOA : Principles of Service Design" pages : 608, , Prentice-Hall, 2007.
11. Michael N.Huhns, Munindar P. Singh, "Service-Oriented Computing:Key Concepts and Principles", IEEE Computer Society 2005.
12. Jamal Bentahar, Zakaria Maamar, Djamal Benslimane, Philippe Thiran, Subramanian Sattanathan." Agent-based Communities of Web Services: An Argumentation-driven ApproachService Oriented Computing and Applications, Springer", 2009.

13. Xebia (IT Architects SAS), Livre blanc : Comprendre et savoir utiliser un ESB dans le cadre d'une SOA, 2009
14. Helga DUARTE-AMAYA, Tcows Canevas pour la composition de services web avec propriétés transactionnelles, thèse de doctorat, UNIVERSITE JOSEPH FOURIER (France), Novembre 2007.
15. Stéphane Drapeau, Maxime Porhel, Etienne Juliot, «comprendre SCA » Livre blanc v1.0, OBEO, décembre 2009.
16. CHOUIREF Zahira, Un système de programmation fonctionnelle pour la composition de services Web, mémoire de magister, Université M'hamed BOUGARA de BOUMERDES, 2008.
17. F. Curbera, W. Nagy ET W. Weerawarana. Web services: Why and how? Conférence de Object-Oriented Programming, Systems, Languages and Application (OOPSLA), Workshop on Object-Oriented Web Services, 2001.
18. Jérôme Daniel. Extrait du livre "Services Web Concepts, techniques et outils" Edition vuibert Informatique 2003.
19. D. Fensel, C. Bussler, & A. Maedche, Semantic Web Enabled Web Services, Conférence internationale du web sémantique, Sardinia 2002, Italie, volume 2348, pages1-2 paper.
20. S. McIlraith, T.C. Son, & H. Zeng, Semantic Web Services, IEEE Intelligent System.Special Issue on the Semantic Web 2001.
21. Kenn Scribner , Kennard Scrinber, Mark C.Stiver ,"Understanding Soap: Simple Object Access Protocol", ams Indianapolis, IN, USA, 2001
22. Advancing open standards for the information society http://www.oasis-open.org/
23. Web service description language, Note W3C, Mars 2001 http://www.w3.org/TR/wsdl
24. Robert Richardsn, "Pro PHP XML and Web Services", page 751-780; Apress 2007
25. Yasmine CHARIF, Chorégraphie dynamique de services basée sur la coordination d'agents introspectifs, thèse de doctorat, UNIVERSITÉ PIERRE ET MARIE CURIE PARIS VI (France), Décembre 2007.
26. Philippe Laublet, Chantal Reynaud, Jean Charlet, Sur quelques aspects du Web sémantique, Journée du Web Sémantique et SHS.

27. ISO 15935 information technology : Open distributed processing reference model - quality of service. October 1998.
28. ISO 13236 : Information technology quality of service : Framework, 1998.

29. . Qos for web services – requirements and possible approaches, 2003.
30. Sonia JAMAL, Environnement de procédé extensible pour l'orchestration Application aux services web, thèse de doctorat, UNIVERSITE JOSEPH FOURIER Grenoble (France), Décembre 2005.
31. XML Modeling Language http://xml.coverpages.org/xlang.html
32. BizTalk Server http://www.microsoft.com/biztalk/en/us/default.aspx
33. Wil M.P. van der Aalst, Marlon Dumas Arthur, H.M. ter Hofstede, Wil M. P, Aalst Marlon Dumas, Arthur H. M, Petia Wohe. Pattern Based Analysis of BPML ; QUT Technical Report,FIT-TR-2002-05. Queensland University of Technology, Brisbane, 2002
34. INTALIO http://xml.coverpages.org/IntalioN3-v20.html
35. Frank Leymann, Web Service Flow Language; IBM Sotware Group; May 2001
36. Gu.Zhifeng,Juanzi li, JieTang, Bin Xu, "Verification of Web Service Conversations Specified in WSCL", Proceedings of the 31st Annual International Computer Software and Applications Conference, Pages: 432-437, 2007
37. A.Brogi, C.Canal, E.Pimentel,and A.Vallecillo, "Formalizing Web Service Choreographies", 10 December 2004, Pages 73-94 .Proceedings of the First International Workshop on Web Services and Formal Methods (WSFM 2004).
38. WSCI, Note W3C, Aout 2002 http://www.w3.org/TR/2002/NOTE-wsci-20020808/
39. E. Bertino, L.Martino, F. Paci, A. Squicciarini, Security for Web Services and Service-Oriented Architectures.Springer 2009.
40. Matjaz B. Juric, Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition, Packt Publishing 2007.
41. Feroze Mohammed; Lawrence Pravin, Oracle SOA Suite ; Sys-Con XML Journal, 2007.
42. W. A. Nagy, R. Khalaf et al. "Implementing BPEL4WS: The Architecture of a BPEL4WS Implementation", Journal of Concurrency and Computation: Practice and Experience, 2005.

43. Keen M., Cavell J., Hill S., Kee C.K., Neave W., Rumph B., Tran H., BPEL4WS Business Processes with WebSphere, "Business Integration :Understanding, Modeling,Migrating", IBM Redbook, December 2004
44. WOODGATE Scott, MOHR Stephen, LOESGEN Brian ,Microsoft biztalk server 2004 unleashedn, 768p. Paperback 2004.
45. David Skogan, Roy Gr, Ida Solheim, "Web Service Composition in UML," Enterprise Distributed Object Computing Conference, IEEE International, pp. 47-57, Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04), 2004.
46. Bull Service Oriented Architecture, BSOA Orchestra v3.0, Novembre 2006.
47. Patino-Martinez, Jimenez-Peris, Perez-Sorrosal, ZenFlow: a visual Web service composition tool for BPEL4WS, Visual Languages and Human-Centric Computing, 2005 IEEE Symposium. Page(s): 181 – 188
48. Chun Ouyang, Eric Verbeek, Will M.P . van der Aalst, Stephan Breutel, Marlon Dumas and Arthur H.M ter Hofstede,“WofBPEL“ : a tool for Automated Analysis of BPEL Processes“.
49. Stéphane Drapeau, Maxime Porhel, Etienne Juliot, « Comprendre SCA » Livre blanc v1.0, OBEO 2009.
50. Anis Charf, Mira Mezini. Aspect-oriented web service composition with AO4BPEL. In Proceedings of the 2nd European Conference on Web Services (ECOWS), volume 3250 of LNCS, pages 168–182. Springer, September 2004.
51. Anis Charfi, Benjamin Schmeling, Andreas Heizenreder,Mira Mezini. Reliable, secure, and transacted web service compositions with ao4bpel. In Proceedings of the 4th IEEE European Conference on Web Services, December 2006.
52. Van der Aalst, Nick Russell, Arthur H.M. ter Hofstede et Wil M.P. newYAWL: Specifying a Workflow Reference Language using Coloured Petri Net; Janvier 2009.
53. Van der Aalst; M. Adams, A.H.M. ter Hofstede, M. Pesic, and H.Schonenberg “Flexibility as a Service”, BPM Center Report BPM-08-09, BPMcenter.org, 2008.
54. Laukkanen, M.,Helin, H. Composing workflows of semantic web services. In Proceedings of the Workshop on Web-Services and Agent-based Engineering, 2003.

55. Waltinger, R. Web agents cooperating deductively. In Proceeding of FAABS (April 2000), Greenbelt, Ed., vol. 1871 of LNCS, Springer-Verlag, pp. 250–262.
56. McIlraith, S., and Son, T. Adapting. Golog for composition of semantic web services. In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR ’02) (Toulouse, France, April 2002), Morgan Kaufmann Publishers, pp. 482–493.
57. Narayanan, S., and McIlraith, S. In Proceedings of the Simulation, verification and automated composition of web services 11th international conference on World Wide Web (Honolulu, USA, July 2002), ACM.
58. Cheng, Z., Singh, M., Vouk, M, Composition constraints for semantic web services. In Proceedings of the WWW-2002 Workshop on Real World RDF and Semantic Web Applications,2002.
59. Constantinescu, I., Faltings, B., and Binder, W. Type based service composition. In Proceedings of the 13th international World Wide Web Conference on Alternate Track (New York, USA, 2004), ACM Press, pp. 268–269.
60. Jinmin Hu, Paul Grefen, Conceptual Framework and Architecture for Service Mediating Workflow Management, Information and Software Technology Journal 45 (13). pp. 929-939, 2003.
61. Anh Tuyet, Fédération : une Architecture Logicielle pour la Construction d’Applications Dirigée par les Modèles, thèse de DEA, UNIVERSITE JOSEPH FOURIER Grenoble (France) ,2006.
62. R.Allen. A Formal Approch to Software Architecture. Phd Thesis, Carnegie Mellen University, School of Computer Science, May 1997.
63. Nenad Medvidovic, Architecture-Based Specification-Time Software Evolution, Phd Thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 1999.
64. ZEL96 Gregory Zelesnik : «the UniCon language Reference Manual », Camegie Mellon University, Pittsburgh,Pensylvania,1996
65. Steve Vestal: "MetaH User's Manual" version 1.27, Honeywell Technologie Centern; 1998.
66. Jeff Magee, Naranker Dulay, Susan Eisenbach, Jeff Kramer: "Secifying Distributed Software Architecture" Proc of the 5th European Software Engeneering Conference (ESE'C95) LNC 989,(Springer-Verlag) page 137-153, 1995.

67. David C. Lukham: "Rapide, A langage Toolset for Simulation of Distributed System By Partial ordering of Events" In Proceedings of DIMACS Workshop on Partiel Order Methods in Verification (POMIV) page 329-358, July 25-26, 1996.
68. Conference ICSOFT 2007, Barcelone, INTEGRATING SOFTWARE ARCHITECTURE CONCEPTS INTO THE MDA PLATFORM, Alti Adel, Khammaci Tahar, Smeda Adel, Bennouar Djamal
69. Bennouar ,D., Khammaci, T., Henni, A. :Modeling The Component's Interaction Point In The IASA Approach, The Mediteranean Journal of Computer and Networks, Vol 4, N°4, 2008, © 2008 So ftMotor Ltd., UK
70. D. Garlan : Software Architecture. Encyclopedia of Software Engineering, John Wiley & Sons, Inc. 2001
71. D. Bennouar , A. Henni : SEAL :An aspect oriented ADL. Conférence ACIT 2009, Sanaa (Yemen), Décembre. 2009.
72. E Daschofy, D Garlan, A. van der Hoek, B Schmerl, An infrastructure for the rapid development of XML-based architecture description languages, Proceedings of the 24th International Conference on Software Engineering, Orlando, Florida 2002.
73. E Dashofy, UCI, xADL extension d'xArch, hiérarchisé et spécialisé "famille de produit" 2001.
74. B Schmerl ,xAcme, CMU, extension d'xArch, inspiré d'Acme, 2001, http://www.cs.cmu.edu/~acme/pub/xAcme/.
75. xOLAN L Bellissard & al, INRIA Rhône-Alpes, 2001, dérivé d'Olan
76. K. Bentlemsan, D. Bennouar, A. Henni: Composing Web services using the Integrated Approach to Software Architecture, the Third Symposium on Web Services, Dubai, 2010