REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOHAMED KHIDER - BISKRA

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
جامعة محمد خيضر بسكرة

Faculté des sciences exactes et des sciences de la nature et de la vie
Département d'informatique

THESE

Présentée pour l'obtention du diplôme de

DOCTORAT EN SCIENCES

Spécialité : Informatique

Présentée par :

Meftah Mohammed Charaf Eddine

Thème

UNE APPROCHE FORMELLE POUR LES APPLICATIONS WEB 2.0

Soutenu publiquement le : **devant le jury composé de :**

Président :	Pr. Chaoui Allaoua	Université de Constantine 2
Rapporteur :	Pr. Kazar Okba	Université de Biskra
Examinateur :	Pr. Petit Jean-Marc	INSA de Lyon
Examinateur :	Pr. Saidouni Djamel Eddine	Université de Constantine 2
Examinateur :	Dr. Kahloul Laid (MCA)	Université de Biskra
Examinateur :	Dr. Benharzallah Saber (MCA)	Université de Biskra

بسم الله الرحمان الرحيم

والصلاة والسلام على سيدنا محمد و على اله وصحبه أجمعين

إلى أمي وأبي الغاليين حفظهم الله

إلى زوجتي الكريمة وابنائي وإخوتي وأخواتي الأعزاء

إلى جميع معلمي واساتذتي المحترمين

إلى جميع الأهل والأقارب والاصدقاء

Remerciements

Je remercie **ALLAH** tout puissant, de m'avoir donné la force et la patience nécessaire pour achever ce travail de thèse.

Je tiens à présenter mes sincères remerciements et ma profond reconnaissance à mon directeur de thèse Monsieur KAZAR Okba, professeur à l'université de Biskra, pour sa disponibilité, sa patience et son soutien.

Je remercie beaucoup les membres du jury de m'avoir fait l'honneur d'accepter de participer à mon jury de thèse.

ملخص

.

هذا العمل هو عبارة عن طرح لمقاربة منهجية رسمية لتطوير الجيل الثاني والذي يليه من تطبيقات الويب ,حيث أن هذه التطبيقات هي عبارة عن مزيج متنوع من التكنولوجيات وهي تعتمد أساسا على بنية هندسية ذات توجه خدماتي من ناحية,وعلى تقنيات الأجاكس لتطوير وسائطها البينية مع المستعملين من ناحية أخرى,إن هذا المزيج من الطرق والتقنيات يتطلب من الباحثين دراسة وطرح حلول يمكن من خلالها تطوير تطبيقات سليمة وخالية من النقائص والأخطاء ,ما يجعلها تؤدي دورها كما هو مسطر لها ابتداءا.بعد دراسة لكثير من الأعمال التي تم طرحها في هذا الميدان لاحظنا أن أغلب هاته الأعمال إن لم نقل كلها تطرق إلى كيفية المنهجة الرسمية للجانب الهندسي لهذه التطبيقات ,**مع عدم التطرق لإعتبار وجود** الجانب التكنولوجي الذي يُعنى بالوساطة البينية مع المستعملين,أو العكس بعض الأعمال الأخرى اهتمت بالمهنجة الرسمية **للغة البر**مجة جافا باعتبارها كلغة برمجة بحته دون اعتبار ارتباطها بالهندسة الخدماتية لتطبيقات الويب,لذا حاولنا في هذه الأطروحة طرح منهجية رسمية شاملة للجانبين المذكورين الهندسي والتكنولوجي تعتمد على مفاهيم في مجال الرياضيات بغرض استعمالها أثناء سيرورة عملية تطوير تطبيقات الويب,وبالتالي الحصول على تطبيقات ويب أكثر سلامة من وأقل نقائص مع إمكانية تطويرها وانشاءها آليا.

كلمات البحث

تطبيقات ويب 2, خدمات الويب, مقاربة منهجية رسميةB ,Java, BPL4SW.

Résumé

Ce travail propose une approche formelle pour le développement des applications Web 2.0 sûres. Cette approche consiste en la génération d'une implémentation de l'application à partir de spécifications formelles. On décrit préalablement l'application à l'aide de notations (CTT), puis un processus automatique est appliqué afin de les traduire en spécifications formelles B. En utilisant le processus de raffinement B, un ensemble de règles de raffinement, opérant sur les opérations, est appliqué sur les spécifications ainsi obtenues. Ces règles considèrent l'aspect dynamique (opérations) de l'application Web étudiée ; ces phases de raffinement ont pour but de rendre les spécifications finales proches des langages d'implémentations cibles choisi (WS-BPEL, JAVA), de telle sorte que la dernière phase de codage devienne intuitive et naturelle. De manière générale, le processus de raffinement est une tâche manuelle, relativement coûteuse, en particulier en phase de preuve. Grâce au caractère générique de ces règles de raffinement, un outil de raffinement assisté peut être réalisé, permettant ainsi la réduction du coût du processus de raffinement.

Mots Clés

Application Web, Méthode Formelle B, Raffinement, BPEL4SW, JAVA

Abstract

We propose a formal approach for development of safe web applications. This approach involves the generation of a web application on both sides (users' side (Ajax) and the web service side (Composition)) from formal specifications. First, we describe the application using symbolic notations (CTT), then an automatic process is applied in order to translate them into formal specifications B. Using the B refinement process, a set of refinement rules, operating on data and operations. These phases of refinement are intended to make the final specifications close to the implementation language chosen so that the last coding phase becomes intuitive and natural. In general, the process of refinement is a manual task, relatively expensive; with character generic of these refinement rules, an assisted refinement tool can be achieved, enabling reduction the cost of the refining process. Finally, in the case study, we have developed a complete application (Travel Agency) in order to show the benefits of our approach.

Keywords

Web application, Formal approach, B Model, refinement, BPEL4SW, JAVA

TABLE DES MATIÈRES

Chapitres 1 : Introduction Générale.....10

I.1 Le contexte.....10
I .2 Les problématiques et les motivations.....13
I .3 L'objectif.....14
I.4 La démarche.....15
I.5 Contributions..... …….....15
I.6 L'organisation de la thèse 16

Chapitres 2 : Etat de l'art de Web 2+.....17

II.1. Introduction 18
II.2. Définitions 19
II.3. L'aspect Social de Web 2.....22
II.4. Les Architectures Orientées Service 23
II.4.1 Les services 23
II.4.2 Le modèle de référence OASIS 24
II.4.3 Les architectures orientées service Web 25
II.4.3.1 Les services web.....25
II.4.3.2 Les différentes couches de l'architecture orientés service Web 28
II.4.3.2. 1. La couche Transport.....29
II.4.3.2. 2. La couche communication (messages) 30
SOAP.....30
II.4.3.2. 3. La couche description 31
WSDL.....32
UDDI.....34

Le Web sémantique.....34
II.4.3.2. 4 La couche qualité de service 36
II.4.3.2. 5. La sécurité 36
II.4.3.2. 6. La couche processus 37
La chorégraphie.....37
WSCI.....38
WS-CDL 38
L'orchestration 39
XLANG.....40
WSFL.....40
BPML.....40
Le langage BPEL 41
II.5. Les bases technologiques du Web 44
II.5.1 Représentation de technique de l'approche AJAX 44
II.5.2 Le principe AJAX.....45
II.5.3 Les frameworks Java WEB.....47
II.5.4 Les caractéristiques de java 48
II.5.5 Le développement des interfaces graphiques.....51
II.5.5.1 Les éléments d'interfaces graphiques de l'AWT 52
II.5.5.2 L'interception des actions de l'utilisateur avec Java 54
II.5.6 Principes de la programmation par événement.....55
II.5.6.1 Modèle de délégation de l'événement en Java.....55
II.5.6.2 Notion d'état d'un objet.....56
II.5.6.3 Notion de comportement d'un objet.....56
1.5.6.4 Les opérations.....56
II.5.6.5 Rôle et responsabilités.....57
II.5.6.6 Les objets en tant que machines.....58
II.5.7 Programme Java-Swing.....58
II.5.7.1 Structure.....58
II.5.7.2 Contrôle de flux.....58
II.6 Conclusion 60

Chapitre 3 : (Les travaux Liés) : 62

III.1 Introduction.....63

III.2 Les systèmes de transitions LTSA-WS 64

III. 3 Les réseaux de Pétri 66

III.4 Les algèbres de processus 67

III.5 ASDL.....68

III.6 LOTOS/CADP……..……..……..69

III.7 Le π –calcul 72

III. 8 Les théories temporelles.....73

III. 9 Les logiques temporelles.....74

III. 10 Le langage FIACRE 74

III. 11 Les logiques temporelles arborescentes basées sur actions……..…..……...76

III. 12 La logique ACTL.....76

III. 13 L'approche Diapason 78

III. 14 Approche dirigée par les modèles (MDA-UML-S) 80

III. 15 Autres formalismes 81

III. 16 Synthèse et Conclusion.....82

Chapitre 4 : (L'approche proposée) …….……..86

IV.1 Introduction……..……..….87

IV.2 L'approche proposée……..……... 88

IV.2.1 Les techniques utilisées…….……..……91

IV.2.1.1 CTT (Concur Task Tree) ……..…….……91

IV.2.1.2 La méthode formelle B ……..……..……92

IV.2.2 Meta Model et Liens structurels entre spécifications B, Service Web et diagrammes de classes (Java) ……...……94

IV.2.3 L'Architecture générale de l'approche proposée ……..……..97

IV.2.4 Modèle pour la spécification …….……99

IV.2.5 Modèle pour la vérification et la validation ……100

IV.2.6 Modélisation des propriétés statiques et dynamiques……..102

IV.2.7 Le raffinement…….103

IV.2.7.1 Les types de raffinement ……..……..…..…103

IV.2.7.2 La dépendance des messages ……..…..……..104

IV.2.7.3 Niveaux des raffinements proposées pour les applications Web2+……...…..104
IV.2.8 La décomposition de complexités en B pour une application Web 2+……...….109
IV.2.9 Développement pratique pour la génération automatique de code…….….111
IV.2.9.1 Structure générale ……111
IV.2.9.2 Les Règles de traduction ……112
IV.2.9 .2.1 Les règles de codage descendant (B-BPEL4SW)……...112
IV.2.9 .2.2 La traduction ascendante (BPEL4SW-B) ……..……..…112

IV.2.9 .2.3 Les règles de codage descendante (B-JAVA) ……..…114
IV.2.9 .2.4 La traduction ascendante (JAVA-B) ……..114
IV.3 Conclusion …….……118

Chapitre 5 : (Etude de cas) ……119

V.1 Introduction…….……..…120
V.2 Préparation de l'environnement de développement…….……122
V.3 Composition des services de l'application (Processus BPEL4SW)…….……...122
V.3.1 Définir les liens partenaires des services Web concernés (importation des fichiers WSDL externes) ……122
V.3.1.2 Définir la logique du processus (des notations CTT)…….....…124
V.3.1.3 La traduction en spécifications formelles B……..…….….125
V.3.1. 4 La génération automatique de code source (BPEL4SW)……..……126
V.4 Le comportement de l'interface de l'application agence de voyage……129
V.4.1 Spécifications formelles B de l'interface de l'application…….…..130
V.4.2. La génération automatique de code source (JAVA)……..……135
V.5 Outils B: Click'n prouve et B4free…….……...137
V.1 **Comparaison** ……..……..139
V.2 **Conclusions générales et perspectives** ;;……...……140
Bibliographies…….……...……142

Liste des figures

Figure I.1: Contexte du travail. 11
Figure II.1 La différence entre le Web 1.0 et 2.0.....21
Figure II.2 Les aspects de Web 2.0…….……22
Figure II.3 Schéma générale d'une WSOA…….……27
Figure II.4 Les couches d'une architecture orientée service Web……..29

Figure II.5 Le protocole SOAP.....31
Figure II.6 Structure d'un document WSDL.....33
Figure II.7 La chorégraphie des services web.....38
Figure II.8 L'orchestration des services web……..……..……39
Figure II.9 Processus BPEL4SW…….....42
Figure II.10 Les activités dans un processus BPEL4SW.....44
Figure II.11 Le principe AJAX.....46
Figure II.12 les tiers techniques des applications web 2+.....48
Figure II.13 Les éléments d'interfaces graphiques de l'AWT.....53
Figure II.14 Modèle de délégation de l'événement en Java.....55

Figure III.1 Les composants des réseaux de Pétri……66
Figure III.2 LOTOS (Exemple : producteur-consommateur)…..……..69
Figure III.3 LOTOS (Exemple-suite : producteur-consommateur)……..70
Figure III.4 Communication inter processus……72
Figure III.5 Structure de la transformation (BPEL-FIACRE)……..75
Figure III.6 Les couches du langage π –Diapason.....…...78
Figure III.7 Processus de développement architectural de l'approche Diaposan..……79
Figure III.8 Approche MDA (MDA-UML-S)……80

Figure IV.1 Mise en relation des Tâches applicative ……..……89

Figure IV.2 Décomposition et raffinement des activités…….……..… ..…..90

Figure IV.3 Grammaire de langage CTT……91

Figure IV.4 Structure de machine B……...……..92

Figure IV.5 Substitutions généralisées de Disjkstra …… ..…….93

Figure IV.6 Structuration d'un développement en B……..94

Figure IV.7 Correspondances entre méta-modèles B, Service Web et Class Java…….…….95
Figure IV.8 L'architecture générale de l'approche proposée……..……98
Figure IV.9 Approche descendante (B-BPEL4SW, B-JAVA)……...…..……...101
Figure IV.10: Approche Ascendante (BPEL4SW-B, JAVA-B)…….……..101

Figure IV.11: Approche de raffinement pour les applications Web 2+.....108

Figure IV.12: Mauvaise composition des taches.....109
Figure IV.13: Bonne composition des taches 110

Figure V.1 Les services web d'une agence de voyage……..……...……..121

Figure V.2 La logique du processus de l'agence…….……..124

Figure V.3 La traduction en spécifications formelles B……..125

Figure V.4 Le modèle B de service web Agence de Voyage……..126

Figure V.5 Modèle B-BPEL4SW de service Web Agence de Voyage.……..127

Figure V.6 Code source BPEL4SW de service web Agence de Voyage……129

Figure V.7 L'interface de L'application web (Agence de voyage)…….130

Figure V.8 Les composants B du modèle de service web Agence de Voyage (1)……..138

Figure V.9 Les composants B du modèle de service web Agence de Voyage (2)……..139

Liste des tableaux

Tableau III.1 Tableau comparatif des travaux liés……..…….....…….83

Tableau IV.1 Niveaux des raffinements proposées pour les applications Web2+….……..105
Tableau IV.2 Génération automatique du code (Structure Générale)……..111

Tableau IV.3 Génération automatique du code BPEL4SW (règles de traductions) …..112

Tableau IV.4 Génération automatique du JAVA (règles de traductions). …….……..115

Tableau V.1 Outils de développement…….……..….122

Tableau V.2 Les URL WSDL des services Web ……..….123

Tableau VI.1 Comparaison entre l'approche proposée et les autres approches de domaine…139

I

Introduction Générale

L'un des objectifs du génie logiciel est de rationaliser le développement des applications de qualité. Ces dernières années, des progrès considérables ont été réalisés dans la conception et le développement des applications et des services web participatif (Web 2.0). L'expression « Web 2.0 » désigne certaines des technologies et des usages du World Wide Web qui ont suivi la forme initiale du web, en particulier les interfaces permettant aux internautes ayant peu de connaissances techniques de s'approprier les nouvelles fonctionnalités du web et ainsi d'interagir de façon simple à la fois avec le contenu et la structure des pages et aussi entre eux, créant ainsi notamment le Web social. En ce sens, les sites (applications) Web participatif facilitent l'interaction entre utilisateurs ; et agissent plus comme des points de présence, ou portails Web centrés sur l'utilisateur plutôt que sur les sites Web traditionnels.

Le Web 2.0 repose sur trois piliers : une architecture orientée service (SAO), des applications web riches (RIA) et un aspect social web.

L'aspect social fait référence à une vision d'internet considéré comme un espace de socialisation, un lieu dont l'une de ses fonctions principales est de faire interagir (ensemble des relations) les utilisateurs (individus) entre eux afin d'assurer une production continuelle de contenu. Les concepts sociologiques pour définir un réseau social sont : les individus, leurs liens (contacts) et l'environnement. Ceci est un côté purement sociologique n'a pas d'importance dans le contexte de notre travail.

I .1 Le contexte

Le contexte du travail présenté dans cette thèse s'articule autour les domaines suivants :

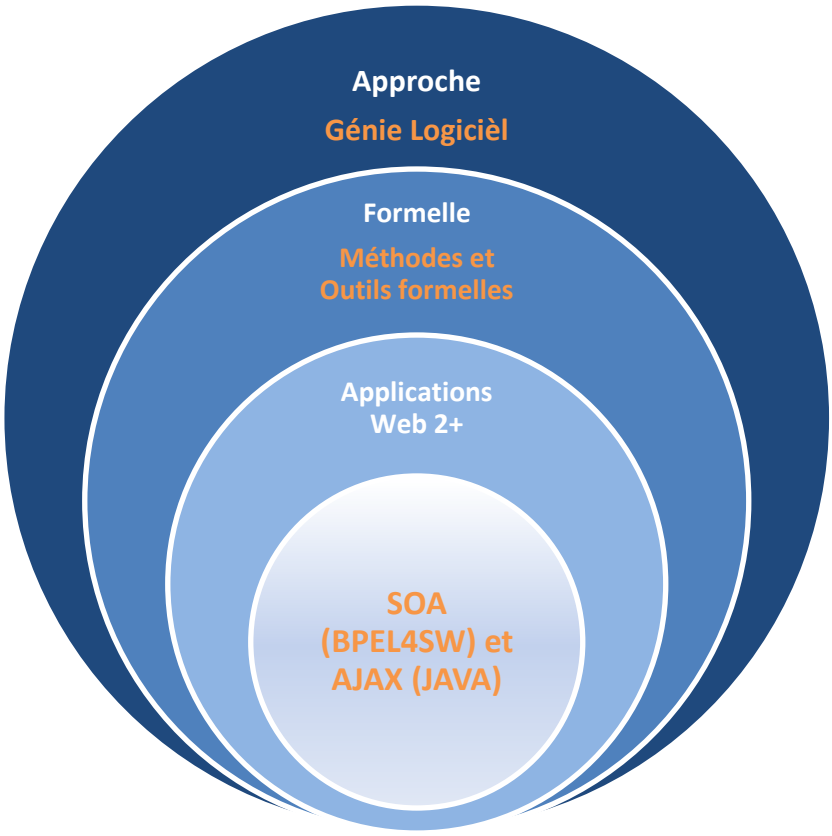

Figure I.1: Contexte du travail.

- **Le génie logiciel**

Est l'ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi. Le génie logiciel s'intéresse en particulier aux procédures systématiques qui permettent d'arriver à ce que des logiciels de grande taille correspondent aux attentes du client, soient fiables, aient un coût d'entretien réduit et de bonnes performances tout en respectant les délais et les coûts de construction.

Le génie logiciel touche au cycle de vie des logiciels. Toutes les phases de la création d'un logiciel informatique : l'analyse du besoin, l'élaboration des spécifications, la conceptualisation du mécanisme interne au logiciel ainsi que les techniques de programmation, le développement, la phase de test et finalement la maintenance.

L'approche formelle est connexe au génie logiciel dans la mesure où ils partagent des outils communs.

- **Le domaine des méthodes formelles**

Les méthodes formelles sont des techniques informatiques permettant, à l'aide de règles mathématiques, logiques et de langages spécialisés de s'assurer (idéalement) de l'absence de tout défaut de programmes informatiques, En plusieurs étapes :

- ✓ *Spécifier* le logiciel en utilisant les mathématiques ;
- ✓ *Vérifier* certaines propriétés sur cette spécification (Corriger la spécification si besoin) ;
- ✓ *Raffiner* ou *dériver* de la spécification un logiciel concret ;
- ✓ *Génération automatique de code* : faire un *lien* entre la spécification et (une partie d') un logiciel concret.

Pour cela il y a nombreux formalismes :

- Spécification algébrique, Machines d'état abstraites, Interprétation abstraite, *Model Checking*, Systèmes de types, Démonstrateurs de théorèmes automatiques ou interactifs, …

- **Le domaine SOA (ou Architecture Orientée Service Web) : cadre d'un environnement des services web**

Le web 2.0 comme les technologies SOA conçoivent les logiciels comme des services. Plus important encore, ils repensent les services comme des plates-formes. Plutôt que de considérer les services comme des produits dont le mode de consommation est déterminé à l'avance, ces deux ensembles de technologies considèrent qu'un service est avant tout une brique dont le rôle est de permettre à d'autres services de se construire en l'utilisant.

Notre sujet s'articule autour l'utilisation des méthodes formelles pour les applications et les services (un service définit une entité: ressource, module, composant logiciel, application, etc.…qui communique via un échange de messages) de web participatif dans un contexte du Génie Logiciel.

- **Le domaine des applications web riches (RIA)**

Le concept d'architecture de type Rich Internet Application (RIA) illustre bien les évolutions récentes induites par le Web 2.0. La notion de RIA fait référence à une évolution essentielle des applications Web. Auparavant basé sur un simple navigateur Web se comportant comme un client léger, l'architecture évolue désormais vers un modèle plus complexe où, l'interface utilisateur intègre localement une partie de plus en plus importante des applications.

Ces usages ainsi enrichis sont rendus possibles par des technologies « applications client » comme Java, Ajax et Flash, , utilisant des protocoles Internet et Web standard. Dans le modèle RIA, le temps de téléchargement d'une page Web devient un indicateur non significatif, puisque le moteur client peut contenir en local une partie du contenu attendu.

Ajax (acronyme de Asynchronous Javascript and XML) est une manière de construire des applications Web et des sites web dynamiques basés sur diverses technologies Web ajoutées aux navigateurs dès 1995.

Ajax est la combinaison de technologies telles que Java, CSS, XML, le DOM et le XMLHttpRequest dans le but de réaliser des applications Web qui offrent une maniabilité et un confort d'utilisation .

Dans ce qui suit nous présentons, la problématique et les motivations, les objectifs du travail, la démarche, les contributions, et enfin la structure d'organisation de la thèse.

I .2 Problématiques et Motivations

Dans ce type d'applications web (2+), les concepteurs ont plutôt tendance à utiliser des méthodes dites semi-formelles telles: UML, OMT... basées principalement sur des notations graphiques (classes, entités, états/transitions,...) ou symboliques, permettant une représentation intuitive, simplifiée et synthétique du système à étudier. Ces méthodes représentent des avantages indéniables pour la modélisation. Elles constituent un support idéal pour la communication entre les différents acteurs du système. Néanmoins, ces méthodes souffrent encore d'un manque de sémantique précise de leurs concepts. Ce manque de sémantique réduit considérablement toute possibilité de raisonnement et de preuve sur les modélisations ainsi obtenues.

Dans le Web 2.0, les internautes sont considérés comme co-développeurs et l'internaute devient acteur en alimentant les sites en contenu (exemple : blogs,…). A l'heure où le triptyque : coût, qualité et délai, devient un maître mot ; et différencient aujourd'hui les approches orientées service Web des précédentes approches, et deviennent un réel défi.

Le langage BPEL4WS est actuellement le standard émergeant pour la description de processus mettant en œuvre des services Web. Cependant, la sémantique de BPEL4WS n'est pas clairement définie et peut prêter à confusion.

En effet, la sémantique opérationnelle de chacune des structures du langage BPEL4WS n'étant pas formellement décrite, personne ne peut garantie que :

- ✓ D'une part, l'orchestration exécutée aura exactement le comportement que l'on pense avoir d´écrit en BPEL4WS,
- ✓ D'autre part, qu'une orchestration décrite en BPEL4WS aura une exécution identique quel que soit l'interpréteur BPEL4WS.

De l'autre côté le langage Java n'a pas été défini avec une sémantique formelle (De nombreux travaux ont proposé une telle formalisation, mais ils traitent en général un sous-ensemble du langage). ; Une preuve de correspondance entre un code Java est une spécification apparaît donc impossible à ce stade. Cependant, il est tout à fait envisageable d'associer à un code Java une implémentation ou un raffinement formelle. Il s'agit ici d'une étape de traduction, que l'on veut autant que possible automatique. Une implémentation ou un raffinement dispose d'une sémantique formelle ; il est donc envisageable de rédiger, avec l'assistance d'un outil, des abstractions successives, jusqu'a aboutir à une spécification sous la forme d'une machine abstraite manipulant des concepts de haut niveau.

Toutes les approches proposées dans ce domaine séparant entre l'aspect architectural (Orchestration des service web) et l'aspect technique (AJAX) lors de développement, et ce dernière est complètement ignoré lors de la formalisation des applications (services) web, ce qui peut réduire considérablement l'efficacité de formalisation des applications développées , et finalement, il n'y a aucune approche complet qui traite la développement des applications web tenant compte les des deux aspects architectural et technologique dans une approche unique.

I .3 L'objectif

L'objectif de notre travail est la définition d'une approche formelle pour le développement des applications Web 2.0 sûres. En analysant les travaux de recherche existants sur la formalisation de développement des applications web, nous détectons certaines limites cruciales: la nécessité d'une nouvelle approche pour décrire tous les aspects descriptifs des applications web.

Afin de répondre aux limitations citées ci-dessus, nous proposons une approche complète qui propose une formalisation de processus de développement des applications web 2+ tenant compte les des deux côtés architectural et technique. Ce qui permet de :

- ✓ Raisonner sur les organisations des composants et des services web et d'effectuer des vérifications des propriétés.
- ✓ Faciliter la génération automatique de code (BPEL, JAVA).
- ✓ Faciliter la génération de tests et permettre la réutilisation des tests pour tester de nouveaux services.

- ✓ Définir de manière formelle des fonctionnalités que les services doivent fournir, et pour tester si un service qui vient d'être ajouté (une nouvelle fonctionnalité) fonctionne correctement par rapport aux contraintes exprimées.
- ✓ Dans le domaine de la sécurité : obligation d'utiliser des méthodes formelles pour certains niveaux de sécurité.
- ✓ L'obtention d'une modélisation formelle non seulement précise et concise mais surtout analysable par des outils (contrôle de types, preuves, simulation, …).

I .4 Démarche

L'approche proposée dans cette thèse consiste à la génération d'une implémentation de l'application à partir de spécifications formelles. On décrit préalablement l'application à l'aide de notations (CTT), puis un processus automatique est appliqué afin de les traduire en spécifications formelles B. En utilisant le processus de raffinement B, un ensemble de règles de raffinement, opérant sur les opérations, est appliqué sur les spécifications ainsi obtenues. Ces règles considèrent l'aspect dynamique (opérations) de l'application Web étudiée ; ces phases de raffinement ont pour but de rendre les spécifications finales proches des langages d'implémentations cibles choisi (WS-BPEL, JAVA), de telle sorte que la dernière phase de codage devienne intuitive et naturelle. De manière générale, le processus de raffinement est une tâche manuelle, relativement coûteuse, en particulier en phase de preuve. Grâce au caractère générique de ces règles de raffinement, un outil de raffinement assisté peut être réalisé, permettant ainsi la réduction du coût du processus de raffinement.

I .5 Contributions

Les contributions de l'approche présentée dans ce travail s'articule autour des avantages suivants:

- ✓ Réduction du coût de développement : automatisation des phases de spécification, de raffinement, et de génération automatique de code des applications Web.

- ✓ Normalisation du code généré : les deux phases de traduction et de raffinement sont dictées par des règles précises et déterministes. Cette normalisation de code offrira une meilleure compréhension et maintenance du code ainsi produit.

- ✓ La proposition des métas modèles des structures (B-BPEL4SW-JAVA).

- ✓ L'approche proposée c'est une approche formelle unique descendante/ ascendante pour la spécification, la validation formelle et la génération automatique de code des orchestrations WS-BPEL et de code Java.

La suite de ce travail est organisée de la façon suivante :

Chapitre 2 : nous présenterons un état de l'art sur les architectures orientées service et plus particulièrement orientées service Web. Nous détaillerons les différentes couches relatives à ces architectures ainsi que les différents langages et travaux au sein de chacune des couches. Dans ce chapitre nous présenterons aussi un état de l'art sur les bases technologique du Web (AJAX et JAVA).

Chapitre 3 : dans ce chapitre nous présenterons plusieurs travaux tentent de pallier le manque de formalisme et ainsi permettre certaines vérifications sur le comportement des applications Web. Nous ferons enfin la synthèse et une analyse comparative de ces travaux au regard de notre objectif

Chapitre 4: nous détaillerons notre propre approche, qui permet de raisonner formellement sur les applications web. Pour ce faire, nous présenterons notre schéma globale de notre approche puis une vue générale sur les modèles utilisés (les notations CTT et plus particulièrement la méthode B). Nous verrons ensuite comment nous avons tenté de utilisé ces dernières, pour les utiliser dans le cadre de la spécification, génération automatique de code (BPEL, JAVA), vérification et la validation formelle des applications Web.

Chapitre 5 : nous présenterons une étude de cas (Agence de voyage) qui nous a permis de valider notre approche. Nous détaillerons pas à pas les différentes étapes de cette dernière à travers ce cas d'étude et présenterons l'environnement que nous avons développé pour supporter l'approche proposée. Nous compléterons cette étude de cas par différents scénarios de génération automatique des codes (BPEL et JAVA) et présenterons dans ce cadre les solutions apportées par notre approche.

Conclusion : Nous terminerons notre travail par un bilan comparatif critique des approches proposées avec notre approche et nous exposerons quelques perspectives que nous envisageons comme suite à donner à ce travail.

Chapitre 02

Etat de l'art de Web 2+

II
Web 2+

II.1. Introduction

Depuis que l'homme existe, que ce soit dans la vie personnelle ou professionnelle, il se regroupe par centres d'intérêt pour former des réseaux. Alors les réseaux sociaux ont toujours été présents dans la vie sociale de l'être humain et ils n'attendaient plus qu'un moyen pour prendre de l'ampleur. Ce moyen est l'internet et les nouvelles technologies. Maintenant, les individus ont la possibilité de se regrouper en ligne via Internet en particulier sur les réseaux sociaux.

Internet s'ouvre véritablement au grand public, lors du Conseil européen pour la recherche nucléaire (CERN), en 1991, du World Wide Web. Le web et l'Internet sont parfois difficiles à distinguer pour l'internaute. Internet désignant un réseau de réseaux tandis que le web est une application d'Internet. Le web se réfère ainsi plus à l'aspect informationnel qu'à la structure physique. Il s'agit d'un système d'interface graphique, très ergonomique et très facile d'utilisation, qui permet de passer d'une page ou d'un site à un autre en "cliquant" sur un lien dit "hypertexte". La navigation sur la "Toile" devient ainsi extrêmement aisée. Le web ouvre donc le réseau à de nouveaux utilisateurs peu familiarisés avec l'informatique. En quelques mois, les sites web se multiplient ; et le Web est devenu un phénomène technique et social de grande ampleur.

Depuis ses débuts le Web a fortement renforcé la diffusion d'information à travers le monde ; mais il faut noter que les réseaux d'ordinateurs ; en plus de la diffusion d'information ; vont participer au développement de la communication. La relation producteur- consommateur qui caractérise la première version du Web, est remplacée par la participation.

Depuis quelques années, et plus précisément depuis 2005, le web que nous connaissions, celui du *html* et des sites personnels, subit de fortes mutations tant au niveau technologique que fonctionnel ; avec l'apparition de la technologie AJAX (Javascript + XML) a permis des interactions plus rapides avec les pages Internet. De ce fait, le nombre de membres de ces réseaux sociaux s'est allongé. D'une part car les interactions étant plus rapide, consulter Internet est devenu plus confortable. Mais d'autre part, car les utilisateurs prennent conscience de leur pouvoir d'interagir sur la toile. C'est ce qui a donné naissance au Web 2.

Dans ce chapitre nous étudions les concepts du Web 2. La première partie définit la notion de web 2 ou le web participatif. Dans la seconde partie, nous présentons les concepts. de SOA et les

différentes couches de modèle de référence OASIS, nous présentons par la suite les principaux concepts comportemental de langage Java et enfin une conclusion de ce chapitre.

II.2. Définitions

L'expression « Web 2.0 » utilisée par Dale Dougherty en 2003, diffusée par Tim O'Reilly en 2004 et consolidée en 2005 avec l'exposé de position « What Is Web 2.0 » s'est imposée à partir de 2007. Les principes fondateurs du web 2.0 sont ceux exposés par Tim O'Reilly et consolidés en octobre 2005 dans un article désormais célèbre publié en septembre 2005 : "What is the web 2.0" [Tim, 2005].

O'Reilly et Battelle résument [Tim et Batte , 2005] les principes-clés des applications Web 2.0 comme-suit :

- ✓ Le Web comme plate-forme ;
- ✓ Les données comme « connaissances implicites » ;
- ✓ Les effets de réseau entraînés par une « architecture de participation » ;
- ✓ l'innovation comme l'assemblage de systèmes et de sites distribués et indépendants ;
- ✓ Des modèles d'entreprise poids plume grâce à la syndication de contenus et de services ;

Le Web 2.0 désigne certaines des technologies et des usages du World Wide Web qui ont suivi la forme initiale du web [MonInfo 2005] ; Dans le Web1.0, l'internaute était seulement passif. A partir du Web 2.0, l'internaute, l'utilisateur devient actif : il crée son propre contenu, il partage des informations (photos, vidéos…), il se crée un réseau.

Depuis l'émergence de ce terme, il est utilisé comme une annexe « 2.0 » à tout concept pour signifier « s'appuyant sur le web 2.0 » (par exemple le Marketing 2.0). De même, pour décrire l'évolution du Web, de nombreux numéros de versions ont été proposés (Web 1.0 pour désigner, par opposition, les débuts du Web).

Le Web 2.0 ne constitue un concept de dimension technologique, car ses applications s'appuient sur des technologies déjà existantes. L'évolution n'est pas tant dans la technologie elle-même que dans la façon de mixer ces technologies pour apporter des services et une ergonomie nouvelle à l'utilisateur. La relative simplicité de mise en place de ses techniques et leurs coûts de programmation réduits expliquent le très fort développement que connaissent toutes ces applications.

Le Web 2.0 est une évolution du Web vers plus de simplicité (ne nécessitant pas de connaissances techniques) et d'interactivité (permettant à chacun de contribuer sous différentes formes) [MonInfo 2005] ; en particulier les interfaces permettant aux internautes ayant peu de connaissances techniques de s'approprier les nouvelles fonctionnalités du web. Ainsi, les internautes contribuent à l'échange d'informations et peuvent interagir (partager, échanger, etc.) de façon simple, à la fois avec le contenu et la structure des pages, mais aussi entre eux, créant ainsi notamment le Web social (Ou « Web communautaire », « Web interactif », « Web participatif »).

Dans le Web 2.0, l'internaute devient acteur en alimentant les sites en contenu (exemple : *blogs*), éventuellement collaborativement avec les *wikis*, et devient ainsi co-créateur de nouvelles applications en ligne, initiant souvent de façon collective ou communautaire de nouvelles formes de relations numériques. Les nouveaux produits et services associés au web 2.0 (mutualisation des connaissances, échange, travail collaboratif, interactivité, intelligence collective…etc.) ont réussi à repositionner l'internaute au cœur du système.

En ce sens, les sites Web 2.0 agissent plus comme des points de présence, ou portails Web centrés sur l'utilisateur plutôt que sur les sites Web traditionnels. L'évolution des supports permettant de consulter les sites Web, leurs différents formats, amène en 2008 une approche recentrée sur le contenu plus que sur l'aspect. Les nouveaux gabarits Web 2.0 (en anglais : template) tentent d'apporter un soin graphique, des effets, en restant compatibles avec cette diversité de supports.

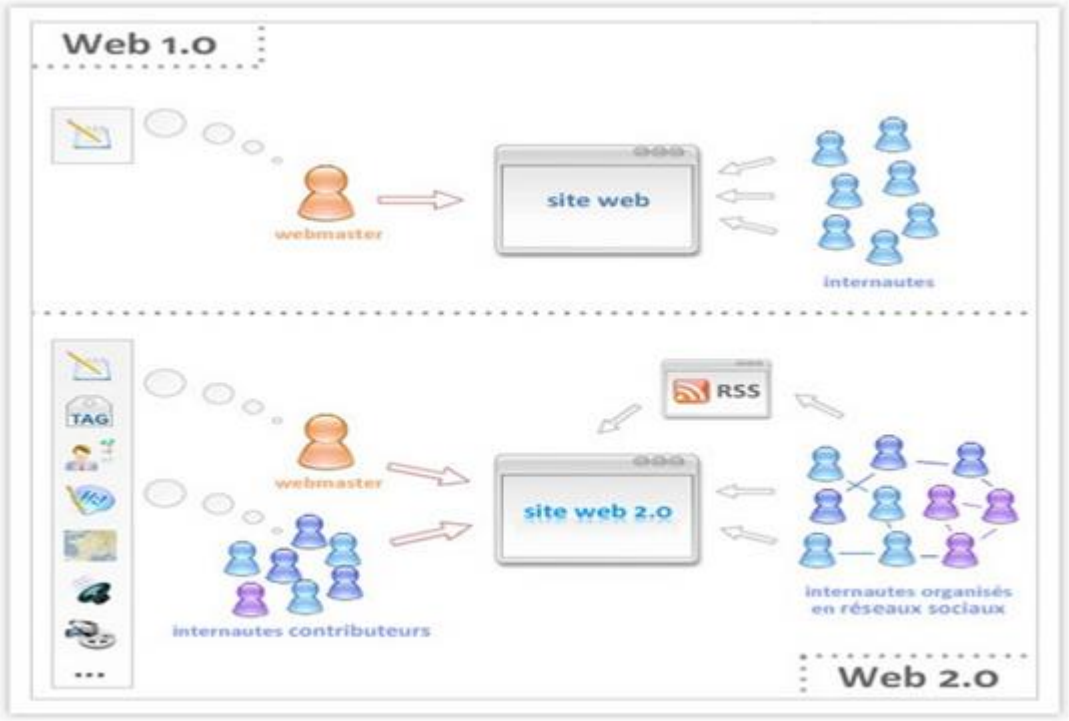

Figure II.1 La différence entre le Web 1.0 et 2.0.

En même temps, la tendance fait que l'on est en train de passer, grâce au web 2.0, d'une collection de sites web à une plateforme informatique à part entière, fournissant des applications web aux utilisateurs, et permettant aux internautes de devenir de véritables co-développeurs des applications ; par ailleurs, le web 2.0 en permettant le passage de la notion de logiciel produit à celle de logiciel service [Asse et Alii, 2007].

Pour ci raisons le Web 2.0 s'articule autour de trois aspects : Social, Architectural et Technologiques. Ces aspects sont présents sur la figure ci-dessous :

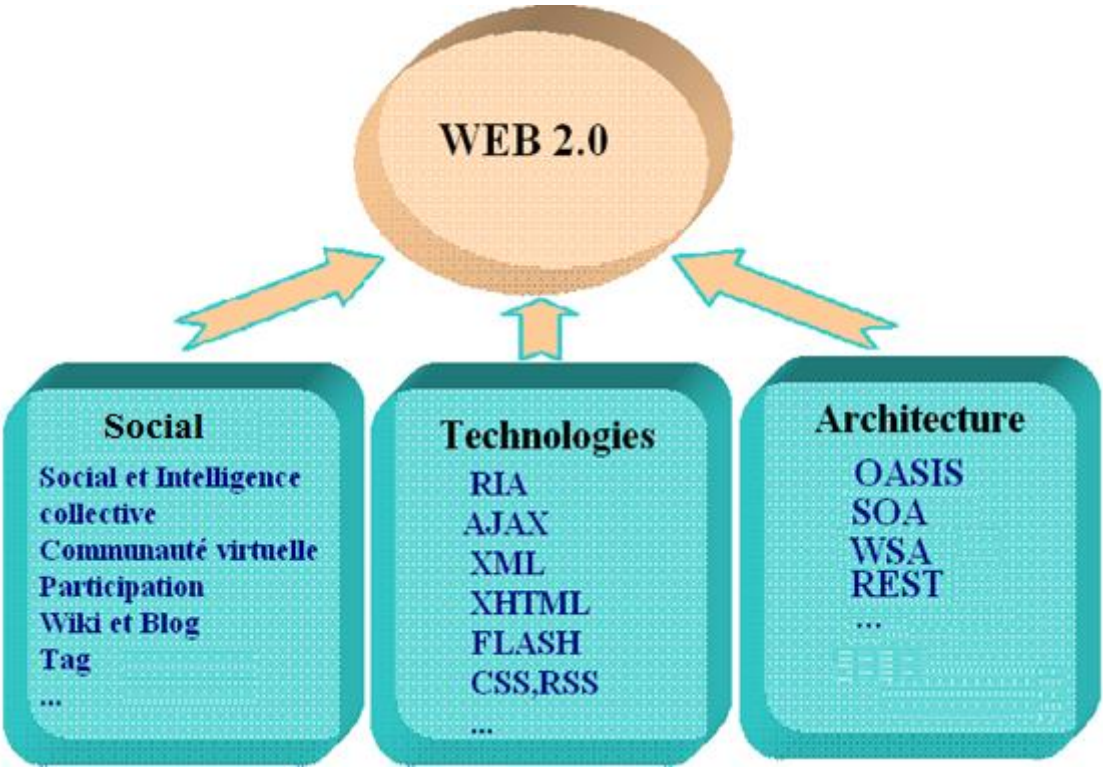

Figure II.2 Les aspects de Web 2.0.

Deux aspects se côtoient quand on parle de réseaux sociaux : d'un côté, l'aspect purement sociologique et communautaire et de l'autre côté purement informatique c'est l'aspect architectural et technologique.

II.3. L'aspect Social de Web 2

L'aspect social fait référence à une vision d'Internet considéré comme un espace de socialisation, un lieu dont l'une de ses fonctions principales est de faire interagir les utilisateurs entre eux afin d'assurer une production continuelle de contenu, et non plus uniquement la distribution de documents.

D'un point de vue sociologique, selon [Wass, 1994].Un réseau social est un ensemble de relations entre des entités sociales (individus). Les contacts entre ces individus peuvent être, par exemple, des relations de collaboration, d'amitié, ou des citations bibliographiques. Ces ressources sont donc aussi bien formelles qu'informelles, matérielles qu'immatérielles. Toujours selon [Wass, 1994], trois concepts sont également retenus dans cette analyse des réseaux sociaux :

- ✓ Les acteurs et leurs actions sont considérés comme des entités indépendantes.
- ✓ L'environnement des acteurs procure des opportunités et exerce des contraintes sur leurs actions individuelles.
- ✓ Les structures sociales, politiques, économiques, etc. ont une influence sur les formes de relations entre les acteurs.

Les concepts sociologiques pour définir un réseau social sont : les individus, leurs liens (contacts), leurs affinités et l'environnement les entourant.

II.4. Les Architectures Orientées Service

II.4.1 Les services

L'un des objectifs du génie logiciel est de rationaliser le développement d'applications de qualité. En particulier, la question de la réutilisation des briques logicielles déjà développées constitue un enjeu majeur. Les approches objet, composant et service donnent une des réponses à cette problématique.

Le concept de service est actuellement le sujet de définitions très variées. Nous en avons retenu trois, qui se placent selon différents points de vue à priori non contradictoires :

- ✓ "Un service représente certaines fonctionnalités (application fonctionnelle, transaction commerciale, un service du système de base, etc.) exposées sous la forme d'un composant au sein d'un processus métier." [Dodani, 2004],
- ✓ "Service : le moyen par lequel un fournisseur regroupe ses savoir-faire pour répondre aux besoins d'un client." [MacKenzie et al., 2006],
- ✓ "Un service, dans le cadre des architectures orientées services, expose une partie de la fonctionnalité fournie par l'architecture et respecte trois propriétés :
  (1) le contrat du service est exposé dans une interface indépendante de toute plate-forme,
  (2) le service peut être dynamiquement localisé et invoqué,
  (3) le service est autonome et sait maintenir son propre état courant." [Hashimi, 2003].

  De ces trois définitions, nous ressortons une idée principale, à savoir qu'un service permet d'exposer une ou plusieurs fonctionnalités, offertes par un fournisseur, à des clients potentiels. Indépendamment de ces définitions, plusieurs caractéristiques se distinguent [Collet, 2006] :
- ✓ Les interfaces et les données exhibées par les services sont exprimées en termes métiers (propres à un domaine d'application),
- ✓ Les aspects technologiques ne sont plus essentiels car les services sont autonomes, c'est à dire qu'ils sont indépendants du contexte d'utilisation ainsi que de l'état des autres services, et qu'ils interopèrent via des protocoles standardisés,

- ✓ Un service définit une entité (ressource, application, module, composant logiciel, etc.) qui communique via un échange de messages et qui expose un contrat d'utilisation. De façon similaire aux approches par objet ou par composant, l'approche par service cherche à fournir un niveau d'abstraction encore supérieur, en encapsulant des fonctionnalités et en permettant la réutilisation de services déjà existants. Comment faire interagir ces services au sein d'un environnement ?. On parle alors d'architecture orientée service.

II.4.2 Le modèle de référence OASIS

Le consortium OASIS [MacKenzie et al., 2006] travaille sur un modèle de référence pour les architectures orientées service. Cet effort a pour objectif de définir un cadre conceptuel commun qui pourra être utilisé de façon consistante à travers les différentes implémentations. Ce cadre conceptuel a pour but [Frédéric ; 2007]:

- ✓ D'établir les définitions qui devraient s'appliquer à toutes les architectures orientées service,
- ✓ D'unifier les concepts afin de pouvoir expliquer les patrons de conception génériques sous-jacents,
- ✓ De donner une sémantique qui pourra être utilisée de façon non ambigüe lors de la modélisation de solutions données.

Le consortium OASIS définit une architecture orientée service comme étant un paradigme permettant d'organiser et d'utiliser des savoir-faire distribués, pouvant être de domaines variés. Plus précisément, le modèle de référence OASIS repose sur les notions de besoins et de capacités. Ainsi, des personnes ou des organisations créent des capacités afin de résoudre et d'apporter une solution à leurs besoins. Les besoins d'une personne peuvent être adressés par les capacités offertes par une autre personne. Il n'y a pas de relation une à une entre les besoins et les capacités. Un besoin donné peut nécessiter de combiner plusieurs capacités, et une capacité peut répondre à plusieurs besoins. Une architecture orientée service offre donc un cadre pour faire correspondre des besoins à des capacités et pour combiner des capacités pour répondre à ces besoins.

Dans le modèle de référence OASIS le concept central est évidemment le service. Le modèle de référence souligne que cette notion de service correspond implicitement soit :

- ✓ à la capacité d'effectuer une tâche pour un tiers,
- ✓ à la spécification de la tâche qui est offerte par un tiers,
- ✓ à l'offre d'effectuer une tâche pour un tiers.

La personne qui offre un service est un fournisseur de service et celui qui l'utilise un consommateur. De plus, un service est considéré comme étant opaque dans le sens où son implémentation est cachée au consommateur. En plus du concept de service, le modèle OASIS identifie des concepts liés aux services eux-mêmes. Ces concepts regroupent notamment la

description de service, ainsi que les contrats et les politiques qui sont liés aux services, aux fournisseurs et aux consommateurs de services.

Une description de service représente les informations nécessaires afin d'utiliser un service et facilite la visibilité et l'interaction entre les consommateurs et fournisseurs de services. Le modèle de référence d'OASIS précise qu'il n'existe pas qu'une seule "bonne" description. Les éléments d'une description dépendent du contexte et des besoins des différentes parties impliquées. De façon générale, une description de service doit au moins spécifier les informations nécessaires afin qu'un consommateur puisse décider d'utiliser ou non un service. Ainsi le consommateur a besoin de savoir [Frédéric ; 2007] :

- ✓ que le service existe et est accessible,
- ✓ que le service effectue une fonction donnée ou un ensemble de fonctions,
- ✓ que le service fonctionne selon un ensemble de contraintes et politiques données,
- ✓ que le service sera en accord avec les contraintes imposées par le consommateur,
- ✓ comment interagir avec le service, ce qui inclut le format et le contenu des informations échangées ainsi que la séquence des informations échangées qui doit être respectée.

II.4.3 Les architectures orientées service Web

II.4.3.1 Les services web

Les SOA sont un ensemble de composants qui peuvent être appelés, et dont les descriptions d'interfaces peuvent être éditées et découvertes. De nos jours, les Services Web fournissent les technologies les plus adaptées pour rendre possible la création d'architectures orientées services.

Selon (W3C) : Un service Web est un composant logiciel identifié par une URI, dont les interfaces publiques sont définies et appelées en XML. Sa définition peut être découverte par d'autres systèmes logiciels. Les services Web peuvent interagir entre eux d'une manière prescrite par leurs définitions, en utilisant des messages XML portés par les protocoles Internet.

Ces architectures reposent sur l'utilisation d'un ensemble de Services Web ayant les caractéristiques suivantes [Fay, 2010] :

- ✓ **Orienté message :** la communication entre les services web définie en termes d'échange de messages.
- ✓ **Orienté description :** un service est décrit par des métadonnées.
- ✓ **Granularité :** les services communiquent en utilisant un nombre réduit de messages qui sont généralement grands et complexes.

- ✓ **Orienté réseau :** les services ont tendance à être utilisés sur un réseau. Cependant, ceci n'est pas une exigence absolue.
- ✓ **Dynamicité :** liée au fait que les Services Web reposent sur des communications entre un client et un service. Si la localisation d'un service est souvent stable (pour permettre de le localiser sur le long terme), il n'en est pas de même pour le client, car celui-ci est souvent connecté à travers Internet par un fournisseur d'accès ne lui attribuant pas une adresse IP fixe par exemple.
- ✓ **Interopérabilité :** présente à plusieurs niveaux. Tout d'abord, entre les services eux-mêmes : en effet, rien n'oblige à utiliser la même plateforme entre les différents services formant une application à part entière. Ensuite, au niveau des clients et des services : l'architecture logicielle et l'architecture matérielle peuvent être totalement différentes, l'essentiel est la mise à disposition des protocoles des Services Web sur ces architectures, indépendamment du langage et des logiciels utilisés.
- ✓ **Neutralité de la plate-forme :** les services communiquent en utilisant des messages codifiés dans une représentation indépendante de la plate-forme. Par exemple, CORBA utilise CDR (Common Data Representation) comme une représentation de données indépendante de la plate-forme ; les Services Web utilisent eux, XML.qui a l'avantage d'être extensible et de mieux représenter les informations.
- ✓ **Existence de langages spécifiques de descriptions comportementales :** Ce type d'application, étant dynamique et ne reposant pas sur un serveur centralisé, requiert une description du comportement de l'application complète, indiquant comment les communications vont se réaliser entre les différents services. Ce type de langage hérite des travaux réalisés dans la gestion des workflows, permettant d'indiquer les différents traitements effectués sur un ensemble de données pour un acteur donné.

La Figure II.3 illustre les éléments fondamentaux de cette architecture. Un Service Provider contient les services. Ces services sont décrits à travers une représentation (WSDL) utilisant des métadonnées, c.-à-d. la description du service. Ensuite, Le Service Provider enregistre les informations (Documents WSDL) de ses services dans l'annuaire. Un Service Requester cherche dans l'annuaire des services selon les critères spécifiques. L'annuaire retourne au les informations d'un service demandé. Le Service Requester récupère les métadonnées de ce service et les utilise pour échanger des messages avec le service.

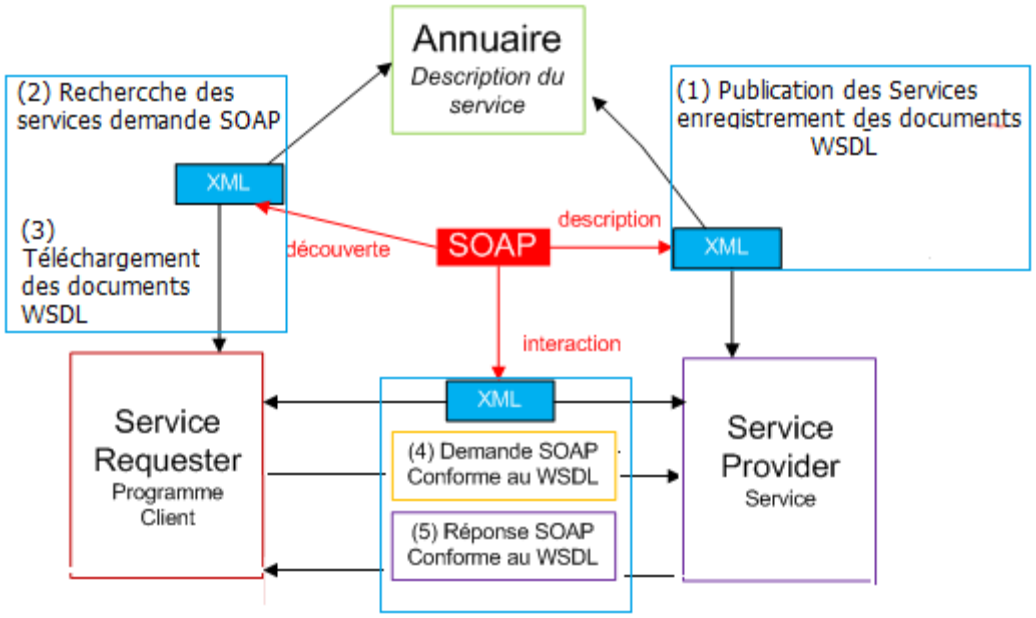

Figure II.3 Schéma générale d'une WSOA

Les Services Web consistent à exposer sur un réseau (et donc Internet), une ou plusieurs applications répondant à certains impératifs technologiques. Ces services peuvent proposer des fonctions très simples (du type requête/réponse) ou un ensemble complet d'outils, permettant d'aller jusqu'à la composition des services pour proposer une application complète.

Le client d'un service, qui peut être une personne ou un programme informatique, est situé sur la même machine ou sur une machine distante. Il devra, lui aussi, respecter certains impératifs permettant de répondre au problème principal d'interopérabilité. Cette interopérabilité concerne non seulement le codage des données, mais également les plateformes : le client n'est pas obligé de connaître la plateforme utilisée par le fournisseur de services pour communiquer avec celui-ci. La communication entre le client et le service passe par une phase de découverte et de localisation du service, à l'aide du protocole et des annuaires UDDI. Cet annuaire contient un ensemble de fichiers de descriptions de Services Web, utilisant le langage WSDL, basé sur XML. Le client interroge l'annuaire à l'aide de mots clés pour obtenir un ensemble de descriptions WSDL. Ces descriptions WSDL contiennent toutes les informations nécessaires à l'invocation du service (URL de localisation, description des fonctions et des types de données). Les communications entre les différents acteurs (service, client, annuaire) utilisent le protocole SOAP, également basé sur XML. Elles utilisent aussi les protocoles réseaux tel que HTTP pour faire parvenir les messages, permettant ainsi de s'affranchir des différents problèmes de pare-feux liés aux architectures réseaux [Fay, 2010].

L'usage effectif des services impose des contraintes et présente certaines caractéristiques :

- **À grain épais** (Coarse-grained) : Les opérations (fonctionnalités) dans les services sont fréquemment implémentées pour englober plusieurs fonctionnalités et exploitent de grands

ensembles de données par comparaison avec les composants à "grain fin" constituant des interfaces orientées objet, par exemple.

- **Conception basée sur les interfaces** : Différents services implémentent des interfaces communes ou bien un même service peut implémenter plusieurs interfaces.
- **Découvrables :** Les services doivent être découverts lors de la conception, mais aussi lors de l'exécution (run-time). Les services son identifiés tout autant par leur identifiant, mais aussi par leur interfaces et le type de services qu'ils procurent.
- **Faiblement couplés :** Les services sont connectés aux autres services et aux clients en utilisant des méthodes standards, basées sur les échanges de messages XML par exemple, qui privilégient la réduction des dépendances.
- **Asynchrones** : En général, les services utilisent une approche basée sur les échanges asynchrones de messages ; cependant cette règle n'est pas générale et des services peuvent bien utiliser une approche synchrone. L'ensemble de ces critères différencie une application basée sur les services, d'une application développée en utilisant les architectures à composants telles que CORBA, DCOM, J2EE ou .NET classique. Il est avantageux, par exemple de faire des requêtes asynchrones, pour réduire le temps d'attente du requérant. Lors d'un appel asynchrone, le requérant peut continuer son évolution tout en laissant le temps au fournisseur de lui fournir la réponse à sa requête. Dans certains cas, bien entendu, le cas synchrone s'impose, mais le cas asynchrone est souvent plus avantageux surtout quand le coût des communications est élevé, ou quand la latence du réseau est imprévisible.

Cependant, il ne faut pas confondre les Services Web avec SOA. Les Services Web fournissent le support pour la description, la publication/recherche et l'infrastructure de communication pour les services, alors que l'architecture SOA décrit comment un système composé de services peut être construit [Fay, 2010].

II.4.3.2 Les différentes couches de l'architecture orientées service Web

Les architectures orientées service Web sont la déclinaison du paradigme des architectures orientées service, sur le Web. Les services Web [Papazoglou, 2004] ont été proposés initialement par IBM et Microsoft, puis en partie standardisés sous l'égide du W3C. Techniquement, les services Web se présentent comme des entités logicielles sans état, dont la caractéristique majeure est de promouvoir un couplage faible. Une architecture à base de services Web est une architecture en couches [Myerson, 2002] [Booth et al.] [Perrin, 2005]. Le groupe de travail "Web Service Architecture" (WSA) [Booth et al.] du W3C propose une vision de cette architecture, qui peut elle-même être raffinée [Perrin, 2005].

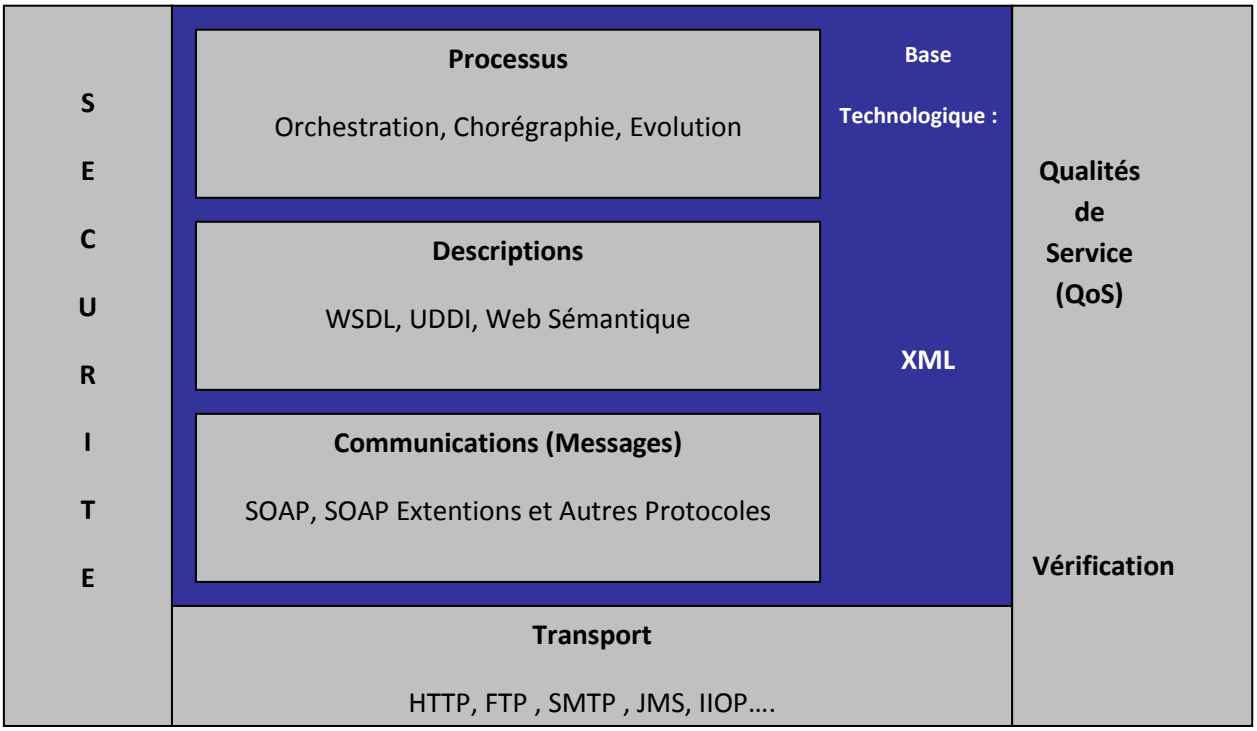

Figure II.4 Les couches d'une architecture orientée service Web.

II.4.3.2. 1. La couche Transport

Cette couche est responsable du transport des messages XML échangés entre les applications, c'est-à-dire les différentes communications. Il est souvent possible de spécifier le style, le mode et le protocole d'une communication. Actuellement, deux styles architecturaux totalement différents sont utilisés pour ces échanges de données. :

- ✓ le style RPC (Remote Procedure Call - appel de procédure à distance),
- ✓ le style Document (communication sous la forme de documents XML auto-descriptifs).

Trois modes de communication peuvent être envisagés :

- ✓ le mode RPC ou mode requête-réponse,
- ✓ le mode "one-way messaging" ou mode requête simple,
- ✓ le mode "asynchronous callback" ou mode requête-réponse asynchrone.

Actuellement, cette couche inclut HTTP, SMTP, FTP, JMS (Java Message Service), et de nouveaux protocoles tels que BEEP.

II.4.3.2. 2. La couche communication (messages)

Cette couche est responsable du formatage des données échangées de sorte que les messages peuvent être compris à chaque extrémité. La communication par messages constitue un point central dans toutes architectures orientées service web, afin de promouvoir un faible couplage, et ainsi couvrir les recommandations du modèle de référence OASIS. Les messages qui transitent au sein d'une architecture orientée service web sont, en général, basés sur XML [Bray et al., 2006] afin de permettre l'échange de données structurées, indépendamment des langages de programmation ou des systèmes d'exploitation. Les types de données utilisés sont eux aussi basés sur XML, c'est ce qu'on appelle l'encodage. Deux cas peuvent être distingués :

- ✓ "Literal" (suit littéralement les définitions de schéma XML - XML Schema [Fallside and Walmsley, 2004]),
- ✓ "SOAP encoded" (suit la spécification de SOAP [Gudgin et al., 2003]).

Actuellement, deux styles architecturaux totalement différents sont utilisés pour ces échanges de données. Nous avons d'un côté l'architecture orientée opérations distribuées (protocoles RPC) basée sur XML et qui comprend XML-RPC et SOAP et de l'autre côté une architecture orientée ressources Web, REST (Representational State Transfer) qui se base uniquement sur le bon usage des principes du Web (en particulier, le protocole HTTP).

SOAP

SOAP (Simple Object Access Protocol) [Gudgin et al., 2003] est un protocole d'invocation de méthodes sur des services distants. Leur objectif est d'assurer la communication entre machines. Le protocole permet d'appeler une méthode RPC et d'envoyer des messages aux machines distantes via HTTP. SOAP est une spécification XML [Fallside and Walmsley, 2004] qui définit un protocole léger d'échange de données structurées, entre un réseau d'applications, dans un environnement totalement distribué et hétérogène. Il est indépendant du contenu du message et laisse la responsabilité de l'interprétation aux couches d'abstractions supérieures. Ce protocole est très bien adapté à l'utilisation des services Web, car il permet de fournir au client une grande quantité d'informations récupérées sur un réseau de serveurs tiers. Il a aussi l'intérêt de pouvoir être employé dans tous les types de communication : synchrones ou asynchrones, point à point ou multi-point. Dans la pratique, le transfert est le plus souvent assuré via le protocole HTTP, cependant il peut aussi reposer sur d'autres protocoles.

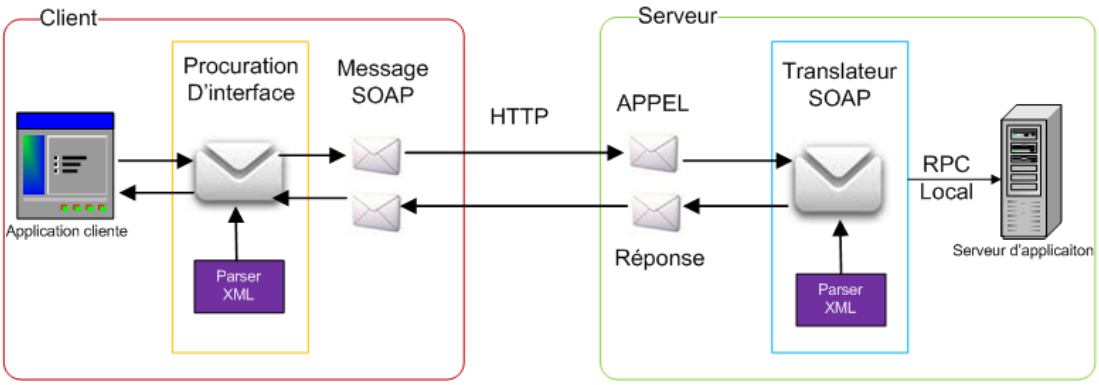

Figure II.5 Le protocole SOAP

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. 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 métier tels que :

- ✓ les appels de méthode, avec transferts de données spécifiques, dans le cadre d'une requêté (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).

SOAP est bien plus populaire et utilisé que XML-RPC. C'est une recommandation du W3C. D'après cette recommandation, SOAP est destiné à être un protocole léger dont le but est d'échanger des informations structurées dans un environnement décentralisé et distribué.

II.4.3.2. 3. La couche description

Dans le cadre des recommandations du modèle de référence OASIS, une description de service représente les informations nécessaires afin d'utiliser un service et facilite la visibilité et l'interaction entre les consommateurs et les fournisseurs de services [Frédéric, 2007].

Le protocole SOAP met à la disposition des services Web, un moyen standard de structuration et d'échange de messages XML. Il ne fournit en aucun cas une indication sur la structure que le message doit respecter vis à vis du service web sollicité. La spécification WSDL [Christensen et al., 2001] [Chinnici et al., 2007] a vu le jour afin d'offrir une grammaire qui décrit l'interface des services Web de manière générique. Ces deux standards, SOAP et WSDL, définissent ensemble l'aspect le plus basique du développement de l'infrastructure des services Web. Toutefois, 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 a été con¸cu pour réduire l'écart entre les applications clientes et les services Web, appelé UDDI [Clement et al., 2004].

WSDL

La spécification WSDL (Web Services Description Language) [Christensen et al., 2001] [Chinnici et al., 2007] présente les services comme des boîtes noires et s'intéresse à fournir une abstraction fonctionnelle du service. La spécification du service est composée de deux parties [Frédéric, 2007] :

- ✓ une définition abstraite des services en terme de messages échangés,
- ✓ la définition des mécanismes de liaison entre les définitions abstraites et un ensemble de techniques de déploiement (généralement des protocoles Internet).

La spécification WSDL joue un rôle important dans l'interopérabilité des services Web et permet de définir ce qui est nécessaire à leur invocation. En réalité, la notion d'invocation de service est un 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. La complétude de la spécification WSDL permet l'automatisation du processus d'invocation. En effet, elle contient toutes les informations nécessaires pour la mise au point d'un ensemble d'interfaces (API) qui génèrent automatiquement un programme client pour l'invocation d'un service Web. Par exemple, WSDL2Java [Graham et al., 2004] est un utilitaire qui génère un client Java pour invoquer un service Web et ce, à partir de sa description WSDL [Frédéric, 2007].

Le schéma suivant illustre la structure du langage WSDL qui est un document XML, en décrivant les relations entre les sections constituant un document WSDL.

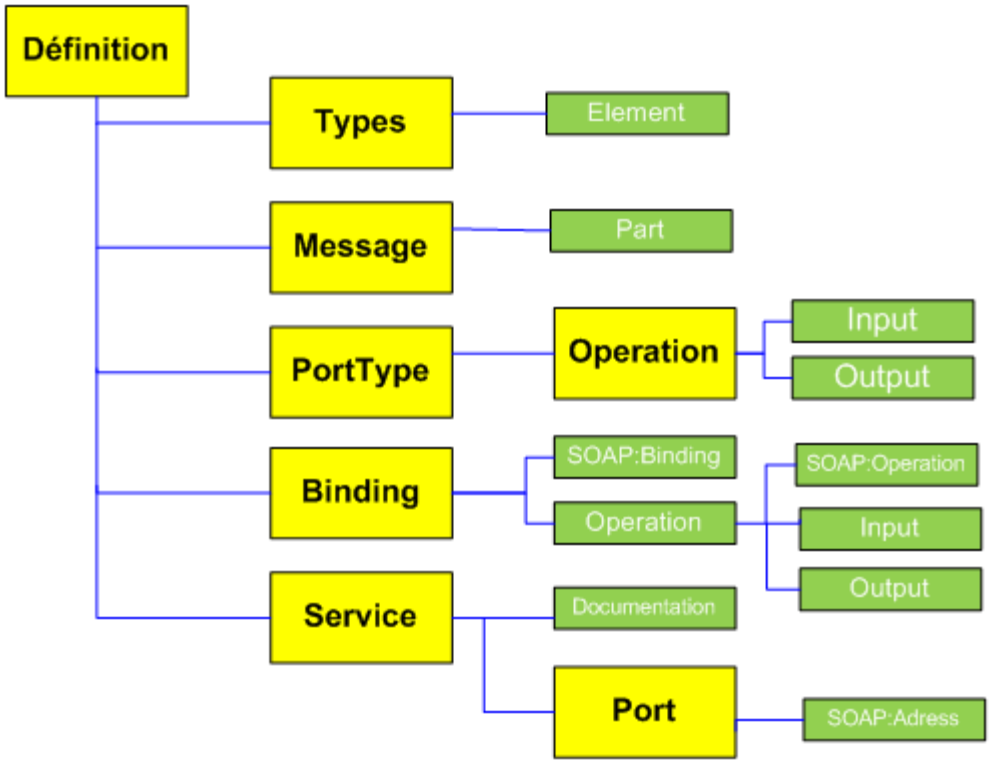

Figure II.6 Structure d'un document WSDL

Le document WSDL peut être divisé en deux parties. Une partie pour les définitions abstraites, tandis que la deuxième contient les descriptions concrètes.

La description concrète est composée des éléments qui sont orientés vers le client pour le service physique. Les trois éléments concrets XML présents dans un WSDL sont :

- ✓ <wsdl:service> ; **Service** : indique les adresses de port de chaque liaison.
- ✓ <wsdl:port> ; **Port** : représente un point d'accès de services défini par une adresse réseau et une liaison.
- ✓ <wsdl:binding> ; **Binding** : spécifie une liaison entre un <portType> et un protocole concret (SOAP, HTTP...).

La description abstraite est composée des éléments qui sont orientés vers la description des capacités du service Web. Ses éléments abstraits définissent les messages SOAP de façon totalement indépendante de la plate-forme et de la langue. Cela facilite la définition d'un ensemble de services pouvant être implémentés par différents sites Web. Les quatre éléments abstraits XML qui peuvent être définis dans un WSDL sont :

- ✓ <wsdl:types> ; **Types** : fournit la définition de types de données utilisés pour décrire les messages échangés.

- ✓ <wsdl:message> ; **Messages** : représente une définition abstraire (noms et types) des données en cours de transmission.
- ✓ <wsdl :operation>; **Opération** : c'est la description d'une action exposée dans le port.
- ✓ <wsdl :portType>; **PortTypes** : décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou d'erreurs.

UDDI

La spécification UDDI (Universal Description, Discovery and Integration) [Clement et al., 2004] constitue une norme pour les annuaires de services Web . Il fournit l'infrastructure de base pour la publication et la découverte des services Web. UDDI permet aux fournisseurs de présenter leurs services Web aux clients.

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. La spécification UDDI offre aussi une API aux applications clientes, pour consulter et extraire des données concernant un service et/ou son fournisseur. En effet, les registres UDDI sont eux-mêmes exposés comme des services Web. La mise en place d'un registre UDDI suit un processus uniforme imposé par la spécification. Chaque organisation qui veut mettre en place un registre UDDI doit suivre ce processus pour devenir un opérateur UDDI. Les registres UDDI créés sont organisés en réseaux, ils partagent ainsi les différentes informations publiées. La publication d'un service chez un opérateur donne automatiquement lieu à un processus de propagation des informations aux différents registres UDDI. L'accès à l'ensemble des informations des registres peut se faire de n'importe quel opérateur UDDI [Frédéric, 2007]. Chaque registre UDDI, les informations qu'il contient peuvent être séparées en trois types :

- ✓ les pages blanches qui incluent l'adresse, le contact et les identifiants relatifs au service Web ;
- ✓ les pages jaunes qui identifient les secteurs d'affaires relatifs au service Web ;
- ✓ les pages vertes qui donnent les informations techniques.

Le Web sémantique

WSDL fournit une vision très boîte noire d'un service Web (ses entrées et sorties), le Web sémantique permet un niveau de description plus introspectif. En effet, en plus des informations fonctionnelles fournies par WSDL, le Web sémantique permet de modéliser les pré et post conditions d'utilisation d'un service Web et permet de formaliser les concepts du domaine auquel

un service se rapporte par le biais d'ontologies [Srivastava and Koehler, 2003]. Dans ce cadre, OWL-S [Martin et al., 2004a], fondé sur les bases de DAML-S [Ankolenkar et al., 2001] [Ankolenkar et al., 2002], fait office de standard.

OWL-S décrit trois ontologies de haut niveau, dédiées aux services Web, et permettant de répondre à trois questions :

- ✓ "Que fait le service ?", décrite par l'ontologie Service Profile,
- ✓ "Comment fonctionne le service ?", décrite par l'ontologie Service Model,
- ✓ "Comment accéder le service ?", décrite par l'ontologie Service Grounding.

L'ontologie Service Profile [Martin et al. 2004a] a pour but de décrire les fonctionnalités d'un service web en termes de paramètres d'entrée et de sortie et en termes d'actions effectuées. Des informations additionnelles peuvent être décrites comme, par exemple, la catégorie du service web (selon une classification commune), les paramètres restreignant son utilisation, l'estimation de la qualité du service web, etc., afin de fournir une aide à l'interprétation.

L'ontologie Service Model [Martin et al. 2004a] a pour but de fournir une description spécifique et détaillée de la séquence des actions que le producteur du service va suivre lors de l'interaction avec un consommateur potentiel. Cette description offre une certaine autonomie au consommateur du service Web, qui peut en effet déduire facilement le protocole d'interaction à utiliser ainsi que les conséquences concrètes de chaque échange de message.

L'ontologie Service Grounding [Martin et al., 2004a] a pour but de décrire les différents modes d'accès d'un service en termes de protocole de communication, de formats de message et de détails plus spécifiques, comme par exemple, le numéro du port utilisé pour l'invocation du service Web. En ce sens, cette ontologie reprend les différentes informations fournies par le description WSDL.

Il est à noter que OWL-S [Martin et al., 2004a] est uniquement un langage de description, qui par conséquent n'implique aucune forme d'exécution. Il en est de même pour les différents travaux [Martin et al., 2004b] du domaine du Web sémantique. En effet, plusieurs approches de planification [McDermott, 2002] [McIlraith and Son, 2002] [Nau et al., 2003] [Wu et al., 2003], issues du domaine de l'intelligence artificielle, s'intéressent à composer automatiquement des services Web selon un très haut niveau d'abstraction. Ces travaux permettent d'étudier la description sémantique des services Web afin d'enchaîner ces services pour atteindre un but, lui aussi défini sémantiquement, indépendamment des détails liés à l'exécution elle-même [Frédéric, 2007].

II.4.3.2. 4 La couche qualité de service

Il paraît indispensable de voir émerger des caractéristiques et des métriques qui portent sur la qualité des services Web et des architectures orientées service Web. Une spécification pour la qualité de service a été proposée : WS-QoS [KangChan et al.] met en évidence les différentes composantes de la qualité, comme par exemple la performance, la disponibilité, la mise à l'échelle, la robustesse, l'intégrité, l'accessibilité, etc. Plusieurs travaux ont pour objectif d'exprimer et de mesurer l'une ou l'autre de ces composantes [Krishnaswamy et al., a] [Kalepu and Loke] [Krishnaswamy et al., b] [Tao et al.], et de regrouper ces mesures sous la forme de registres [Liu et al.] afin de permettre la sélection d'un service Web en fonction des mesures précédentes.

II.4.3.2. 5. La sécurité

Compte tenu du grand volume d'informations, et de la confidentialité et du secret d'une grande partie entre eux, la notion de sécurité devient un point important au sein des architectures orientées service Web [Mysore, 2003] [Chanliau, 2006]. Plusieurs aspects sont alors à prendre en compte, comme par exemple :

- ✓ l'identification d'un client et/ou d'un fournisseur,
- ✓ l'encription des données,
- ✓ l'intégrité des données,
- ✓ la délégation d'identité,
- ✓ le contrôle d'accès.

Au début des services Web, leurs fournisseurs considéraient que la sécurité serait entièrement gérée au niveau de la couche transport, au moyen de SSL/TLS (HTTPS), c'est pourquoi les standards initiaux (SOAP, WSDL, UDDI) n'abordaient pas la sécurité. Mais celle-ci a pris de l'importance dès lors que les services Web se sont multipliés. Une transaction de service Web passe souvent par de nombreuses mains, dont chacune a besoin d'accéder à certaines parties de la transaction mais pas à d'autres. Deux spécifications font aujourd'hui référence sur le plan de la sécurité au sein d'une architecture orientée service Web : WS-Security [Nadalin et al.] et WS-SecurityPolicy [Della-Libera et al.]. De l'authentification des utilisateurs et des composants en présence en passant par le chiffrement, et la gestion de l'intégrité des messages par le biais de certificats, WS-Security permet de définir la manière de décrire, au sein d'un message SOAP, les droits utilisateur correspondants aux systèmes en présence. Pour ce faire un message SOAP peut être complété par différentes assertions en utilisant par exemple les langages XML Signature [Bartel et al., 2002], XML Encryption [Imamura et al., 2002], SAML (Security Assertion Markup Language) [Ragouzis et al., 2006] ou encore XKMS (XML Key Management Specification) [Ford et al., 2001].

D'autre part, des travaux [Koshutanski and Massacci, 2003] [Imamura et al., 2005] [Nakamura et al., 2005] [Rouached and Godart, 2006] [Skogsrud et al., 2007] s'intéressent à la gestion de certaines politiques de sécurité au sein d'une composition de service Web. En effet, de manière formelle [Rouached and Godart, 2006] ou non [Koshutanski and Massacci, 2003] [Imamura et al., 2005] [Nakamura et al., 2005], la politique de gestion des accès (autorisations) aux services et aux ressources a été abordée sous cet angle. De même, des travaux [Skogsrud et al., 2007] spécifiques à la gestion de la politique de confiance et à la prise en compte de ses modifications potentielles tout au long du cycle de vie d'une orchestration sont menés [Frédéric, 2007].

II.4.3.2. 6. La couche processus

La composition de services Web est au cœur des architectures SOA puisque elle supporte la construction de services, dits composés ou composites, à partir de fonctionnalités de base. Elle a pour but la réutilisation des services (simples ou composés). L'exécution d'un service composé implique des interactions avec les services partenaires en faisant appel à leurs fonctionnalités. On peut distinguer deux méthodes de composition de services Web :

La chorégraphie

Qui décrit une collaboration entre services pour accomplir un certain objectif. Elle ne se repose pas sur un seul processus principal. La chorégraphie est, par nature, collaborative. Elle décrit les différents messages qui transitent entre les différents acteurs d'un processus (les services), l'identité de ceux-ci n'étant pas forcément encore connue. La chorégraphie permet la collaboration point-à-point entre plusieurs services Web. Et donne ainsi une vision abstraite des échanges au sein d'un processus. Elle ne permet en aucun cas une exécution, mais sert cependant de première spécification au processus concret (orchestration) à réaliser. La chorégraphie est une coordination décentralisée où chaque participant est responsable d'une partie du workflow, et connaît sa propre part dans le flux de messages échangés. Elle permet la modélisation d'un point de vue global afin de prendre en compte des situations de concurrences dans les environnements distribués et ainsi donner une vue plus flexible.

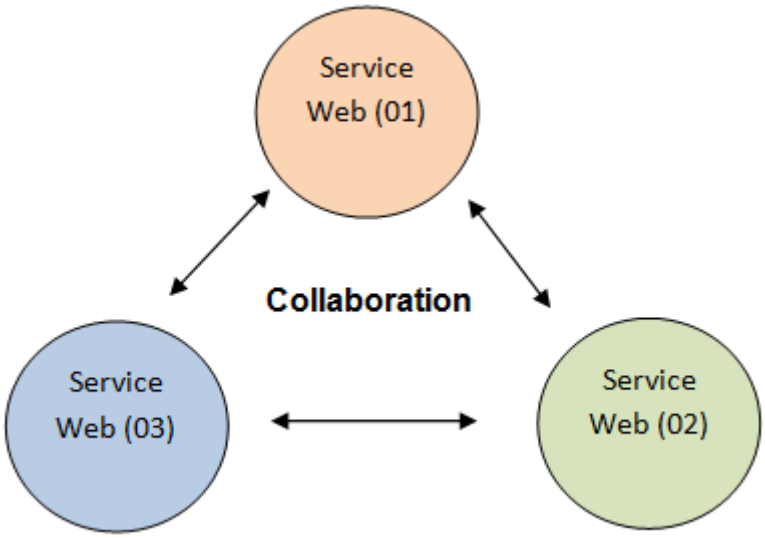

Figure II.7 La chorégraphie des services web

Actuellement, la chorégraphie de services Web repose essentiellement sur deux standards que sont les spécifications WSCI [Arkin et al., 2002] et WS-CDL [Kavantzas et al., 2005] :

La spécification WSCI (Web Service Choreography Interface)

Décrite par Sun, SAP, BEA et Intalio, L'interface de service Web Chorégraphie (WSCI) est une description de l'interface langage basé sur XML qui décrit le flux de messages échangés par un service Web participer à des interactions chorégraphiés avec d'autres services.

WSCI fonctionne en conjonction avec le Web Service Description Language (WSDL), la base pour le Groupe de travail Web Services Description W3C; il peut, aussi, travailler avec un autre langage de définition de service qui présente les mêmes caractéristiques que WSDL.

WSCI décrit le comportement observable d'un service Web. Ceci est exprimé en termes de dépendances temporelles et logiques entre les messages échangés, avec des règles de séquençage, la corrélation, la gestion des exceptions et les transactions. Cette description permet aux développeurs, des architectes et des outils pour décrire et composer une vue globale de la dynamique de l'échange de messages par la compréhension des interactions avec le service web [W3C., 2002].

La spécification WS-CDL (Web Services Choreography Description Language)

Le Web Services Chorégraphie Description Language (WS-CDL) [Kavantzas et al., 2005] est un langage basé sur XML qui décrit les collaborations point-à-point des participants, à partir d'un point de vue de comportement global; où les échange des messages aboutissent à atteindre un

objectif commune [W3C., 2005] . Les spécifications de services Web offrent un pont de communication entre les environnements informatiques hétérogènes utilisés. L'avenir des applications e-business exige la capacité d'effectuer à long terme, des collaborations point-à-point entre les services participants.

L'orchestration

Qui décrit la manière dont les services Web peuvent interagir ensemble au niveau des messages, incluant l'ordre d'exécution des interactions (des messages) (souvent qualifiée de "logique métier"). L'orchestration décrit quant à elle comment les services Web peuvent interagir entre eux selon une perspective opérationnelle, avec des structures de contrôle. Dans l'orchestration, un seul processus, appelé orchestrateur, est responsable de la composition et contrôle les interactions entre les différents services. Cet orchestrateur coordonne de manière centralisée les différentes opérations des services partenaires qui n'ont aucune connaissance de cette composition. L'identité des services Web est alors connue. L'orchestration donne une vision concrète qui permet l'expression d'un processus exécutable.

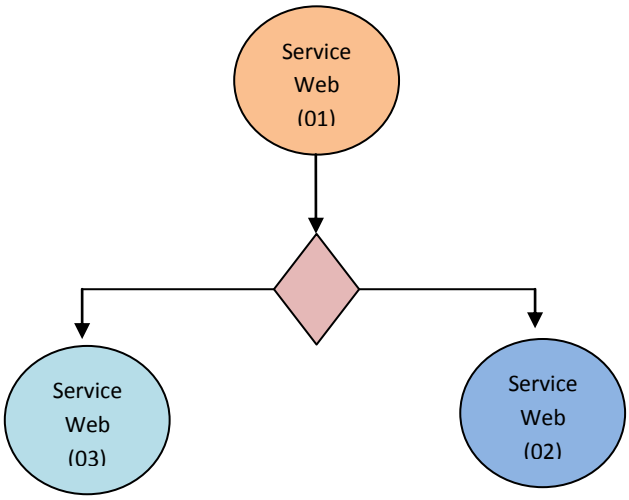

Figure II.8 L'orchestration des services web

Un Service Web est un regroupement logique d'une ou plusieurs opérations : dans la pratique ce n'est pas stricto sensu un service Web qui est invoqué mais une opération de ce service Web. On distingue dès lors deux types d'opérations de service Web :

- ✓ les opérations atomiques : Une opération atomique se suffit à elle-même dans le sens où elle n'a pas besoin d'invoquer d'autre opération de service Web au sein de son comportement interne.
- ✓ les opérations complexes : une opération complexe invoque d'autres opérations (du même et/ou d'autres services Web) au sein de son propre comportement.

La différenciation entre ces deux types d'opérations réside dans la présence ou l'absence de mémoire. La mémoire désignée ici correspond à une mémoire d'interaction. L'invocation d'une opération atomique n'excède pas un simple échange synchrone ou asynchrone de deux messages.

Par contre, une opération complexe est dotée d'un comportement défini par le séquencement d'échanges de messages avec diverses autres opérations (atomiques ou complexes). Une opération complexe peut elle même, être vue comme le résultat d'une orchestration. En d'autres termes, le processus exécutable décrit par le biais d'une orchestration peut être déployé comme une opération de service Web et ainsi faciliter son utilisation et sa réutilisation au sein d'autres orchestrations. Le nombre de niveaux d'imbrication est alors illimité.
Plusieurs travaux se sont succédé afin de fournir des langages appropriés à la description de tels processus [Frédéric, 2007] :

Microsoft a initialement développé la spécification XLANG [Thatte, 2001]

Associée à son serveur BizTalk. Cette spécification était dédiée à la création de processus métier et aux interactions entre fournisseurs de services Web. XLANG supportait le séquencement, la mise en parallèle ou encore les branchements conditionnels pour la gestion du flux de contrôle des processus métier. Un robuste système de gestion des exceptions, supportant les transactions longues durées, était aussi fourni. Enfin, XLANG utilisait WSDL pour décrire l'interface des différents services Web utilisés au sein d'un processus.

WSFL (Web Service Flow Language) [Leymann, 2001]

IBM a, quant à elle, proposé WSFL pour décrire à la fois le modèle de flux et le modèle global d'un processus métier. Le modèle de flux correspondait au séquencement des activités du processus alors que le modèle global permettait la corrélation entre une activité et une instance d'opération de service Web. Une description WSFL pouvait enfin être exposée par le biais d'une interface WSDL afin de permettre une décomposition récursive d'un processus (utilisation d'une description WSFL comme activité d'une autre description WSFL).

BPML (Business Process Management Language)

Est quant à lui un méta langage de description de processus métier. BPML est l'initiative de BPMI (Business Process Management Initiative - BPMI.org) et de différents partenaires dont entre autres Sun et Intalio. Ce langage supporte, d`es sa première version, les descriptions issues de WSCI. En effet, WSCI peut être utilisé pour décrire les interactions publiques (la chorégraphie) alors que l'implémentation privée (l'orchestration) peut être décrite par BPML. De plus, WSCI et BPML fournissent le même modèle d'exécution de processus et une syntaxe similaire. Au même titre que BPEL4WS, BPML supporte la notion de rôle et des mécanismes de gestion des transactions et des exceptions [Frédéric, 2007].

Le langage BPEL4SW

BPEL, tout d'abord nommé BPEL4WS [IBM, 2003] et WS-BPEL [OASIS, 2007], s'est imposé comme le langage standard OASIS d'orchestration de services Web. Il est largement utilisé dans le cadre de la mise en oeuvre d'une architecture orientée services. BPEL4WS fournit une grammaire, basée sur XML, il permet la modélisation du comportement des services Web au sein des interactions d'un processus métier [Weerawarana and Francisco, 2002]. La grammaire BPEL4SW peut ensuite être interprétée et exécutée par un moteur d'orchestration, contrôlé par l'un des participants du processus. Ce moteur coordonne les activités du processus et fournit des mécanismes de compensation en cas d'erreur.

BPEL4WS est essentiellement une couche au dessus du langage WSDL. En effet, WSDL permet la description des différentes opérations de services Web alors que BPEL4WS permet la description de leur séquencement. De plus, BPEL4WS utilise WSDL de trois manières :

- ✓ tout processus BPEL4WS est exposé comme un service Web. De fait, l'interface publique du processus (le type et le nom de ses paramètres d'entrée/sortie) est décrite avec WSDL,
- ✓ les types de données WSDL sont utilisés par BPEL4WS pour décrire les informations qui transitent entre les différentes activités d'un processus,
- ✓ WSDL peut être utilisé pour référencer les différents services Web requis par le processus.

Les conteneurs et les partenaires sont deux autres notions importantes de BPEL4WS.

Un conteneur identifie une donnée échangée au cours d'un processus et est typé sur le même schéma qu'un message WSDL. Lorsqu'un processus BPEL4WS reçoit un message, le conteneur approprié est mis à jour pour permettre de retrouver cette donnée (le message) par la suite.

Un partenaire peut être n'importe quelle opération de service invoquée au cours du processus mais aussi n'importe quelle opération de service invoquant le processus lui même. Chaque partenaire est associé à un rôle, ce dernier pouvant être différent d'un processus à l'autre [Frédéric, 2007].

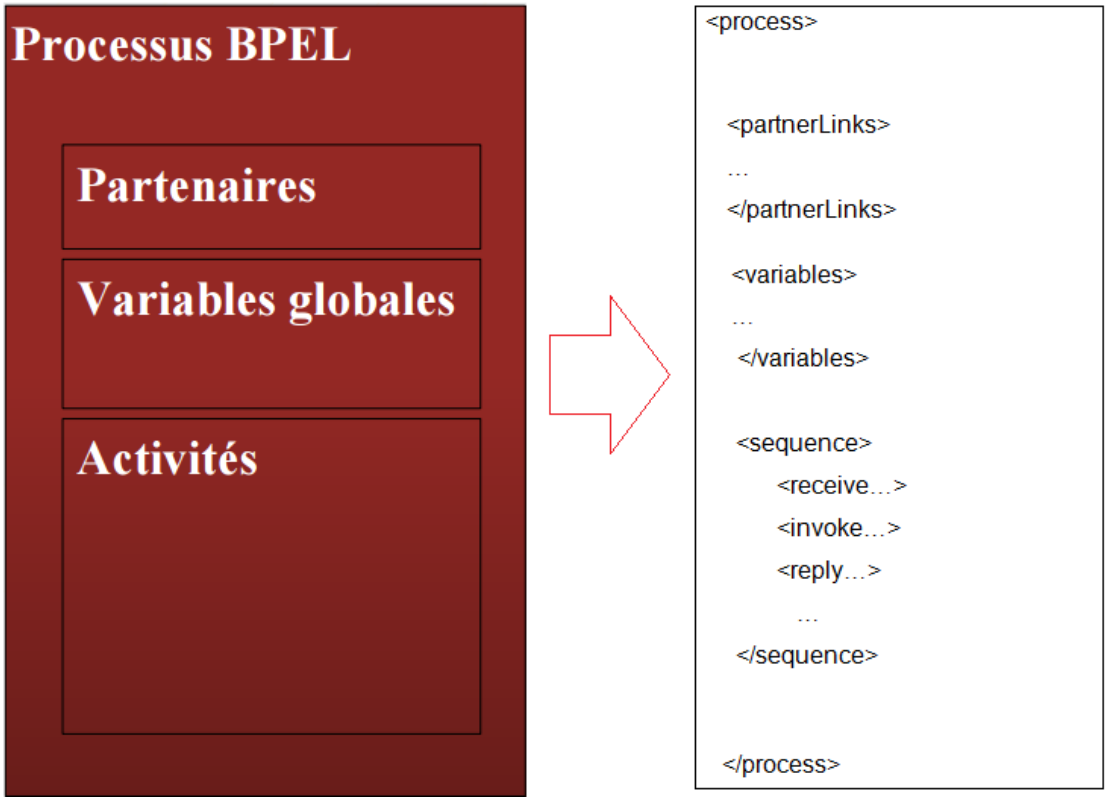

Figure II.9 Processus BPEL4SW

Le langage BPEL permet de décrire à la fois :

- ✓ ***l'interface comportementale***, via la définition d'un processus abstrait spécifiant les échanges de messages entre différents partenaires ;
- ✓ ***l'orchestration***, via la définition d'un processus privé exécutable 3 qui représente la nature et l'ordre des échanges entre partenaires.

BPEL se caractérise par rapport aux autres langages (d'orchestration) par [LALLALI, 2009] :

- ✓ sa gestion des exceptions, en particulier, des fautes et des événements ;
- ✓ l'exécution parallèle des activités et la synchronisation des flots ;
- ✓ la description des transactions (est une suite d'actions dont l'exécution est cohérente, atomique, isolée des autres transactions et dont les effets sont durables) contextuelles (stateful) et de longue durée ;
- ✓ son mécanisme de compensation qui est très utile pour les transactions de longue durée ;
- ✓ sa gestion de la corrélation des messages.

Les activités (La partie dynamique de BPEL) : Le procédé dans BPEL est constitué d'activités liées par un flot de contrôle. Ces activités peuvent être basiques, structurées ou des communications :

A .Les activités basiques :

- ✓ <wait>, pour attendre un certain temps.
- ✓ <assign>, pour copier les données d'une place à l'autre.
- ✓ <throw>, pour lancer une erreur d'exécution.
- ✓ <terminate>, pour terminer l'instance de service en entier.
- ✓ <empty>, qui ne fait rien (utile pour la synchronisation des activités parallèles).

B. Les activités structurées : Les activités structurelles définissent l'ordre dans lequel les activités imbriquées sont exécutées,

- ✓ <sequence>, pour définir un ordre d'exécution.
- ✓ <switch>, pour l'acheminement conditionnel.
- ✓ <while>, pour les boucles.
- ✓ <pick>, pour attendre l'arrivée d'un événement
- ✓ <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.

C. Les activités de communication : Les activités de communication sont : <receive>, <reply>, et <invoke>. Elles sont utilisées pour l'échange des messages entre un processus BPEL, son client et ses partenaires.

- ✓ <invoke>, pour invoquer une opération dans un service partenaire. Elle peut servir à la fois pour une communication synchrone ou asynchrone. L'activité <invoke> asynchrone joue le même rôle qu'une activité <reply> ; tandis que l'activité <invoke> synchrone invoque une opération d'un service partenaire mais attend sa réponse.
- ✓ <receive>, pour attendre un message d'une source externe. Elle peut créer une instance d'un processus BPEL et le préparer ainsi à l'arrivée d'une réponse.
- ✓ <reply>, pour répondre à une source externe.

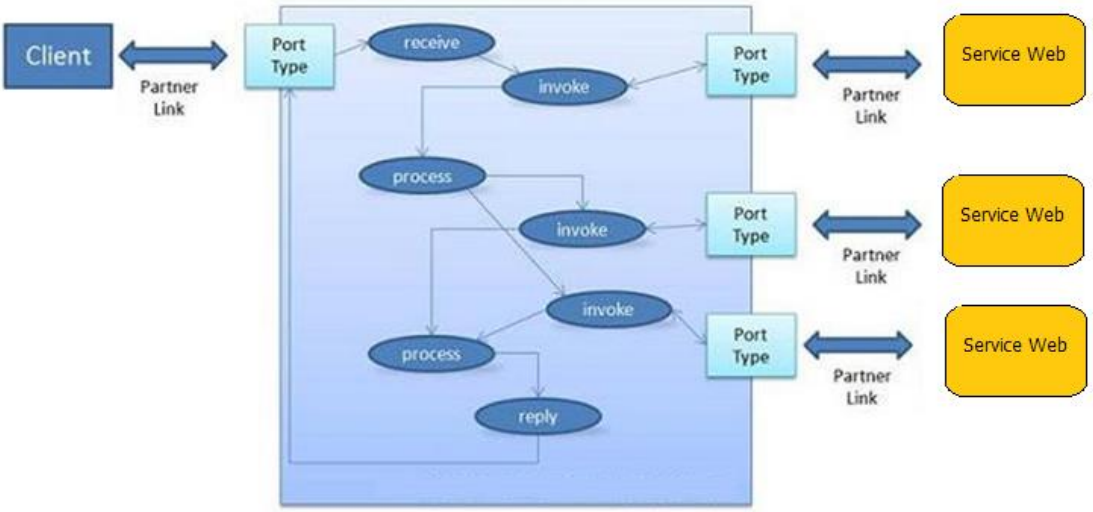

Figure II.10 Les activités dans un processus BPEL4SW

D'autre part, l'évolution dynamique (en cours d'exécution) d'une composition de service Web a été identifiée comme un réel challenge [Papazoglou, 2003b] [Papazoglou et al., 2006]. Certains travaux [Belhajjame et al., 2001] [Belhajjame et al., 2003] portent sur l'adaptation des compositions de services en fonction du contexte d'exécution.

II.5. Les bases technologiques du Web

Alain Lefebvre, co-fondateur de la SSII SQLI et à l'origine de la création du service de réseaux sociaux synergies déclare que : « Le Web 2.0 peut se définir comme la combinaison entre l'interface technologique qu'est Ajax, d'une part, et une vocation à placer l'utilisateur au centre du service Web, d'autre part ». Le Web 2.0 souhaite renforcer l'interactivité et l'ergonomie des services Internet et s'appuie pleinement pour y parvenir sur la technologie AJAX.

II.5.1 Représentation de technique de l'approche AJAX (Asynchronous JavaScript And XML)

Le terme Ajax a été introduit par Jesse James Garrett (informaticien américain), le 18 février 2005, dans un article sur le site Web Adaptive Path5. Depuis, il a rapidement gagné en popularité. En informatique, et plus particulièrement en architecture informatique, Ajax (acronyme de Asynchronous Javascript and XML) est une manière de construire des applications Web et des sites web dynamiques basés sur diverses technologies ajoutées aux navigateurs web entre 1995 et 2005. Ajax est la combinaison de technologies telles que Javascript, CSS, XML, le DOM et le XMLHttpRequest dans le but de réaliser des applications Web qui offrent une maniabilité et un

confort d'utilisation supérieur à ce qui se faisait jusqu'alors - les Rich Internet Application [Michael., 2006] [Luc Van., 2007] :

- ✓ Le Modèle Objet Document (DOM) pour l'affichage dynamique et pour modifier l'information présentée dans le navigateur par programmation.
- ✓ L'objet XMLHttpRequest est utilisé pour dialoguer de manière asynchrone avec le serveur Web.
- ✓ La notation XML et XSLT est utilisée pour structurer les informations transmises entre le serveur Web et le navigateur. En alternative au format XML, les applications Ajax peuvent utiliser les fichiers texte ou JSON.
- ✓ JavaScript pour les mettre en œuvre ensemble.

Les applications Ajax fonctionnent sur tous les navigateurs Web qui mettent en œuvre les technologies décrites précédemment, parmi lesquels Mozilla Firefox, Internet Explorer, Konqueror, Google Chrome, Safari, Opera, etc...

Les fondements technologiques d'AJAX sont loin d'être particulièrement novateurs. Patrick Chassany, fondateur d'Amen et co-fondateur de plusieurs sociétés et services en ligne Web 2.0, dont Fotolia et EveryFeed souligne que : « Les technologies qui sont à la base du Web 2.0 ne sont pas récentes. C'est leur convergence qui est nouvelle, répondant au besoin accru d'interactivité des internautes, un besoin venu petit à petit, à mesure qu'ils passent sensiblement plus de temps sur Internet qu'avant ».

II.5.2 Le principe AJAX

Le modèle d'application web classique fonctionne comme ceci: La plupart des actions utilisateur dans l'interface déclenchent une requête HTTP à un serveur web. Le serveur accomplit quelques traitements :

- ✓ retrouver des données, traiter des nombres, communiquer avec divers systèmes hérités
- ✓ et ensuite retourne une page HTML au client (navigateur).

Celui-ci affichera alors la page qu'il vient de recevoir. Chaque manipulation entraîne la transmission et l'affichage d'une nouvelle page et l'utilisateur doit attendre l'arrivée de la réponse pour effectuer d'autres manipulations.

Une application Ajax élimine le coté marche-arrêt-marche-arrêt de l'interaction sur le Web en introduisant un intermédiaire - un moteur Ajax - entre l'utilisateur et le serveur. Au lieu de charger une page web, au départ de la session, le navigateur charge un moteur Ajax - écrit en JavaScript et habituellement encapsulé dans une frame cachée. Le moteur est responsable à la fois du rendu de

l'interface que voit l'utilisateur et de la communication avec le serveur au nom de l'utilisateur. Le moteur Ajax permet à l'interaction de l'utilisateur avec l'application de s'exercer asynchroniquement indépendamment de la communication avec le serveur. Ainsi l'utilisateur n'est jamais confronté à une fenêtre de navigateur blanche et à l'icône en forme de sablier, en attente parce que le serveur fait quelque chose [Jesse, 2006].

La méthode classique de dialogue utilise des mécanismes propres au World Wide Web, qui sont incorporés dans tous les navigateurs ainsi que les robots d'indexation, et ne nécessite pas de programmation. Au contraire, le fonctionnement d'Ajax nécessite de programmer les dialogues entre le navigateur et le serveur Web. Il nécessite également de programmer les modifications à effectuer dans la page Web, sans quoi les dialogues se font à l'insu de l'utilisateur [Bruno, 2009].

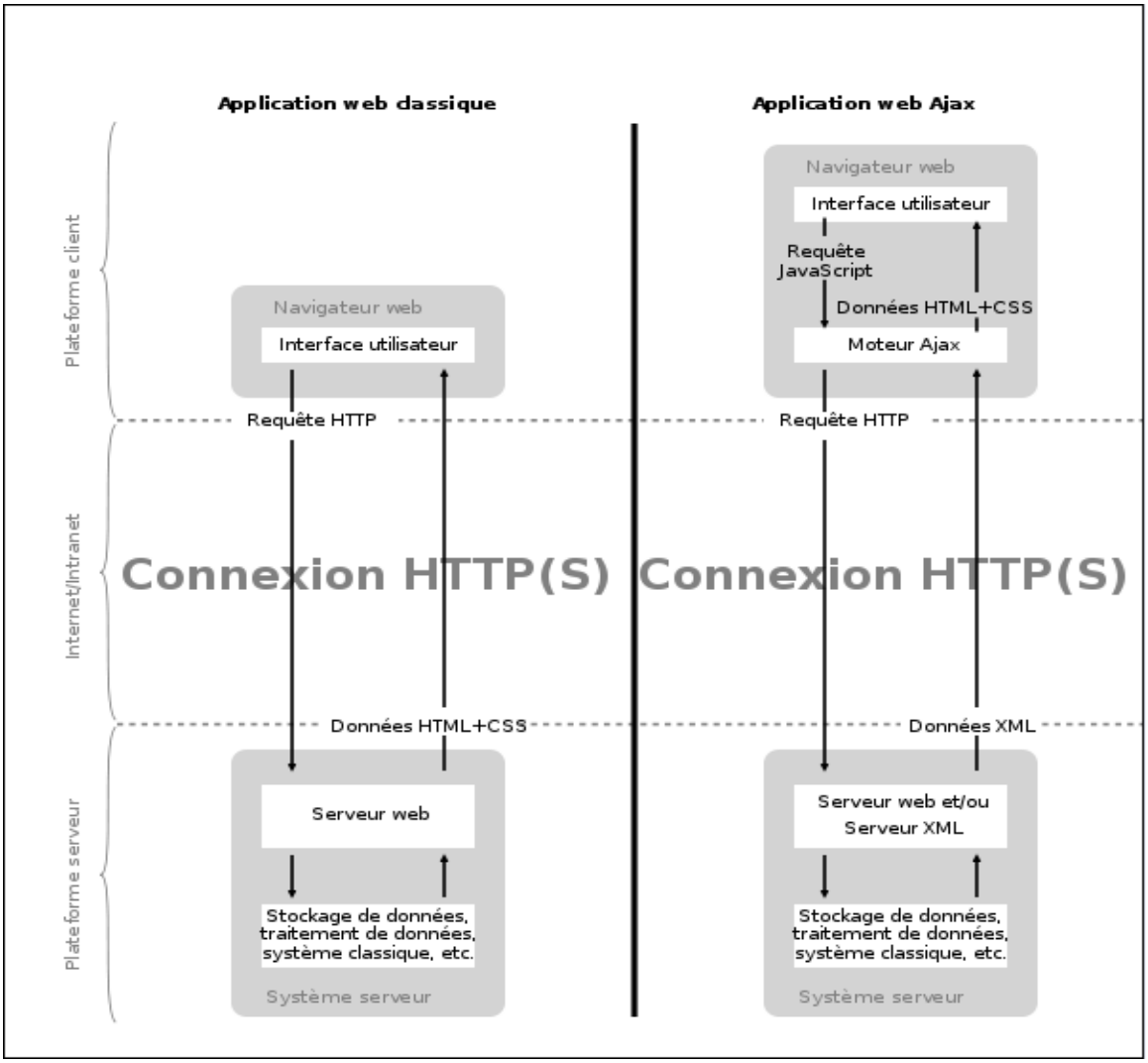

Figure II.11 Le principe AJAX

En Ajax, comme le nom l'indique, les demandes sont effectuées de manière asynchrone : le navigateur Web continue d'exécuter le programme Javascript qui envoie la demande dès que celle-ci est partie, il n'attend pas la réponse envoyée par le serveur Web et l'utilisateur peut continuer à effectuer des manipulations [Nathaniel, 2006] . La technologie Ajax évite un aller-retour de requêtes entre le poste client de l'utilisateur et le serveur du site Web qui soutient l'application ou le service. Elle n'implique plus l'installation de plug-in ou de programme spécifiques tels que des ActiveX ou du Flash.

Javascript fut le premier langage d'importance du côté client. Il était capable de faire exécuter du code coté client, de plus il était Implémenté dans la plupart des clients Web.

Bien que son usage soit relativement restreint de prime abord, combiné à plusieurs couches d'autres langages (i.e. DHTML) il est devenu possible de le faire cohabiter avec un système RIA sans utiliser une solution du type « client monolithique ». AJAX est maintenant le terme utilisé pour se référer à cette combinaison de techniques, elle est récemment devenue la plus importante.

II.5.3 Les frameworks Java WEB

Les frameworks Java WEB, ont longtemps privilégié l'utilisation de technologies de présentations dédiées (JSP, JSF, Struts, JSF, Spring, Hibernate..) exigeant l'apprentissage d'une grammaire spécifique, qui complexifient singulièrement les réalisations. Ils Après avoir essayé Wicket, on est forcément séduit par la facilité d'appréhension du paradigme proposé et la simplicité des développements qui en découle. On peut profiter de son approche orientée composants pour factoriser les traitements et observer la robustesse de l'architecture des applications Web 2.

Ces framewors sont basés sur une séparation des responsabilités entre les tiers techniques :

- ✓ La couche présentation est écrite en HTML (un attribut spécifique 'wicket:id= « idComposant »' permet l'identification des balises) ;
- ✓ La déclaration des composants purs en Java utilise une gestion des évènements proche de « Swing » ; On capture ainsi les modifications et soumissions de données avec une granularité fine ;
- ✓ La liaison avec les objets Modèles, support des informations manipulées (en faisant disparaitre au passage les formulaires).

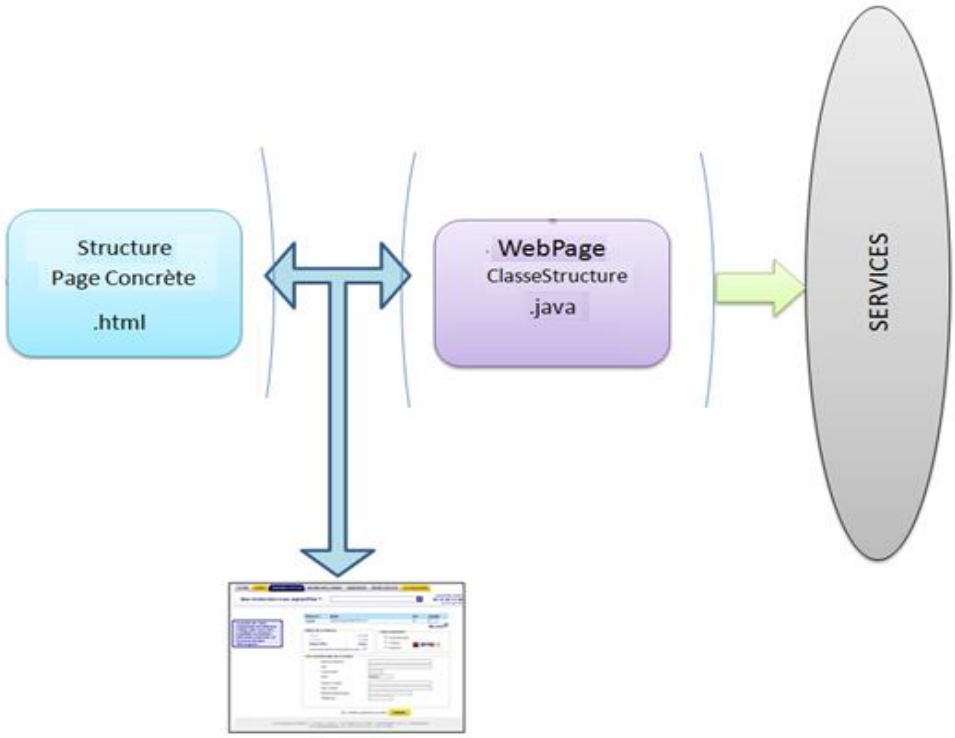

Figure II.12 les tiers techniques des applications web 2+

Pour se conformer à une approche par couches applicatives, on peut représenter le fonctionnement de Wicket comme suit :

- ✓ Rapprochement des pages HTML des classes du même nom, lesquelles supportent le traitement de l'information à présenter et la gestion des événements ;
- ✓ Utilisation du mécanisme d'héritage pour les pages partageant des éléments communs de structure ;
- ✓ Invocation des services (et de la persistance) ;
- ✓ Génération de la page finale.

II.5.4 Les caractéristiques de java

Le langage Java

Les variables, les opérateurs, les expressions, instructions, blocs, contrôle de flot sont très proches de ceux du C .

- ✓ Les exceptions sont une nouveauté
- ✓ Les types primitifs ont une taille et une représentation normée
- ✓ S'y ajoutent des spécificités syntaxiques liées à la programmation objet, aux classes, à l'héritage.

Java: un langage et une plateforme

Dans la technologie Java, on a donc besoin

- ✓ Du **langage de programmation** et du compilateur et plein de commandes bien utiles: jar, javap, javadoc, etc
- ✓ De la **JVM et** des **APIs** (*Application Programming Interfaces*) regroupées dans une « plateforme »:
- ✓ Java SE (Java Platform, Standard Edition): Java SE 6 pour applications classiques, desktop
- ✓ **Java EE** (Java Platform, Enterprise Edition): Java EE 6 pour développer et déployer des applications serveur, Web services, etc.
- ✓ **Java ME** (Java Platform, Micro Edition): J2ME pour les applications embarquées, PDA, téléphones, etc.
- ✓ Si on veut juste exécuter, il suffit du **JRE** (*Java Runtime Execution*) par opposition au **JDK** (*Java Developpement Kit*)

Classes et objets

Une classe « *Composant* » représente plusieurs choses:

- ✓ Un moule pour la création d'objets de type *Composant*
- ✓ Cela nécessite en général la définition d'un ensemble de champs (fields) décrivant **l'état** d'un objet de ce type et d'un ensemble de méthodes définissant son **comportement** ou ses fonctionnalités.

Chaque objet de la classe « *Composant* »

- ✓ Dispose de son propre état (la valeur de ses champs)
- ✓ Répond au même comportement (via les méthodes de la classe)

Java est réparti

Java a été construit avec Internet en tête. Riche bibiothèque pour

- ✓ l'accès aux URL (Universal Resource Locators),
- ✓ la programmation client/serveur via des sockets TCP et UDP,
- ✓ l'exécution de méthodes distantes (RMI : Remote Method Invocation).
- ✓ la conception d'applications réparties selon le modèle d'espaces (issus du langage Linda) avec JavaSpaces,
- ✓ la gestion de serveurs Web via les Servlets,
- ✓ la communication d'objets distants inter-langages avec des IDL (Interface Definition Language) CORBA (Common Request Broker Architecture),
- ✓ l'administration de réseaux via SNMP (Simple Network Management Protocol) avec JMAPI.

Java est multitâches

- ✓ L'un des premiers langages à posséder en interne des tâches, ou activités (threads) d'exécution.

- ✓ La coordination des activités est aisée (moniteurs de Hoare et événements).

Java est dynamique

- ✓ Exécution coté client : dynamisme plus aisé à mettre en œuvre que dans d'autres langages.
- ✓ Chargement des classes en cours d'exécution, lorsque nécessaire, éventuellement à travers le réseau. Chargement dynamique des classes possible grâce à des informations de typage consultables en cours d'exécution.

Java est sûr

- ✓ Écriture mémoire erronée : quasi-impossible en java, car pas de pointeurs.
- ✓ Indices de tableau testés avant qu'ils soient référencés.
- ✓ Test qu'une variable a été assignée avant d'être utilisée.
- ✓ Bytecode également testé avant d'être exécuté :
  – test de bon accès aux classes,
  – tests de congestion et de famine de la pile des opérandes,
  – test de conversion illégale de données,
  – test d'accès aux ressources : fichiers,
  – . . .

Java est indépendant de l'architecture

- ✓ Le bytecode est indépendant de la plate-forme.
- ✓ Les bibliothèques sont intégrées de manière standard au langage, à l'encontre de C++.

Les technologies réseaux de Java

- ✓ Flux de données réseau TCP et UDP par sockets (Socket, . . . ; JDK).
- ✓ Appel de méthodes distantes (RMI ou Remote Method Invocation ; JDK).
- ✓ Interopérabilité réseau inter-langage via CORBA (IDL ou Interface Definition Langage ; JDK).
- ✓ Appel de méthodes distantes au dessus du protocole Internet d'interopérabilité réseau inter-langage (RMI-IIOP ou Remote Method Invocation over Internet Inter-Orb Protocol ; paquetage optionnel).
- ✓ Fonctions de serveurs HTTP (Java Servlets ; paquetage optionnel).
- ✓ Communication distribuée par espaces (JavaSpaces).
- ✓ Applicationsmutli-agent réseau (JDMK, Java Dynamic Management Kit).
- ✓ Administration distribuée (Java Management ; paquetage en accès d'avant première).
- ✓ Gestion de courier (Java Mail ; paquetage optionnel).
- ✓ Service de nommage et de répertoires (JNDI ou Java Naming Directory Interface ; paquetage optionnel).

Adaptation au web

- ✓ le caractère "embarqué" initial de java est-il bien adapté au Web (transfert de pages HTML et exécution de programmes distante via Internet)
- ✓ Calculs coté client via des applets : plutôt que d'exécuter le programme et de transmettre la réponse, le serveur transmet le programme. Le programme s'exécute localement sur le client (machine moins chargée).

Technologies de Sécurité

- ✓ Liste de contrôle d'accès ou "ACLs" (JDK)
- ✓ Authentification et autorisation (JAAS ou Java Authentication and Authorization Service ; paquetage en accès d'avant première)
- ✓ Flux réseau sécurisé par des SSL ou Secure Socket Layer (JSSE ou Java Secure Socket Extension ; paquetage en accès d'avant première)
- ✓ Cryptographie (JCE ou Java Cryptography Extension ; extension standard)

Technologies graphiques

- ✓ Gestion d'interfaces graphiques (AWT ou AbstractWindow Toolkit et Swing, formant les JFC ou Java Foundation Classes ; JDK).
- ✓ Composants réutilisables, éditables au sein d'un concepteur d'interfaces graphiques ou "GUI builder" (Java Beans ; JDK).
- ✓ Dessin vectoriel 2D (Java2D ; JDK).
- ✓ Traitement d'images de base (Java advanced imaging ; paquetage optionnel).
- ✓ Synthèse d'images et VRML (Java3D; paquetage optionnel)
- ✓ Gestion multimédia (JMF, Java Media Framework ; extension standard).
- ✓ Synthèse vocale (Java Sound) [H. Mounier].

Pour approfondir un peu, voici un aperçu technique de la mise en correspondance entre les éléments graphiques (html) et les objets Java qui supportent à la fois la présentation de l'information, le contrôle et la modification du modèle.

II.5.5 Le développement des interfaces graphiques

Les interfaces graphiques assurent le dialogue entre les utilisateurs et une application. Dans un premier temps, Java proposait l'API AWT pour créer des interfaces graphiques. Depuis, Java propose une nouvelle API nommée Swing. Ces deux API peuvent être utilisées pour développer

des applications ou des applets. Face aux problèmes de performance de Swing, IBM a créé sa propre bibliothèque nommée SWT utilisée pour développer l'outil Eclipse. La vélocité de cette application favorise une utilisation grandissante de cette bibliothèque.

La classe Graphics contient les outils nécessaires pour dessiner. Cette classe est abstraite et elle ne possède pas de constructeur public : il n'est pas possible de construire des instances de graphics nous même. Les instances nécessaires sont fournies par le système d'exploitation qui instanciera grâce à la machine virtuelle une sous-classe de Graphics dépendante de la plate-forme utilisée [Jean.M 2002].

II.5.5.1 Les éléments d'interfaces graphiques de l'AWT

Les classes du toolkit AWT (Abstract Windows Toolkit) permettent d'écrire des interfaces graphiques indépendantes du système d'exploitation sur lesquels elles vont fonctionner. Cette librairie utilise le système graphique de la plate-forme d'exécution (Windows, MacOS, X-Window) pour afficher les objets graphiques. Le toolkit contient des classes décrivant les composants graphiques, les polices, les couleurs et les images.

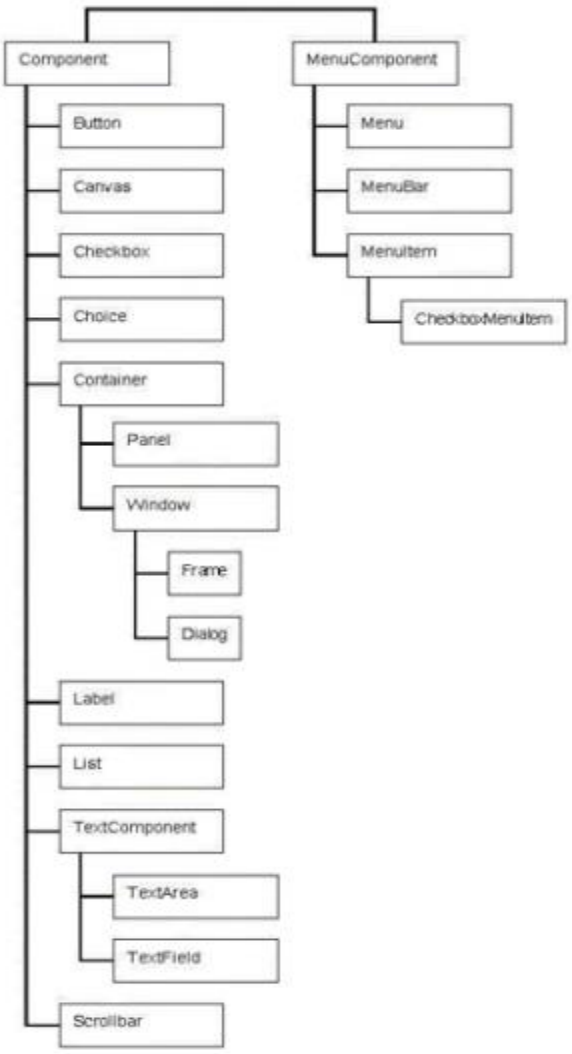

Figure II.13 Les éléments d'interfaces graphiques de l'AWT

Le diagramme ci-dessus définit une vue partielle de la hiérarchie des classes (les relations d'héritage) qu'il ne faut pas confondre avec la hiérarchie interne à chaque application qui définit l'imbrication des différents composants graphiques. Développons en Java les deux classes principales d'AWT sont Component et Container. Chaque type d'objet de l'interface graphique est une classe dérivée de Component. La classe Container, qui hérite de Component est capable de contenir d'autres objets graphiques (tout objet dérivant de Component) [Jean.M 2002].

II.5.5.2 L'interception des actions de l'utilisateur avec Java

Les événements utilisateurs

sont gérés par plusieurs interfaces EventListener. Les interfaces EventListener permettent de définir les traitements en réponse à des événements utilisateurs généré par un composant. Une classe doit contenir une interface auditrice pour chaque type d'événements à traiter [Jean.M 2002] :

- ✓ ActionListener : clic de souris ou enfoncement de la touche Enter
- ✓ ItemListener : utilisation d'une liste ou d'une case à cocher
- ✓ MouseMotionListener : événement de souris
- ✓ WindowListener : événement de fenêtre

Le mécanisme mis en place pour intercepter des événements est le même quel que soit ces événements

Associer au composant qui est à l'origine de l'événement un contrôleur adéquat : utilisation des méthodes addXXXListener() [Jean.M 2002] :

- ✓ Le paramètre de ces méthodes indique l'objet qui a la charge de répondre au message : cet objet doit implémenter l'interface XXXListener correspondante ou dérivée d'une classe XXXAdapter, ce qui revient à créer une classe qui implémente l'interface associée à l'événement que l'on veut gérer.
- ✓ Cette classe peut être celle du composant qui est à l'origine de l'événement (facilité d'implémentation) ou une classe indépendante qui détermine la frontière entre l'interface graphique (émission d'événements) et celle qui représente la logique de l'application (traitement des événements) .
- ✓ les classes XXXAdapter sont utiles pour créer des classes dédiées au traitement des événements car elles implémentent des méthodes par défaut pour celles définies dans l'interface XXXListener dérivées de EventListener. Il n'existe une classe Adapter que pour les interfaces qui possèdent plusieurs méthodes.
- ✓ implémenter la méthode associée à l'événement qui fournit en paramètre un objet de type AWTEvent (classe mère de tout événement) contenant des informations utiles (position du curseur, état du clavier ...).

L'apparition d'un **événement utilisateur** généré par un composant doté d'un auditeur appelle automatiquement une méthode. Cette dernière doit se trouver dans la classe référencée dans l'instruction qui lie l'auditeur au composant.

II.5.6 Principes de la programmation par événement

II.5.6.1 Modèle de délégation de l'événement en Java

Logique selon laquelle un programme est construit avec des objets et leurs propriétés et d'après laquelle seules les interventions de l'utilisateur sur les objets du programme déclenchent l'exécution des routines associées. L'approche événementielle intervient principalement dans l'interface entre le logiciel et l'utilisateur, mais aussi dans la liaison dynamique du logiciel avec le système, et enfin dans la sécurité.

Il est possible de relier certains objets entre eux par des relations événementielles. Nous les représenterons par un graphe (structure classique utilisée pour représenter des relations).

Avec des systèmes multi-tâches préemptifs sur micro-ordinateur, le système d'exploitation passe l'essentiel de son " temps " à attendre une action de l'utilisateur (événement). Cette action déclenche un message que le système traite et envoie éventuellement à une application donnée [RmdiScala 2006] .

Nous pourrons construire un logiciel qui réagira sur les interventions de l'utilisateur si nous arrivons récupérer dans notre application les messages que le système envoie. Java autorise aussi la consultation de tels messages.

En Java, le traitement et le transport des messages associés aux événements sont assurés par deux objets dans le cadre d'un modèle de communication dénommé le modèle de traitement des événements par délégation (Delegation Event Model)

- ✓ Le message est envoyé par une source ou **déclencheur** de l'événement qui sera un composant Java, à un récepteur ou **écouteur** de l'événement qui est chargé de gérer l'événement, ce sera un objet de la classe des écouteurs instancié et ajouté au composant.

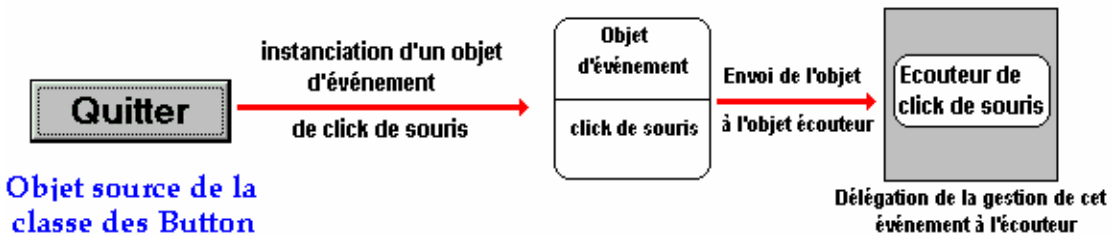

Figure II.14 Modèle de délégation de l'événement en Java

La méthode de programmation de l'interception des événements est nettement plus lourde en Java qu'en Delphi et en Visual Basic. Ce sont des classes abstraites dont le nom généralement se termine par **Listener**. Chacune de ces classes étend la classe abstraite d'interface **EventListener**.

Toutes ces classes d'écouteurs d'événements sont situées dans le package**java.awt.event**, elles se chargent de fournir les méthodes adéquates aux traitements d'événements envoyés par un déclencheur.

Les événements possibles dans Java sont des objets (un événement est un message contenant plusieurs informations sur les états des touches de clavier, des paramètres,...) dont les classes sont dans le package **java.awt.event.**

La gestion des événements est d'une importance capitale pour les programmes ayant une interface utilisateur graphique. En effet, en mode graphique, ce n'est plus une programmation déterministe, qui impose donc un fonctionnement séquentiel, mais plutôt une programmation qui propose un ensemble d'actions spécifiques, au moment où l'utilisateur le désire et en rapport avec des événements particuliers, comme par exemple le clic d'un bouton [RmdiScala 2006].

II.5.6.2 Notion d'état d'un objet

L'état d'un objet englobe toutes les propriétés (habituellement statiques) de l'objet plus les valeurs courantes (généralement dynamiques) de chacune de ces propriétés.

Une propriété est une caractéristique naturelle ou discriminante, un trait, une qualité ou une particularité qui contribue à rendre un objet unique.

Un objet Java peut être décrit par la formule suivante :

OBJET = (état,op1, . . . , opn, ref)

où *etat* ensemble des variables d'instance

opi (pointeur sur) la méthode d'instance n*i*

ref (pointeur sur) un emplacement mémoire contenant l'état et des références internes vers les opérations (pointeurs sur les méthodes)

II.5.6.3 Notion de comportement d'un objet

Le comportement est la façon dont un objet agit et réagit, en termes de changement d'état et de transmission de messages. Généralement, un message est simplement une opération qu'un objet effectue sur un autre, bien que le mécanisme utilisé soit quelque peu différent.les termes opération et message sont interchangeables. La transmission de messages est une partie de l'équation qui définit le comportement d'un objet. L'état d'un objet influence aussi son comportement.

Dans la plupart des langages orientés objets et basés sur objets, les opérations que les clients peuvent effectuer sur un objet sont typiquement appelées des méthodes, qui font partie de la classe de l'objet. Nous pouvons donc affiner la notion d'état [H. Mounier]:

- ✓ L'état d'un objet représente les effets cumulés de son comportement.
- ✓ La majorité des objets intéressants n'ont pas d'état entièrement statique. Ils contiennent des propriétés dont les valeurs sont lues et modifiées en fonction des actions sur ceux-ci.

1.5.6.4 Les opérations

Une opération désigne un service qu'une classe offre à ses clients. En pratique, un client effectuait typiquement 5 sortes d'opérations sur un objet.

Les 3 les plus courantes sont les suivantes :

- ✓ ***Modificateur*** une opération qui altère l'état d'un objet
- ✓ ***Sélecteur*** une opération qui accède à l'état d'un objet, mais qui n'altère pas celui-ci.
- ✓ ***Itérateur*** une opération qui permet d'accéder à toutes les parties d'un objet dans un ordre bien défini.

Deux autres types d'opération sont courants :

- ✓ **Constructeur** une opération qui crée un objet et/ou initialise son état.
- ✓ ***Destructeur*** Une opération qui libère l'état d'un objet et/ou détruit l'objet lui-même.

Avec des langages orientés objets, comme Java ou Smalltalk, les opérations peuvent seulement être déclarées comme méthodes (c.à.d. au sein d'une classe), le langage ne nous autorisant pas à déclarer des procédures ou des fonctions séparées de toute classe. Ce n'est pas le cas en C++ et en AdA, qui autorisent le programmeur à écrire des opérations en dehors de toute classe.

II.5.6.5 Rôle et responsabilités

L'ensemble des méthodes associées à un objet constituent son protocole. Ce dernier définit la totalité des comportements autorisés pour l'objet. Il est utile de diviser ce protocole en groupes logiques de comportements. Ces groupes désignent les rôles que l'objet peut jouer ; un rôle s'apparente à un masque porté par l'objet et définit un contrat entre une abstraction et un client.

Les responsabilités d'un objet sont constituées d'une part de la connaissance que l'objet maintient et d'autre part des actions qu'il peut réaliser. En d'autres termes, L'état et le comportement d'un objet définissent l'ensemble des rôles qu'il peut jouer, lesquels définissent les

responsabilités de l'abstraction. La majorité des objets jouent plusieurs rôles au cours de leur existence, par exemple :

- ✓ Un compte bancaire peut être créditeur ou débiteur, ce qui influe sur le comportement d'un retrait d'argent.
- ✓ Durant une même journée, un individu peut jouer le rôle de mère, de médecin, de jardinier.

II.5.6.6 Les objets en tant que machines

L'existence d'un état dans un objet signifie que l'ordre dans lequel les opérations sont invoquées est important. Chaque objet peut donc être vu comme une petite machine ou un automate à état finis équivalent.

Les objets peuvent être actifs ou passifs. Un objet actif contient sa propre tâche de contrôle, contrairement à un objet passif. Les objets actifs sont généralement autonomes, ce qui signifie qu'ils peuvent présenter un certain comportement sans qu'un autre objet agisse sur eux. Les objets passifs ne peuvent subir un changement d'état que lorsque l'on agit explicitement sur eux [H. Mounier].

II.5.7 Programme Java-Swing

II.5.7.1 Structure

- ✓ Un programme Java-Swing est composé d'un ensemble de **widgets**. A ces widgets peuvent être associes des **listeners** d'évènements.
- ✓ Un listener d'évènements implémente des méthodes. Suivant le type d'évènement produit par l'action de l'utilisateur sur un widget, la "bonne" méthode de listener est exécutée.
- ✓ **Méthode** de listener : le corps d'une méthode de listener modifie le rendu de l'interface et effectue des calculs par appel au noyau fonctionnel de l'application.

II.5.7.2 Contrôle de flux

A) Instruction if-else

Forme strictement analogue à celle du C

if (expression-booleenne) expression1;

else expression2;

expressioni peut être une expression composée entourée de {}.

expression-booleenne est toute expression renvoyant un boolean.

B) Instruction break

Utilisation courante strictement analogue à celle du C : pour sortir d'un case à l'intérieur d'un switch.

Autre utilisation : sortie d'un bloc marqué. Marquage par étiquette : un identificateur suivi de ' :' placé devant le bloc.

break suivi du nom de marquage du bloc permet une sortie du bloc marqué par cette étiquette.

On ne peut se brancher à une étiquette qui n'est pas définie devant un des blocs englobant, sinon break serait identique à goto.

C) Instruction switch

Forme strictement analogue à celle du C

```
switch ( expression ) {
case valeur1 :
break;
case valeurN :
break;
default :
}
```

expression peut être tout type primitif (les *valeuri* doivent être du même type qu'*expression*)

C'est une erreur répandue que d'oublier un break.

D) Instructions while/do-while/for

Mêmes usages et syntaxe qu'en C.

```
[ initialisation; ]
while ( terminaison ) {
corps;
[ iteration; ]
}
```

do-while

```
[ initialisation; ]
do {
corps;
[ iteration; ]
} while ( terminaison );
```

for

```
for (initialisation; terminaison; iteration)
corps;
```

II.6 Conclusion

Dans ce chapitre nous avons discuté les principaux concepts et points clés de la technologie de développement des applications Web 2+, les différents travaux et axes de recherche énoncés dans ce chapitre reposent sur deux domaines, présentés par la figure suivante :

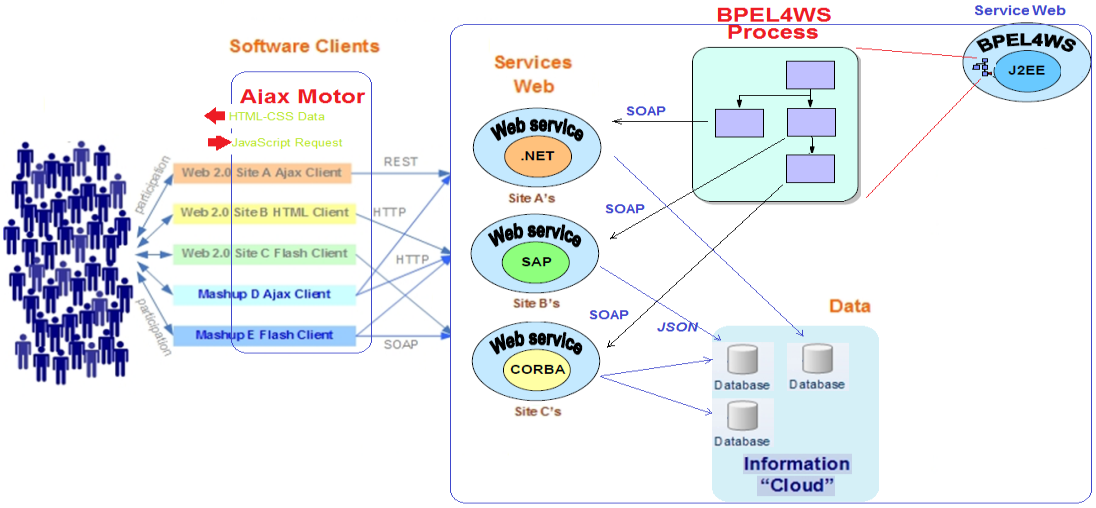

Figure II.13 Architecture des applications web 2+

Nous avons commencé par la définition de notion de Web 2 (social ou participatif) ; grâce à lui, les internautes deviennent des véritables co-développeurs des applications ; par ailleurs, le web 2.0 en permettant le passage de la notion de logiciel produit à celle de logiciel service ; pour cela ensuite nous avons montré l'importance des architectures orientées service Web qui sont considéré la déclinaison du paradigme des architectures orientées service, sur le Web. Dans cette partie nous avons survolé sur les différentes couches de l'architecture SOA puis, sur en plus des standards que sont SOAP, WSDL et UDDI, différents langages et spécifications nous avons également montré l'impact et l'importance et de La composition de services Web, elle est au cœur des architectures SOA puisque elle supporte la construction de services, dits composés ou composites, à partir de fonctionnalités de base . Elle a pour but la réutilisation des services (simples ou composés). On peut distinguer deux méthodes de composition de services Web :

- ✓ l'orchestration (XLANG, WSFL, BPML et BPEL4SW)
- ✓ la chorégraphie (WS-CDL, WSCI)

Ainsi nous avons présenté les principales techniques des applications web riches (RIA), ces technologies qui sont à la base du Web 2.0 ne sont pas récentes. C'est leur convergence qui est nouvelle, répondant au besoin accru d'interactivité des internautes. Ajax est la combinaison de ces

technologies telles que Javascript, CSS, XML, le DOM et le XMLHttpRequest dans le but de réaliser des applications Web qui offrent une maniabilité et un confort d'utilisation .Enfin de ce chapitre nous avons donné des aperçus généraux sous divers angles pour le langage JAVA pour mettre les techniques (AJAX) en œuvre ensemble ; où nous avons abordé les aspects les plus importants se rapportant à l'objet de notre travail, qui sont:

- ✓ Les frameworks Java WEB
- ✓ Les caractéristiques de java
- ✓ Le développement des interfaces graphiques
- ✓ L'interception des actions de l'utilisateur avec Java
- ✓ Principes de la programmation par événement
- ✓ Modèle de délégation de l'événement en Java
- ✓ Notion d'état d'un objet
- ✓ Notion de comportement d'un objet
- ✓ Les opérations
- ✓ Rôle et responsabilités
- ✓ Programme Java-Swing (Structure, flux de contrôle)

Cette combinaison de technologies multiples et complexes nécessite de pouvoir garantir une certaine qualité de service (par exemple le respect de toutes les contraintes initiales et la prise en compte de tous les besoins) afin d'assurer qu'une application Web2+ satisfait les besoins fixés par les spécifications de l'architecture.

Nombreuses travaux et recherches ont été proposés à cet effet afin d'assurer la concordance et la conformité entre la spécification et l'architecture réellement exécutée. Nous allons donc détailler plus précisément ces travaux dans le prochain chapitre.

Chapitre 03

- les travaux liés -

III
Les travaux Liés

III.1 Introduction

Le développement des applications Web 2+ et l'utilisation d'architectures basées sur les Services Web, présente des problèmes de la cohérence des services. En effet, comment peut-on être sûr qu'une application, basée sur des Services Web distants, et interconnectés par des réseaux dont on ne connaît presque rien, aboutira par la bonne composition de Services Web, à une application correcte ? Pour cela, il est nécessaire d'avoir une sémantique opérationnelle précise des langages de description comportementale de ces applications. Cette sémantique opérationnelle permet alors d'appliquer des méthodes formelles i.e. d'un formalisme mathématique défini précisément, syntaxiquement et sémantiquement. Selon la méthode considérée, l'utilisation de ce formalisme peut se limiter à la rédaction de spécifications non ambigües, ou être étendu à des activités de preuves voire à la programmation elle-même ; pour vérifier certaines propriétés comportementales du système étudié.

L'intérêt pratique d'une méthode formelle, cependant, est fortement dépendant de l'existence d'outils implémentant son formalisme, et éventuellement permettant d'automatiser certaines activités, telles que la production de preuves.

Afin de pallier le manque de formalisme dans le domaine de développement des applications Web 2+, Plusieurs travaux sont consacrés à la formalisation des orchestrations de services Web et ainsi permettre certaines vérifications de leur comportement. Cette étape de vérification va permettre d'assurer un certain niveau de confiance quant au comportement interne d'une orchestration. Dans ce chapitre nous allons présenter et discuter ces travaux :

III.2 Les systèmes de transitions LTSA-WS

Le système de transitions LTSA-WS (Labelled Transition System Analyser) [Foster et al., 2003] [Foster et al., 2005] [Foster et al., 2006] est une approche de vérification de systèmes. Il vérifie que la spécification d'un système concurrent satisfait les propriétés requises de son comportement ; à travers la comparaison de deux modèles, le modèle de spécification (conception) et le modèle d'implémentation.

Un système de transition d'états, ou automate au sens large, est un modèle de machine abstraite, utilisé en informatique théorique pour simuler le déroulement d'un calcul. Il consiste en la donnée d'un ensemble d'états, et d'un ensemble de transitions d'un état à un autre. Autrement dit, la définition formelle d'un système de transition d'états est un couple :

$(S, \rightarrow)$ avec $\rightarrow \subset S \times S$, où S est l'ensemble des états, et $\longrightarrow$ est la relation de transition.

Si p et q sont deux états, $(p, q) \in \rightarrow$ veut dire qu'il existe une transition de p à q, et se note aussi $p \rightarrow q$.

On ne fait aucune hypothèse a priori sur S et $\longrightarrow$, et ils peuvent être infinis, voire indénombrables. Cependant, si S est fini (et donc $\longrightarrow$aussi), le système de transition est un graphe orienté.

On peut aussi donner une définition étiquetée de système de transition : à ce moment, il faut de plus se donner un ensemble d'étiquettes Λ, et prendre $\rightarrow \subset S \times \Lambda \times S$. Le système de transition est alors le triplet $(S, \Lambda, \rightarrow)$. S'il existe une transition étiquetée par $\lambda \in \Lambda$entre deux états p et q, on note alors $p \xrightarrow{\lambda} q$.

Dans le cas où S et Λ sont finis, on parlera d'automates d'états finis (en général, on se donnera aussi une condition d'acceptation de mot d'entrée, qui sera souvent la donnée de deux parties de S qui seront les états initiaux, et les états accepteurs).

Le système de transitions est dit déterministe si et seulement si $\longrightarrow$ est une « fonction », et non-déterministe sinon. Dans le cas étiqueté, cette définition ne précise pas si on veut une fonction de S dans $\Lambda \times S$, ou de $S \times \Lambda$ dans S (ce qu'on veut dans le cadre des automates d'états finis), ou encore de $S \times \Lambda_1$ dans $\Lambda_2 \times S$ si $\Lambda = \Lambda_1 \times \Lambda_2$ (cas des transducteurs).

Les systèmes de transitions jouent un rôle important dans la reconnaissance des langages formels, notamment dans leur classification.

Exemples courants

- ✓ Machine de Turing ;
- ✓ Automate d'états finis ;
- ✓ Réseau de Petri ;
- ✓ Graphe orienté ;

Un système dans LTSA est modélisé comme un ensemble d'interactions machines à états finis. Les propriétés requises du système sont également modélisées comme des machines d'état. LTSA effectue l'analyse d'accessibilité de la composition d'une recherche exhaustive sur les violations des propriétés souhaitées. Plus formellement, chaque composante d'un cahier des charges est décrit comme un système de transition étiquetées (LTS), qui contient tous les états qu'un composant peut atteindre et toutes les transitions.

Pendant la phase de conception, l'approche utilise UML par le biais de diagrammes de séquence, afin de décrire comment sont utilisés les différents services Web d'une orchestration et comment ces derniers interagissent. L'ensemble des scénarios obtenus sont ensuite composés et synthétisés pour générer un modèle comportemental en FSP (Finite State Process), lui même compilé en un LTS. Pendant la phase d'implémentation, l'approche utilise BPEL4WS pour concevoir l'orchestration des services Web utilisés. Un certain nombre de mécanismes permettent ensuite la traduction du processus BPEL4WS ainsi créé vers une description FSP, elle aussi compilée en un LTS.

La vérification et la validation de la cohérence des deux modèles se fait alors par comparaison et observation des traces générées par ces deux systèmes de transition. L'approche permet alors de déterminer si l'implémentation contient tous les scénarios spécifiés lors de la phase de conception et si des scénarios additionnels (souhaités ou non) peuvent exister.

D'autre part, le LTS permet aussi la vérification de propriétés de sûreté et de vivacités très générales (pas de chemins bloquants, tout état est atteignable).

Points forts

Premièrement, l'approche utilise des standards (diagrammes de séquence, BPEL4WS) accessibles qui masquent la complexité de FSP (Finite State Process). Deuxièmement, elle fournit un moyen visuel (via l'outil LTSA) de comparer le modèle de conception et le modèle d'implémentation tout en vérifiant l'absence de violation de propriétés générales.

Points faibles

Premièrement, l'approche désolidarise les phases de conception et d'implémentation pour les regrouper une fois ces dernières indépendamment terminées. En cas de non cohérence (concordance) de traces, l'implémentation est donc totalement à reprendre : la phase de vérification est alors très/trop tardive. Deuxièmement, UML étant semi-formel et BPEL4WS n'ayant aucune sémantique formelle, il est alors impossible de prouver que le FSP généré correspond exactement à ce qui est décrit. En ce sens, les comparaisons et les vérifications ne peuvent être que partielles. Troisièmement, la spécification BPEL4WS peut être interprétée par différents moteurs d'exécution, rien ne permettant de prouver que ces derniers aient exactement la même sémantique d'interprétation. De fait, ce qui est implémenté correspond certes, (partiellement) aux besoins définis lors de la conception, mais pas forcément à la réalité de l'exécution.

III. 3 Les réseaux de Petri

Les réseaux de Petri fait aussi l'objet de travaux [Hamadi and Benatallah, 2003] visant à formaliser les orchestrations de services Web. Cette approche permet l'expression de certains opérateurs spécifiques à la gestion du flux de contrôle, comme par exemple la séquence, l'alternative, l'itération, le parallélisme, la discrimination ou encore la sélection. Le réseau de Petri décrivant une orchestration peut ensuite être analysé et même comparé avec d'autres réseaux de Petri.

Un réseau de Petri est un moyen de :

- ✓ Modélisation du comportement des systèmes dynamiques à événements discrets.
- ✓ Description des relations existantes entre des conditions et des évènements.

Un réseau de petri est composé de places, transitions et arcs :

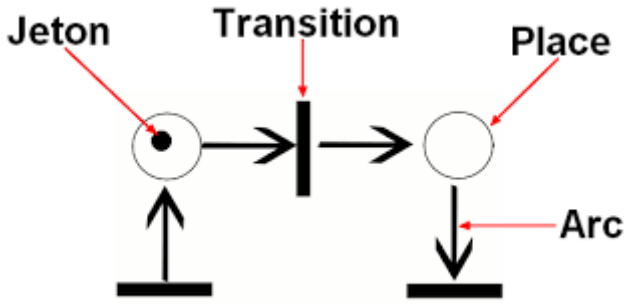

Figure III.1 Les composants des réseaux de Petri

- ✓ Une place est représentée par un cercle
- ✓ Une transition par un trait
- ✓ Un arc relie soit une place à une transition

Points forts

Cette approche formalise certains opérateurs utilisés pour l'orchestration de services Web sous la forme d'un réseau de Petri et permet ainsi certaines vérifications.

Points faibles

Premièrement, cette approche n'offre pas un moyen d'expression proche des standards du domaine des services Web. Deuxièmement, les structures comportementales formalisées sont très restreintes et ne permettent pas la description d'orchestrations complexes. Troisièmement, les réseaux de Petri offrent des mécanismes de simulation pour analyser des processus mais ne permettent aucune exécution de ces derniers. L'approche doit donc être complétée par une phase de traduction vers un langage exécutable, par exemple BPEL4WS. Ce dernier n'étant pas formel, aucune garantie ne peut donc être donnée quant à la correspondance exacte entre les opérateurs supportés par l'approche et leur traduction en BPEL4WS.

III.4 Les algèbres de processus

L'algèbre de processus est une catégorie de formalismes qui permettent de décrire et d'analyser les comportements de systèmes (processus) concurrents ou distribués. Différentes algèbres de processus ont été définies telles que LOTOS [LOTOS 1989] , π-calcul, le calcul des systèmes communicants (CCS). Ces algèbres de processus sont généralement représentées par un système de transitions étiquetées (LTS) où la relation de transition est définie par une collection de règles et d'axiomes. Les caractéristiques principales des algèbres de processus sont :

- ✓ Spécification et étude des systèmes concurrents (communication, synchronisation)
- ✓ Abstraction sur les comportements,
- ✓ Mode de synchronisation (synchrone, RdV, actions complémentaires), etc
- ✓ Mode de composition (parallèle, entrelacement, etc),
- ✓ Modèles sémantiques (opérationnelle, traces, bissimulation, failure-divergence).

Alphabets

L'évolution d'un processus est décrite à l'aide des noms des actions qu'il entreprend : *alphabet.*

Expressions régulières

Combinaison de noms d'action (*alphabet*) à l'aide d'opérateurs prédéfinis. Automate à états.

Processus élémentaire

L'évolution séquentielle, exprimée à l'aide de combinaison d'actions et d'opérateurs (séquence, choix, arrêt) : on parle de comportement (*behaviour*).

Une méthodologie outillée permettant la modélisation formelle et l'analyse des processus BPEL ont été proposée par Rampacek [Rampacek et al 2009] . Ils ont défini une formalisation de la sémantique de BPEL par une algèbre de processus temporisés appelée (Atp). Les comportements d'un processus BPEL sont décrits par des systèmes de transition à temps discret qui sont obtenus par une simulation exhaustive de cette algèbre à l'aide de l'outil WSMod. Ces modèles ont été ensuite analysés par la technique du Model-Checking en utilisant la boîte à outils CADP [CADP] , en particulier, le Model-Checker Evaluator.

III.5 ASDL (Abstract Service Design Language)

ASDL (Abstract Service Design Language) [Solanki et al., 2006] est un langage basé sur ITL (Interval Temporal Logic) [Cau et al., 2006], une algèbre de processus permettant de raisonner sur des contraintes temporelles. ASDL vise à permettre la description du comportement d'un service, d'une orchestration mais aussi la description des protocoles d'interactions entre les services. ASDL est donc un langage formel de haut niveau permettant le raisonnement et qui nécessite une traduction vers un langage d'orchestration de plus bas niveau afin d'être exécuté.

Points forts

Premièrement, ASDL a une sémantique clairement définie puisqu'il est fondé sur une algèbre de processus. Il permet ainsi le raisonnement sur un processus.

Deuxièmement, son haut degré d'abstraction lui permet de décrire une orchestration à différents niveaux (services, orchestration, protocoles).

Points faibles

Premièrement, ASDL présente une syntaxe complexe, loin des standards du domaine des services Web. Deuxièmement, le fait de s'interfacer à un haut niveau d'abstraction nécessite un autre langage (de plus bas niveau) pour exécuter une orchestration décrite avec ASDL, par exemple BPEL4WS. Cette traduction, exempte de tout formalisme, ne permet pas d'assurer que ce qui sera exécuté correspond exactement à ce sur quoi ASDL à permis de raisonner auparavant. Plus encore, il n'est pas certain que tout ce qui a été formalisé trouve une structure d'implémentation (passage d'un haut vers un plus bas niveau d'abstraction).

III.6 LOTOS/CADP

Est une approche [Salaun et al., 2004] [Chirichiello and Salaun, 2005] visant à lier, de manière bidirectionnelle, une orchestration décrite en BPEL4WS et sa formalisation en LOTOS [ISO/IEC, 1989]. Des équivalences ont été décrites afin qu'un comportement décrit dans un langage puisse être traduit dans l'autre et vice-versa. D'autre part, grâce à l'outil CADP, l'approche permet de raisonner sur la description formelle, exprimée en LOTOS, et ainsi vérifier l'orchestration réelle, décrite en BPEL4WS.

Un exemple : pour l'étude : producteur-consommateur de ressources :

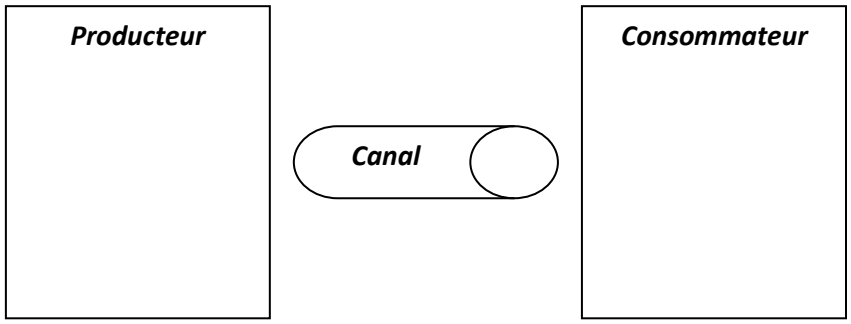

Figure III.2 LOTOS (Exemple : producteur-consommateur)

- ✓ Quel est le comportement du (processus) producteur ?
- ✓ Quel est le comportement du consommateur ?
- ✓ Qu'est-ce que le canal ? quel modèle ? quel comportement ?
- ✓ Quelles stratégies appliquées pour la gestion des ressources ?
- ✓ Produire tout, puis consommer tout ?
- ✓ produire et consommer alternativement ?

Hypothèses de travail :

- ✓ Le canal représente une mémoire tampon (buffer) de 2 places
- ✓ On modélise le canal comme un processus
- ✓ Le producteur est un processus qui doit produire 2 ressources puis s'arrêter
- ✓ Le consommateur est un processus qui doit consommer 2 ressources (ce qui est produit) et s'arrêter.

Soient *pc1*, *pc2* les deux actions du processus producteur : *pc1* représente la production de la ressource 1 vers l'environnement (ici le canal). Une action identifiant une porte, on peut percevoir le système producteur-canal comme suit :

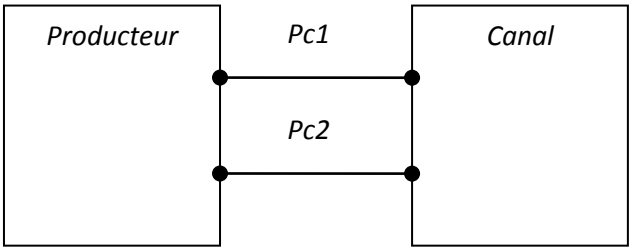

Figure III.3 LOTOS (Exemple-suite : producteur-consommateur)

pc1, *pc2* sont des portes.

Le comportement du processus producteur s'écrit alors :

process

Producteur [pc1, pc2] : exit :=

pc1 ; pc2 ; exit

endproc

pc1, *pc2* sont ici des (portes) paramètres formels (instanciables).

exit indique que le processus peut exécuter un exit à la fin.

On a utilisé le préfixage (noté ;).Un processus peut se terminer normalement avec succès (exit) ou alors sans succès (noexit).Une terminaison avec succès permet au contexte d'exécution de se poursuivre éventuellement avec un autre processus. Le processus consommateur est spécifié de la même façon par :

process

Consommateur [cc1, cc2] : exit :=

cc1 ; cc2 ; exit

endproc

cc1, cc2 sont ici les actions de consommation des ressources. *cc1* représente l'action de consommer la première ressource dans le canal.

La spécification du canal est comme suit :

process

Canal [pc1, pc2, cc1, cc2] : exit :=

pc1; pc2; cc1 ; cc2 ; exit

endproc

Puis il suffit de composer ces trois processus en parallèle.

Points forts

Le point fort de cette approche est le pouvoir d'expression des propriétés grâce l'outil CADP. En effet, cet outil permet l'expression de propriétés de sûreté et de vivacité spécifiques à un processus donné.

Points faibles

Premièrement, malgré son pouvoir d'expression, CADP reste un outil complexe à maîtriser. En effet, il n'est pas trivial de définir une propriété et encore moins de l'exprimer, d'autant plus que la syntaxe du langage ne fait pas référence aux concepts de l'orchestration de services Web. Deuxièmement, comme nous l'avons déjà mentionné dans les approches précédentes, ce type de formalisation ne permet pas d'assurer une cohérence entre ce qui est vérifié et ce qui est exécuté. L'approche proposée ici impose, elle aussi, une traduction d'une algèbre de processus vers un langage sans sémantique formelle afin qu'une orchestration soit exécutée. Cette étape récurrente introduit une perte de sémantique, amplifiée par les ambiguïtés qui peuvent être ajoutées par les différentes implémentations des moteurs BPEL4WS.

III.7 Le π –calcul

Au même titre que LOTOS [ISO/IEC,1989], le π-calcul [Milner, 1989] [Milner, 1999] est une algèbre de processus. Ces algèbres constituent des méthodes formelles pour modéliser des interactions entre processus. Par exemple la figure suivante illustre un processus P qui envoie une valeur v au processus Q à travers un canal de transmission α. Ces algèbres permettent, en construisant un modèle mathématique reprenant certaines descriptions du modèle d'analyse de l'instanciation (celles relatives aux interactions), de garantir la cohérence du modèle et la conformité du programme.

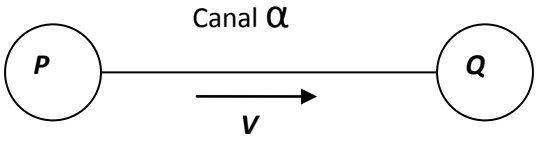

Figure III.4 Communication inter processus

Points forts

La particularité qui caractérise le π-calcul est l'introduction du concept de mobilité, c'est-à-dire la possibilité de modifier dynamiquement les liens entre différents processus ainsi que de déplacer un comportement d'un processus à un autre. Il existe différentes versions du π -calcul, chacune étant associée à des opérateurs spécifiques.

Points faibles

π-calcul présente une syntaxe complexe, loin des standards du domaine des services Web. Deuxièmement, le fait de s'interfacer à un haut niveau d'abstraction nécessite un autre langage (de plus bas niveau) pour exécuter une orchestration décrite avec π-calcul, par exemple BPEL4WS. Cette traduction, exempte de tout formalisme, ne permet pas d'assurer que ce qui sera exécuté correspond exactement à ce sur quoi π –calcul à permis de raisonner auparavant. Plus encore, il n'est pas certain que tout ce qui a été formalisé trouve une structure d'implémentation (passage d'un haut vers un plus bas niveau d'abstraction).

III. 8 Les théories temporelles

Les théories temporelles sont apparues suite à l'application de la logique dans le domaine de l'Intelligence Artificielle. Ces formalismes ont été introduits pour représenter des connaissances temporelles, pour spécifier des domaines d'objets dynamiques ou encore afin de raisonner pour résoudre des problèmes. Les travaux de [Rouached et al., 2006a] [Rouached et al., 2006b] sont basés sur l'une de ces théories . Event Calculus [Kowalski and Sergot, 1986]. Ils permettent, dans un premier temps, la formalisation d'une orchestration BPEL4WS grâce à une étape de traduction, qui rend possible la représentation d'une orchestration sous la forme d'un ensemble de prédicats décrits en Event Calculus.

Ils permettent ensuite la vérification de propriétés, fonctionnelles et non fonctionnelles, avant et pendant l'exécution de l'orchestration, ces propriétés étant elles-mêmes exprimées comme un ensemble de prédicats décrits en Event Calculus. Les vérifications en cours d'exécution sont effectuées grâce à un système de remonté d'informations, à savoir l'outil log4j (”http : //logging.apache.org/log4j/docs/”) couplé à la machine d'exécution bpws4j (”http : //alphaworks.ibm.com/tech/bpws4j”), qui permet la détection de toutes dérives lors de l'exécution.

Points forts

Premièrement, l'approche permet la vérification de propriétés fonctionnelles et non fonctionnelles. Deuxièmement, elle permet la vérification d'une orchestration BPEL4WS à un niveau statique (avant l'exécution) et tout au long de l'exécution d'une orchestration.

Points faibles

Premièrement, la description des propriétés en Event Calculus demande une certaine expertise, éloignée des standards du domaine des services Web. Deuxièmement, la phase de traduction entre BPEL4WS et son langage de formalisation reste présente comme dans les autres approches, ce qui induit les mêmes restrictions, à savoir de potentielles pertes de sémantiques. Ces dernières sont en parties compensées par la phase de vérification tout au long de l'exécution. Cependant la détection de dérives lors de l'exécution intervient une fois que ces dernières sont réellement arrivées.

III. 9 Les logiques temporelles

Pour décrire les propriétés de bon fonctionnement des applications, les logiques temporelles sont des formalismes bien adaptés, notamment par leur capacité à exprimer l'ordonnancement des actions (événements) dans le temps. Les descriptions de propriétés en logique temporelle présentent deux qualités importantes [Manna and Pnueli, 1990] :

- ✓ Elles sont abstraites, c'est-à-dire indépendantes des détails d'implémentation de l'application,
- ✓ Elles sont modulaires, c'est-à-dire que le rajout, le changement ou la suppression d'une propriété ne remet pas en cause la validité des autres.

Il existe en général deux classes fondamentales de propriétés sur les exécutions :

- ✓ Les propriétés de sûreté (sous certaines conditions, quelque chose de "mal" ne va jamais arriver),
- ✓ Les propriétés de vivacité (sous certaines conditions, quelque chose de "bon" finira par arriver).

Lors du choix d'une logique temporelle, plusieurs aspects doivent être considérés, parmi lesquels :

- ✓ L'expressivité (la capacité de la logique à exprimer les classes de propriétés intéressantes, telles que la sûreté ou la vivacité),
- ✓ ℒa complexité d'évaluation (la complexité des algorithmes permettant de vérifier qu'un modèle satisfait une propriété),
- ✓ ℒa facilité d'utilisation (la capacité à exprimer les propriétés de manière concise et naturelle).

L'optimisation de l'un ou l'autre de ces aspects ne pouvant généralement se faire qu'au détriment des autres, le choix doit passer par un compromis judicieux (par exemple, si l'efficacité d'évaluation est l'aspect le plus important, alors l'expressivité de la logique devra être limitée). En outre, en raison de la diversité des logiques temporelles existantes et des résultats présents dans la littérature, il n'est pas toujours aisé de réunir les éléments pertinents pour choisir une logique temporelle adaptée à un certain contexte.

III. 10 Le langage FIACRE

FIACRE [Farail et Gaufillet 2008] est un langage formel dédié à la modélisation des aspects comportementaux et temporels des systèmes, utilisé pour la vérification formelle. C'est une algèbre de processus où les composants hiérarchiques communiquent soit par des messages synchrones au

travers de ports soit via des variables partagées. Les ports et les variables partagées définissent l'interface d'un composant. On distingue deux types de composants : les composants feuilles (Process) et les composants hiérarchiques (Component).

Process : décrit un comportement séquentiel en utilisant des systèmes de transition symboliques basés sur [Henzinger et Manna 1991]. Ainsi, un processus est défini par un ensemble d'états et de transitions. Les transitions sont associées à des gardes et peuvent comporter des actions non déterministes (affectation, synchronisation).

Component : décrit la composition en parallèle de sous-composants. En outre, un component peut introduire des variables et des ports qui seront partagés par ses sous-composants. A ce niveau, des contraintes de temps réel et des priorités peuvent être associées aux ports. Un port p est associé à une contrainte de temps sous la forme d'un intervalle de temps I. Cela signifie qu'une fois l'événement p activé, l'exécution doit attendre au moins la borne minimale de I et au plus sa borne maximale avant de désactiver p ou de franchir la transition synchronisée [Manna and Pnueli, 1990].

Transformation : BPEL vers FIACRE

La partie statique de BPEL (WSDL) est modélisée par des types globaux en FIACRE. La partie dynamique de BPEL est composée de l'activité principale du processus BPEL et des handlers. Elle est modélisée par un component FIACRE racine qui contient la composition des modèles FIACRE associés à l'activité primaire et ses handlers :

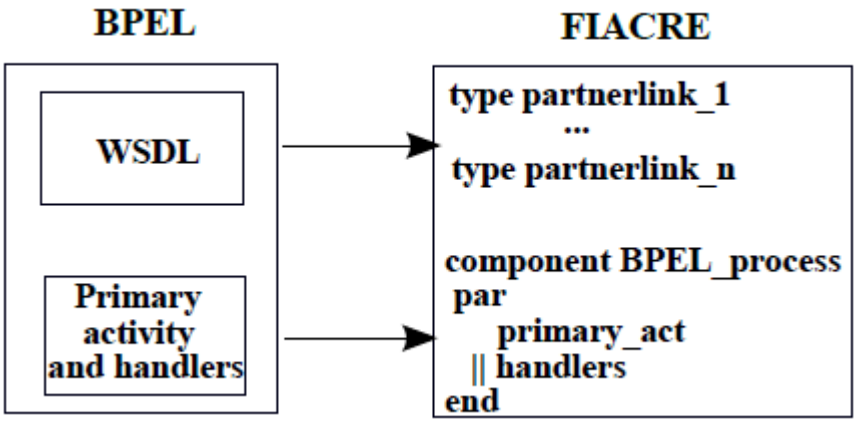

Figure III.5 Structure de la transformation (BPEL-FIACRE)

En outre, les activités basiques de BPEL sont transformées en des processus FIACRE tandis que les activités structurelles et les handlers – capables de contenir d'autres activités - sont transformées en des component FIACRE [Manna and Pnueli, 1990].

III. 11 Les logiques temporelles arborescentes basées sur actions

Les logiques arborescentes permettent de spécifier des propriétés sur les arbres d'exécution issus des états ou des actions d'un STE (Système de Transitions Etiquetées). Elles peuvent être vues comme des extensions des logiques modales avec des opérateurs temporels exprimant l'accessibilité potentielle ou inévitable de certains états et/ou actions. Les logiques temporelles arborescentes basées sur actions que nous allons considérer, sont interprétées sur des STEs, qui sont les modèles naturellement associés aux langages de type algèbre de processus (à la différence des structures de Kripke [Mateescu, 1998], basées sur des états). Formellement, un STE est un quadruplet :

$\mathbf{M = (S, A, T, s_0)}$

avec S, l'ensemble des états
et A, l'ensemble des actions (contenant l'action invisible τ)
et $T \subseteq S \times A \times S$, la relation de transition
et $s_0 \in S$, l'état initial

Tous les états de S sont supposés être accessibles à partir de l'état initial s0 par des séquences de transitions de T. Une transition peut être notée de deux manières :

$\mathbf{(s_1 , a, s_2) \in T \ (avec \ a \in A)}$

$ou \ s_1 \xrightarrow{a} s_2$

Ces deux notations signifient que le système peut passer de l'état s1 à l'état s2 en exécutant l'action a. Elles peuvent être étendues aux séquences de transitions comme suit :

$\mathbf{(s_1 , l, s_2) \in T \ (avec \ l \subseteq A)}$

$ou \ s_1 \xrightarrow{l} s_2$

Ces deux notations signifient qu'à partir de l'état s1 le système peut effectuer une séquence d'actions qui mène à s2.

III. 12 La logique ACTL

ACTL (Action Computation Tree Logic) [Nicola and Vaandrager, 1990] peut être considérée comme le représentant standard des logiques arborescentes pour les LTS (logiques basées sur actions). La logique ACTL contient trois types d'entités :

- ✓ Les formules sur actions (notées α),

- ✓ Les formules sur chemins (notées ψ),
- ✓ Les formules sur états (notées φ).

Ces formules permettent respectivement de caractériser des sous-ensembles d'actions, de chemins et d'états d'un LTS M = (S, A, T, s0). Un chemin est une suite d'actions et d'état, reliés par une relation de transition, formant une exécution possible d'un processus. Un LTS correspond donc en ensemble de chemins.

Syntaxe et sémantique des opérateurs

Les formules sur actions sont construites au moyen d'opérateurs booléens. La sémantique de ces opérateurs est habituelle (théorie des ensembles). La sémantique d'une formule α sur un STE *M = (S, A, T, s0)* est définie par l'interprétation [[α]] $\subseteq$ A, qui dénote le sous-ensemble d'actions du STE satisfaisant α :

α := **a**	[[a]]	=	a
| **false**	[[false]]	=	$\emptyset$
| **true**	[[true]]	=	A
| $\neg\alpha$	$[[\neg\alpha_1]]$	=	A $[[\alpha_1]]$
| $\alpha_1 \vee \alpha_2$	$[[\alpha_1 \vee \alpha_2]]$	=	$[[\alpha_1]] \cup [[\alpha_2]]$
| $\alpha_1 \wedge \alpha_2$	$[[\alpha_1 \wedge \alpha_2]]$	=	$[[\alpha_1]] \cap [[\alpha_2]]$

Les formules sur chemins sont construites au moyen de l'opérateur de succession next (noté X) et de l'opérateur temporel until (noté U). Etant donné un STE M = (S, A, T, s0), l'ensemble des séquences d'exécution maximales (c'est-à-dire les séquences se terminant par un état n'ayant aucun successeur) est noté Path. L'ensemble des séquences d'exécution maximales issues d'un état du STE peut être décrit sous la forme :

$$\mathbf{Path(s)} = s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ \ldots$$

avec $s \in S$

La sémantique d'une formule sur M est définie par l'interprétation [[α]] $\subseteq$ Path, qui dénote le sous-ensemble de séquences satisfaisant ψ :

ψ := $\mathbf{X}_\alpha\ \varphi$	$[[X_\alpha\ \varphi]] = s_1 \xrightarrow{a_1} s_2 \ \ldots \mid a_1 \in [[\alpha]] \wedge s_2 \in [[\varphi]]$
| $\varphi 1_{\alpha}\ \mathbf{U}\ \varphi_2$	$[[\varphi 1_\alpha\ U\ \varphi_2]] = s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{i-1}} s_i \ \ldots \mid i \geq 1 \wedge s_i \in [[\varphi_2]] \wedge \forall j \in [1,i-1].a_j \in [[\alpha]] \wedge s_j \in [[\varphi_1]]$
| $\varphi 1_{\alpha_1}\ \mathbf{U}_{\alpha_2}\ \varphi_2$	$[[\varphi 1_{\alpha_1}\ U_{\alpha_2}\ \varphi_2]] = s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{i-1}} s_i \ \ldots \mid i \geq 2 \wedge s_i \in [[\varphi_2]] \wedge a_{i-1} \in [[\alpha_2]] \wedge s_{i-1} \in [[\varphi_1]] \wedge \forall j \in [1,i-2].a_j \in [[\alpha_1]] \wedge s_j \in [[\varphi_1]]$

III. 13 L'approche Diapason

L'approche Diapason [Frédéric, 2007] propose un moyen de décrire des architectures orientées service Web de manière totalement formelle, garantissant ainsi une unité quant à l'interprétation et à l'exécution d'une orchestration. Cette formalisation étant conservée de la phase de description à la phase d'exécution, il est alors possible de garantir que ce qui est exécuté correspond exactement à ce qui est décrit, et qui plus est quelle que soit la machine virtuelle utilisée. Cette formalisation est réalisée grâce à un langage spécifique à l'orchestration de services Web : le langage π -Diapason, qui offre un pouvoir d'expression supérieur aux langages d'orchestration actuels (par exemple BPEL4WS) et qui est de plus extensible. Le langage π -Diapason est donc défini en trois couches distinctes à savoir :

- ✓ *Une couche noyau*, qui correspond au π -calcul (couche de base, permettant la formalisation et l'évolution dynamique),
- ✓ *Une couche patrons de Workflow*, qui permet d'enrichir la sémantique opérationnelle du π -calcul (sur-couche de la couche noyau, permettant la formalisation des concepts associés à la gestion de processus),
- ✓ *Une couche orientée service Web*, qui spécialise la couche précédente (sur-couche de la couche patrons de Workflow, permettant la formalisation des concepts associés à l'orchestration de services Web).

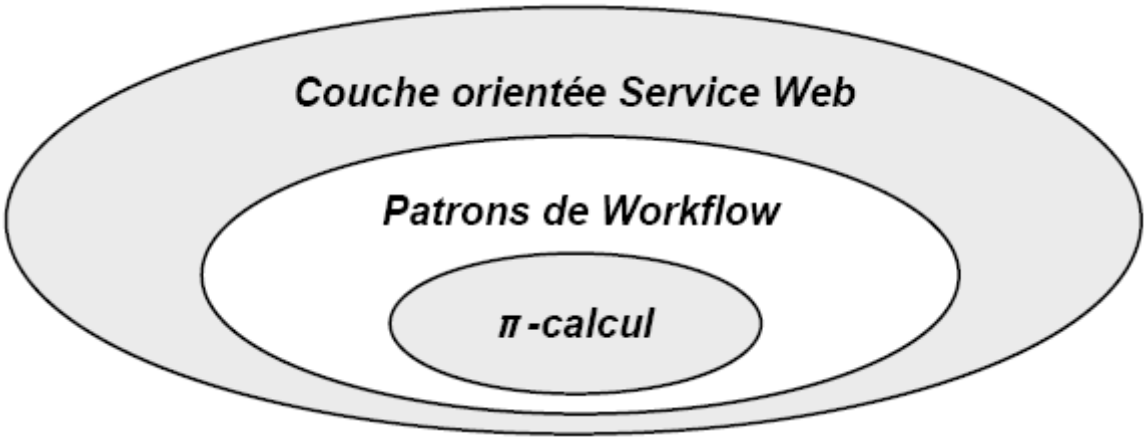

Figure III.6 Les couches du langage π -Diapason

En parallèle à la définition du langage π-Diapason et afin de permettre la vérification de propriétés sur une orchestration décrite avec ce dernier, Il y a aussi un langage permettant d'exprimer de manière intuitive des propriétés spécifiques à une orchestration de service Web. Ce langage est nommé Diapason* et étend les concepts des logiques temporelles ACTL [Nicola and Vaandrager, 1990] et ACTL* [Nicola and Vaandrager, 1990] .

Fort de ces deux langages, le processus de développement architectural Diapason peut être schématisé comme le montre la figure suivante ; et il passe par différentes étapes, certaines étant facultatives. La première étape correspond à la description de l'architecture, grâce au langage π -Diapason et à la description de propriétés liées à cette même architecture, grâce au langage Diapason* . Cette description peut ensuite être vérifiée et modifiée autant de fois que nécessaire jusqu'à obtenir l'architecture souhaitée. Une fois l'architecture validée, cette dernière peut directement être exécutée par une machine virtuelle interprétant le π -calcul. Au cours d'une exécution, la description π -Diapason de l'architecture peut évoluer afin de corriger ou de prévenir une erreur potentielle, de prendre en compte de nouvelles contraintes ou encore de prendre en compte des changements dans les spécifications de l'orchestration. Avant de prendre en compte dynamiquement ces changements, la nouvelle architecture peut à nouveau être vérifiée et modifiée autant de fois que nécessaire. De même, l'expression des propriétés relatives à l'architecture en cours d'exécution peut elle aussi évoluer. Une fois la nouvelle architecture validée, il est alors nécessaire de vérifier l'état courant du processus en cours d'exécution afin de permettre ou non la prise en compte de la nouvelle architecture tout en gardant son état courant. Ce processus peut être répété autant de fois que nécessaire tout au long du cycle de vie de l'architecture. Ces modifications peuvent être appliquées sur une instance précise comme sur toutes les instances de l'architecture en cours d'exécution. De même, elles peuvent être répercutées ou non sur le modèle global de l'architecture.

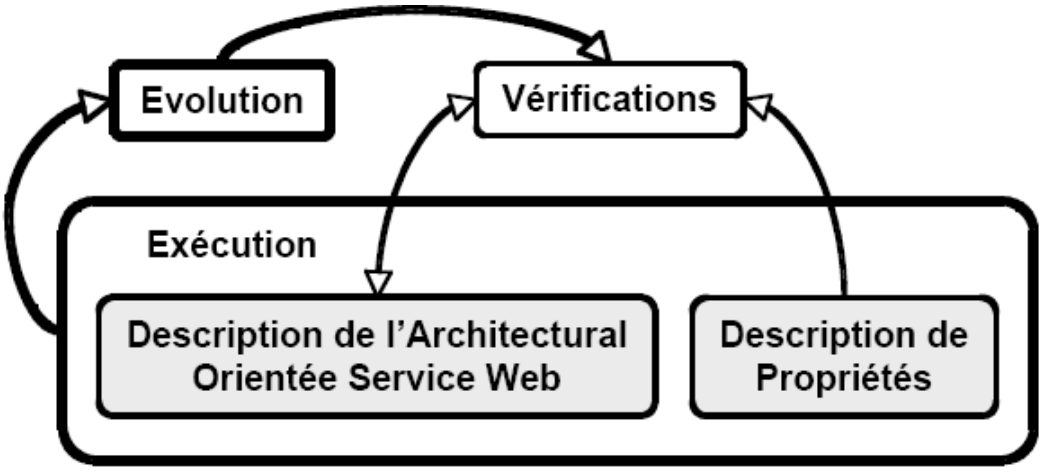

Figure III.7 Processus de développement architectural de l'approche Diaposan

Les différentes étapes de l'approche Diaposon , sont :

- ✓ La description de l'architecture, grâce au langage π -Diapason,
- ✓ La description de propriétés liées à cette même architecture, grâce au langage Diapason*,

- ✓ La simulation et l'extraction de tous les chemins d'exécution possibles de cette architecture,
- ✓ La vérification des différentes propriétés ; cette étape peut être suivie d'une modification de l'architecture et être réalisée autant de fois que nécessaire jusqu'à obtenir l'architecture souhaitée,
- ✓ L'exécution de l'architecture validée,
- ✓ L'évolution dynamique de cette architecture en cours d'exécution ; cette étape est accompagnée d'une nouvelle vérification des précédentes propriétés.

III. 14 Approche dirigée par les modèles (MDA-UML-S)

Dans ce travail [Nicola and Vaandrager, 1990], une approche dirigée par les modèles (MDA) fidèle aux principes définis par l'OMG est proposée pour spécifier, valider et mettre en œuvre la composition de services Web. Pour parvenir à cet objectif tout en conservant le méta-modèle UML standard, un profil (ou personnalisation) de UML2 nommé UML-S été défini pour adapter UML au domaine de la composition de services. UML-S permet de réaliser une spécification claire et compacte de la composition de services, à l'aide des diagrammes de classe et d'activité. Les diagrammes de classes sont utilisés pour décrire la partie statique de la composition, c'est à dire l'interface d'utilisation des services Web. Les diagrammes d'activités sont utilisés pour décrire la partie dynamique de la composition. Le processus de développement de cette approche est présenté dans la figure suivante :

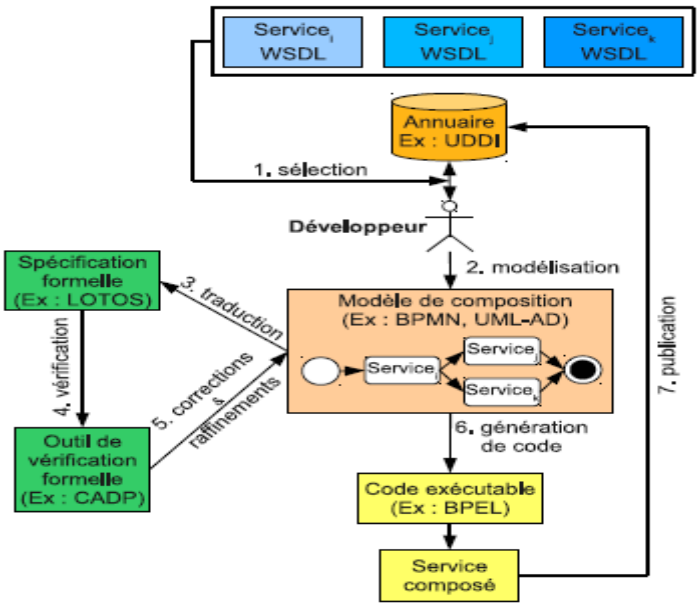

Figure III.13 Approche MDA (MDA-UML-S)

III. 15 Autres formalismes

Un langage stAC (Structured Activity Compensation) [Butler et Ferreira 2005] est défini afin de l'utiliser pour spécifier l'orchestration des activités dans le cadre des transactions de longue durée. Ce type de transactions utilise la compensation pour la gestion des exceptions. stAC considère les comportements séquentiels ou parallèles ainsi que la compensation et la gestion des exceptions. La notation B a été combinée au langage stAC pour spécifier les données des transactions. stAC a été utilisé pour décrire la sémantique d'une partie de BPEL. De leur côté, Yuan et al. [Yuan,Zhong 2006] ont défini un nouveau modèle appelé BFG (BPEL Flow Graph), qui est une extension du graphe de flot de contrôle (CFG), pour représenter les processus BPEL sous forme d'un modèle graphique. Ce modèle BFG a servi dans la génération de cas de test pour BPEL. Ces cas de test sont construits en combinant les chemins de test générés en traversant le modèle BFG, et les données générées par résolution de contraintes. Ce modèle BFG permet de spécifier le flot de contrôle de BPEL, le flot de données mais qu'une partie de la sémantique de BPEL.

Il existe dans la littérature d'autres travaux de modélisation de la composition de services par des diagrammes UML [OMG 2008]. Nous donnons, par exemple, les travaux de Cambronero et al. [Emilia , Gregorio 2007] qui ont utilisé les diagrammes UML pour décrire les comportements de processus BPEL ainsi que les contraintes temporelles qui y sont associées. Ils ont aussi proposé une méthode automatique de transformation de ces diagrammes UML en BPEL [LALLALI, 2009].

III. 16 Synthèse et Conclusion

Plusieurs approches ont été discutées dans ce chapitre, ces travaux visant à formaliser le comportement des services web. Les services Web sont très vite devenus des acteurs de processus complexes. Ces processus peuvent être abordés selon deux approches : la chorégraphie et l'orchestration. Seule l'orchestration de services Web donne une vision concrète qui permet l'expression d'un processus exécutable. Les orchestrations sont, pour la majorité, exprimées grâce au langage BPEL4WS.

Il ya des travaux s'appuient sur des STE comme LTSA-WS [Foster et al., 2003] [Foster et al., 2005] [Foster et al., 2006] ou sur les algèbres de processus, le π-calcul [Milner, 1989] [Milner, 1999] et π-Diapason [Frédéric , 2007] sont, quant à eux, un formalisme mathématique pour la description et l'étude des systèmes concurrents. Ils permettent la représentation de modèle avec précision, en utilisant les opérateurs de l'algèbre pour définir chacune des structures d'un processus. Parmi les travaux visant à formaliser et analyser BPEL4WS, certains comme ASDL [Solanki et al., 2006] sont fondés sur l'algèbre ITL (Internal Temporal Logic) [Cau et al., 2006] alors que d'autres [Salaun et al., 2004] [Chirichiello and Salaun, 2005] ont utilisé LOTOS [ISO/IEC, 1989].

Les théories temporelles sont apparues suite à l'application de la logique dans le domaine de l'Intelligence Artificielle. Ces formalismes ont été introduits pour représenter des connaissances temporelles, pour spécifier des domaines d'objets dynamiques ou encore afin de raisonner pour résoudre des problèmes ; comme. Les travaux de [Rouached et al., 2006a] [Rouached et al., 2006b] sont basés sur l'une de ces théories : Event Calculus [Kowalski and Sergot, 1986].

Les théories temporelles (Les travaux de [Rouached et al., 2006a] [Rouached et al., 2006b], FIACRE [Farail et Gaufillet 2008]), Les logiques temporelles, Les logiques temporelles arborescentes basées sur actions et La logique ACTL [Nicola and Vaandrager, 1990] sont apparues suite à l'application de la logique dans le domaine de l'Intelligence Artificielle. Ces formalismes ont été introduits pour représenter des connaissances temporelles, pour spécifier des domaines d'objets dynamiques.

Autres travaux proposés des approches semi formels pour la composition des services Web tel que : les réseaux de Pétri [Hamadi and Benatallah, 2003] ; et dirigée par les modèles comme (MDA-UML-S) [Nicola and Vaandrager, 1990].

Après avoir présenté les différents travaux et langages relatifs à l'orchestration de services Web, nous avons constaté qu'un certain nombre de points faibles n'étaient pas traités, ou tout du moins, pas de manière satisfaisante. :

Approches liés	Points forts	Points faibles
- **Les systèmes de transitions LTSA-WS**	- L'approche utilise des standards (diagrammes de séquence, BPEL4WS) accessibles qui masquent la complexité de FSP. - Elle fournit un moyen visuel (via l'outil LTSA) de comparer le modèle de conception et le modèle d'implémentation tout en vérifiant l'absence de violation de propriétés générales.	- L'approche désolidarise les phases de conception et d'implémentation pour les regrouper, une fois ces dernières indépendamment terminées. En cas de non cohérence (concordance) de traces, l'implémentation est donc totalement à reprendre : la phase de vérification est alors très/trop tardive. - UML étant semi-formel et BPEL4WS n'ayant aucune sémantique formelle, il est alors impossible de prouver que le FSP généré correspond exactement à ce qui est décrit. En ce sens, les comparaisons et les vérifications ne peuvent être que partielles. - La spécification BPEL4WS peut être interprétée par différents moteurs d'exécution, rien ne permettant de prouver que ces derniers aient exactement la même sémantique d'interprétation. De fait, ce qui est implémenté correspond certes, (partiellement) aux besoins définis lors de la conception, mais pas forcément à la réalité de l'exécution.
- Les réseaux de Petri	- Cette approche formalise certains opérateurs utilisés pour l'orchestration de services Web sous la forme d'un réseau de Pétri et permet ainsi certaines vérifications.	- Cette approche n'offre pas un moyen d'expression proche des standards du domaine des services Web. - Deuxièmement, les structures comportementales formalisées sont très restreintes et ne permettent pas la description d'orchestrations complexes. - Les réseaux de Petri offrent des mécanismes de simulation pour analyser des processus mais ne permettent aucune exécution de ces derniers. L'approche doit donc être complétée par une phase de traduction vers un langage exécutable, par exemple BPEL4WS. Ce dernier n'étant pas formel, aucune garantie ne peut donc être donnée quant à la correspondance exacte entre les opérateurs supportés par l'approche et leur traduction en BPEL4WS.
- Les algèbres de processus ASDL	- ASDL a une sémantique clairement définie puisqu'il est fondé sur une algèbre de processus. Il permet ainsi le raisonnement sur un processus. - Son haut degré d'abstraction lui permet de décrire une orchestration à différents niveaux (services, orchestration, protocoles).	- ASDL présente une syntaxe complexe, loin des standards du domaine des services Web. - Le fait de s'interfacer à un haut niveau d'abstraction nécessite un autre langage (de plus bas niveau) pour exécuter une orchestration décrite avec ASDL, par exemple BPEL4WS. Cette traduction, exempte de tout formalisme, ne permet pas d'assurer que ce qui sera exécuté correspond exactement à ce sur quoi ASDL à permis de raisonner auparavant.

		Plus encore, il n'est pas certain que tout ce qui a été formalisé trouve une structure d'implémentation (passage d'un haut vers un plus bas niveau d'abstraction).
LOTOS/CADP	- Le point fort de cette approche est le pouvoir d'expression des propriétés grâce l'outil CADP. En effet, cet outil permet l'expression de propriétés de sûreté et de vivacité spécifiques à un processus donné.	- Malgré son pouvoir d'expression, CADP reste un outil complexe à maîtriser. En effet, il n'est pas trivial de définir une propriété et encore moins de l'exprimer, d'autant plus que la syntaxe du langage ne fait pas référence aux concepts de l'orchestration de services Web. - Ce type de formalisation ne permet pas d'assurer une cohérence entre ce qui est vérifié et ce qui est exécuté. - L'approche impose, une traduction d'un algèbre de processus vers un langage sans sémantique formelle afin qu'une orchestration soit exécutée. Cette étape récurrente introduit une perte de sémantique, amplifiée par les ambiguïtés qui peuvent être ajoutées par les différentes implémentations des moteurs BPEL4WS .
Le π -calcul	- La particularité qui caractérise le π-calcul est l'introduction du concept de mobilité, c'est-à-dire la possibilité de modifier dynamiquement les liens entre différents processus ainsi que de déplacer un comportement d'un processus à un autre.	- Le π –calcul présente une syntaxe complexe, loin des standards du domaine des services Web. -Deuxièmement, le fait de s'interfacer à un haut niveau d'abstraction nécessite un autre langage (de plus bas niveau) pour exécuter une orchestration décrite avec π –calcul. Cette traduction, exempte de tout formalisme, ne permet pas d'assurer que ce qui sera exécuté correspond exactement à ce sur quoi π –calcul a permis de raisonner auparavant. Plus encore, il n'est pas certain que tout ce qui a été formalisé trouve une structure d'implémentation (passage d'un haut vers un plus bas niveau d'abstraction).
Les théories temporelles	- L'approche permet la vérification de propriétés fonctionnelles et non fonctionnelles. - Elle permet la vérification d'une orchestration BPEL4WS à un niveau statique (avant l'exécution) et tout au long de l'exécution d'une orchestration.	- La description des propriétés en Event Calculus demande une certaine expertise, éloignée des standards du domaine des services Web. - La phase de traduction entre BPEL4WS et son langage de formalisation reste présente comme dans les autres approches, ce qui induit les mêmes restrictions, à savoir de potentielles pertes de sémantiques. Ces dernières sont en parties compensées par la phase de vérification tout au long de l'exécution. Cependant la détection de dérives lors de l'exécution intervient une fois que ces dernières sont réellement arrivées.

Les logiques temporelles et La logique ACTL	- Leur capacité à exprimer l'ordonnancement des actions (événements) dans le temps .	- L'optimisation de l'un ou l'autre de ces aspects (L'expressivité, La complexité d'évaluation, La facilité d'utilisation) ne pouvant généralement se faire qu'au détriment des autres. - En raison de la diversité des logiques temporelles existantes et des résultats présents dans la littérature, il n'est pas toujours aisé de réunir les éléments pertinents pour choisir une logique temporelle adaptée à un certain contexte.
L'approche Diapason	- Décrire des architectures orientées service Web de manière totalement formelle. - Garantissant ainsi une unité quant à l'interprétation et à l'exécution d'une orchestration. Cette formalisation étant conservée de la phase de description à la phase d'exécution, il est alors possible de garantir que ce qui est exécuté correspond exactement à ce qui est décrit. - Le langage π-Diapason, qui offre un pouvoir d'expression supérieur aux langages d'orchestration actuels (par exemple BPEL4WS) et qui est de plus extensible.	- Complexité du syntaxe et l'utilisation du langage. -Certaines étapes de l'approche Diapason reste encore manuelles. - Le langage π-Diapason ne prendre pas en compte les quantificateurs existentiel et universel sur les actions. - Ne prendre pas en compte les propriétés dites non fonctionnelles.

Tableau III.1 Tableau comparatif des travaux liés

Toutefois, il convient de signaler que ces travaux ont été proposés, mais aucuns d'eux ne satisfait amplement les exigences de développements des applications web :

- ✓ Les travaux proposés sont axés autour l'aspect architectural (BPEL4WS) de développement et visant à formaliser le comportement des services web (Seule l'orchestration de services Web donne une vision concrète qui permet l'expression d'un processus exécutable. Les orchestrations sont, pour la majorité, exprimées grâce au langage BPEL4WS) sans aborder l'aspect technologique (AJAX) des applications web. Bien que cet aspect technologique a un effet essentiel sur le processus et le comportement général de l'application, y compris les services web partenaires eux-mêmes.
- ✓ Même pour l'aspect architectural ces propositions restent des solutions partielles aux problèmes de composition des services web sûres. Les spécifications générées sont trop abstraites pour être directement supportées par un langage d'implémentation. Ces spécifications correspondent au niveau conceptuel du développement considéré. Une étape de raffinement (codage) de ces spécifications devient donc indispensable.

Ainsi dans notre prochain chapitre nous allons présentons notre approche pour le développement des applications Web sûres.

Chapitre 04

-L’approche proposée-

IV
L'approche proposée

IV.1 Introduction

L'objectif de notre travail est la définition d'une approche formelle pour le développement des applications Web 2.0 sûres. Dans ce type d'applications, les concepteurs ont plutôt tendance à utiliser des méthodes dites semi-formelles telles: UML, OMT... basées principalement sur des notations graphiques (classes, entités, états/transitions,...) ou symboliques permettant une représentation intuitive, simplifiée et synthétique du système à étudier. Ces méthodes représentent des avantages indéniables pour la modélisation. Elles constituent un support idéal pour la communication entre les différents acteurs du système. Néanmoins, ces méthodes souffrent encore d'un manque de sémantique précise de leurs concepts. Ce manque de sémantique réduit considérablement toute possibilité de raisonnement et de preuve sur les modélisations ainsi obtenues.

Par ailleurs, en analysant les travaux de recherche existants sur la formalisation de développement des applications web, nous détectons certaines limites cruciales: la nécessité d'une nouvelle approche pour décrire tous les aspects descriptifs des applications web. Comme nous l'avons vu dans le chapitre précédent, toutes les approches séparant entre l'aspect architectural (Orchestration des service web) et l'aspect technique (AJAX) lors de développement, et ce dernière est complètement ignoré lors de la formalisation des applications (services) web, ce qui peut réduire considérablement l'efficacité de formalisation des applications développées , et finalement, il n'y a aucune approche complète qui traite le développement des applications web tenant compte les deux aspects architectural et technologique.

Afin de répondre aux limitations citées ci-dessus, nous proposons une approche complète qui propose une formalisation de processus de développement des applications web 2+ tenant compte les deux côtés architectural et technique.

Notre travail s'inscrit dans le cadre de la génération d'une implémentation à partir de spécifications (abstraites) formelles B (notations mathématiques). Cette génération est réalisée par raffinements successifs des spécifications abstraites obtenues par la phase de traduction. Pour cela un ensemble de règles génériques de raffinement B a été défini. Ces règles considèrent l'aspect dynamique (opérations) de l'application Web étudiée.

L'objectif d'une telle traduction est l'obtention d'une modélisation non seulement précise et concise mais surtout analysable par des outils (contrôle de types, preuves, simulation, …).

IV.2 L'approche proposée

Une notion importante lors de la conception des applications web 2+ est celle d'activité. D'une part les activités permettent de décrire les différents chemins d'interaction entre les services web, et permettant d'atteindre un objectif particulier. D'un coté, la conception d'une orchestration fait appel en particulier à des descriptions d'activités des services web permettant de décrire les actions et ou tâches qu'un service est sensé réaliser au sein d'une orchestration. Il est possible de relier les services web entre eux par des relations événementielles. Nous les représenterons par un graphe CTT (structure classique utilisée pour représenter des relations) [Paterno, 2000,2003].

De l'autre coté, le langage Java, comme tout langage moderne, permet de créer des applications qui ressemblent à l'interface du système d'exploitation. Cette assurance d'ergonomie et d'interactivité avec l'utilisateur est le minimum qu'un utilisateur demande à une application. Les interfaces homme-machine (dénommées IHM) font intervenir de nos jours des éléments que l'on retrouve dans la majorité des systèmes d'exploitation : les fenêtres, les menus déroulants, les boutons, etc...

Il est possible de relier ces objets graphiques entre eux et avec les utilisateurs par des relations événementielles. Nous les représenterons aussi par un graphe CTT.

Les deux, aspects (Architecturale-service- et technologique-IHM-) avoir considéré des tâches de haut niveau, très globales, puis les transforment progressivement en tâches plus typées .La structuration arborescente décrit l'organisation de l'application. La disparition de tâches globales au profit de tâches plus précises en ce qui concerne les opérations entre les services partenaires (BPEL4SW) d'un coté et les actions IHM (Java) de l'autre coté, s'accompagnent par l'explicitation des liens organisationnels et comportementaux entre ces tâches.

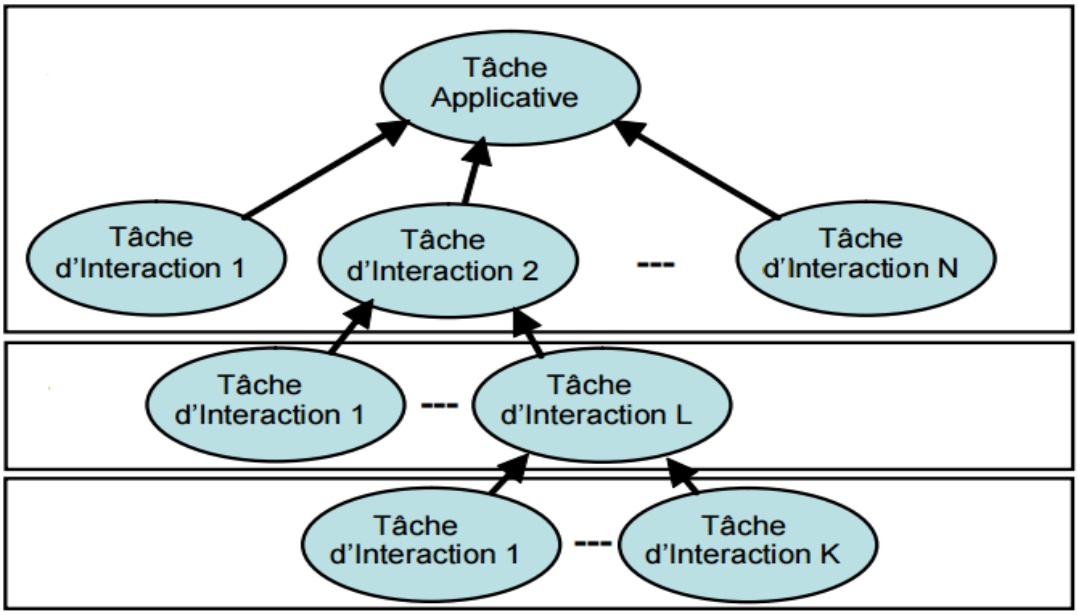

Figure IV.1 Mise en relation des Tâches applicative

Dans notre domaine (les applications web) la spécification abstraite de composition des taches (services web et composants ergonomiques) signifiée la spécification de nature de composition entre deux tâches, il peut prendre deux formes :

Ordonné : détermine si une composition peut exécuter les tâches composées séquentiellement ou en parallèle.

Alternative : indique si une composition peut invoquer des tâches alternatives jusqu'à ce que l'une d'entre elles réussie.

Nous représentons la décomposition des activités (SOA et IHM) de l'application web en utilisant le raffinement. Cette démarche permet la prise en compte des modèles des taches (CTT) utilisé dans les développements en B événementiel [Abrial, 1996] comme le montre la figure suivante :

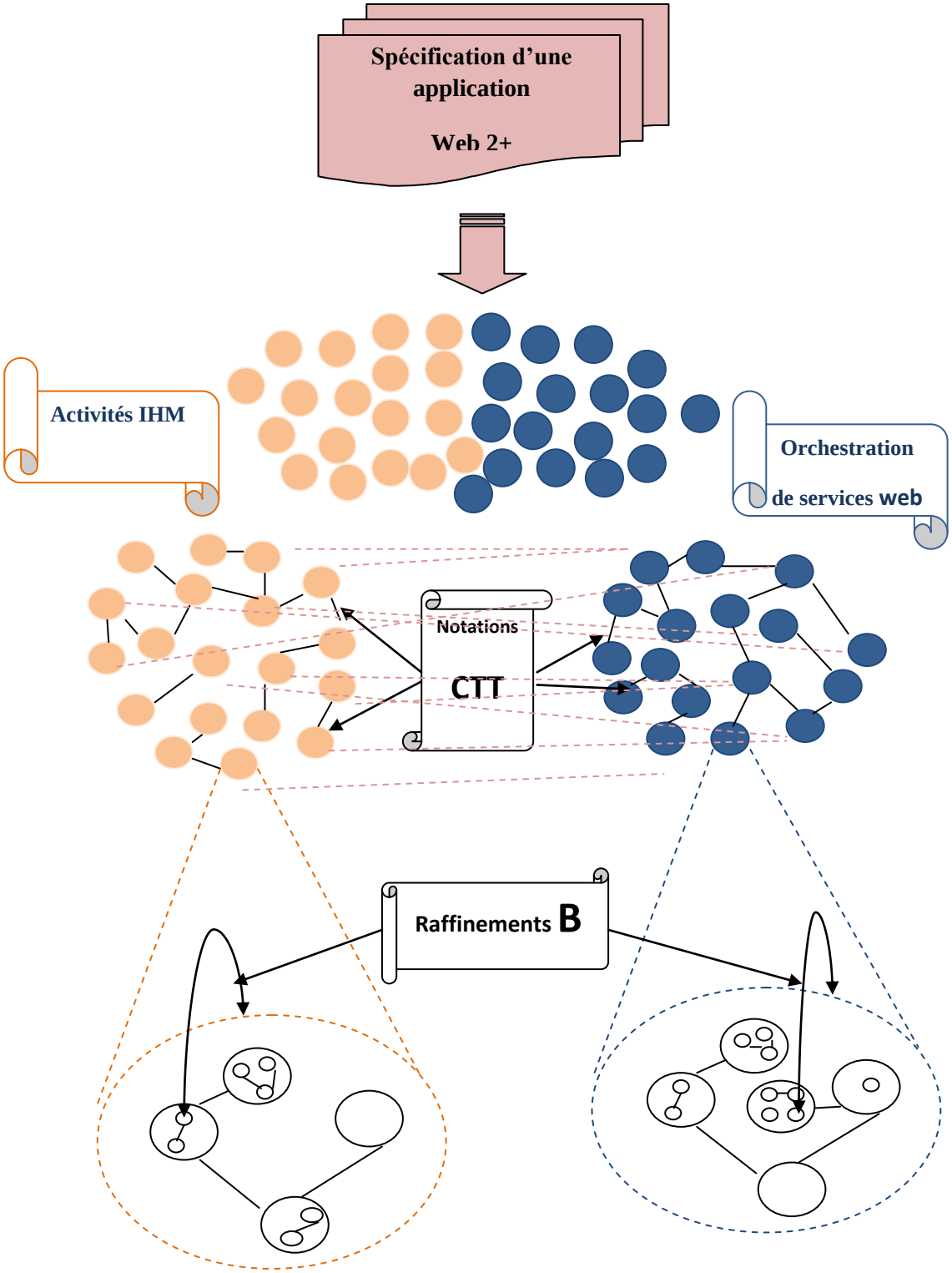

Figure IV.2 Décomposition et raffinement des activités.

IV.2.1 Les techniques utilisées

Des tâches (activités) applicatives sont issues du cahier des charges de l'application tel qu'exprimé par les futurs utilisateurs. Il s'agit donc de tâches que les utilisateurs voudront accomplir. Ces tâches peuvent être multiples et variées et ne sont aucunement standardisées quant à leur nom, leur contenu, etc.

IV.2.1.1 CTT (Concur Task Tree)

Nous avons exploité les modèles des tâches pour la spécification des taches de l'application. Notre choix s'est porté sur CTT [Paterno, 2000,2003] pour les raisons suivantes :

- ✓ Model CTT est une structure hiérarchique représentée sous forme d'arbre.
- ✓ CTT est un modèle de tâches fondé sur la décomposition par une approche descendante ;
- ✓ Permet la description de tâches en combinant les tâches via des opérateurs temporels.
- ✓ CTT permet d'exprimer précisément les différentes actions qui sont réalisées en utilisant des expressions d'une algèbre de processus ;
- ✓ CTT est une notation définie pour la spécification de modèles de tâches.
- ✓ Les modèles de tâches sont utilisés dans le développement à des fins de conception et/ou de validation.

Choix	*T[]T,*
Indépendance d'ordre	*T|=|T,*
Entrelacement	*T |||T,*
Synchronisation	*T|[]|T,*
Désactivation	*T[>T,*
Interruptibilité	*T |>T,*
Ordre séquentiel	*T>>T,*
Ordre séquentiel avec passage d'information	*T[]>>T,*
Processus infinie	*T**
Itération finie	*| TN*
Processus Atomique	*| Tat*

Figure IV.3 Grammaire de langage CTT

D'un point de vue des systèmes de transitions, une tâche CTT représente un système de transitions codé en B événementiel. Cette tâche peut représenter soit le système dans le cas d'une spécification par une tâche, soit représenter une tâche à valider. Les modèles B événementiels associés à ces deux tâches décrivent des systèmes de transitions qui sont reliés par une relation de simulation.

IV.2.1.2 La méthode formelle B

La méthode B [Abrial, 1996] initiée par J.R. Abrial ; simultanément avec Z et VDM et industrialisée grâce à la convention B : STÉRIA, RATP, SNCF et INRETS Puis par : ALSTHOM, MTI.

La méthode B est une méthode formelle qui permet le raisonnement sur des systèmes complexes ainsi que le développement logiciel. Elle permet, d'une part, de vérifier la cohérence globale de system et, d'autre part, d'assurer, la conformité du code final par rapport à sa spécification initiale.

Les constituants de base du langage B sont :

- ✓ Machines abstraites (modules)
- ✓ Théorie des ensembles
- ✓ Substitutions généralisées
- ✓ Raffinement avec obligations de preuves

Une machine abstraite décrit un système à travers un ensemble de *VARIABLES* dites états. Le typage et les contraintes régissant ces variables, écrits dans une variante de la théorie des ensembles de Zermalo- Frankel(ZF), sont exprimés dans la clause *INVARIANT* de la machine.

```
MODEL nameM
    REFINES nameR
    SETS . . .
    PROPERTIES . .
    VARIABLES . . .
    INVARIANT . . .
    ASSERTIONS…
    INITIALISATION . . .
    OPERATIONS . . .
END
```

Figure IV.4 Structure de machine B

La partie dynamique du système est spécifiée par un ensemble d'opérations B. Les opérations sont décrites dans le langage des substitutions généralisées de Disjkstra :

Substitution élémentaire : x **:=** e

Précondition : **PRE** P **THEN** S **END**

Garde : **SELECT** P **THEN** S (**WHEN** Pi **THEN** Si)* **END**

Choix non déterministe borné : **CHOICE** S1 **OR** S2 ... **END**

Choix non déterministe non borné : **ANY** X **WHERE** P **THEN** S **END**

Parallèle : S1 **||** S2

Identité : **skip**

Figure IV.5 Substitutions généralisées de Disjkstra

La substitution de base est l'assignation d'une valeur à une variable. La généralisation de ce langage permet de définir des substitutions plus élaborées: substitution non déterministe, substitution préconditionnée,… .

Une substitution préconditionnée est de la forme:

$$SELECT\ p\quad THEN\ s\quad END,$$

p étant un prédicat et S une substitution.

Si p est vérifié alors la substitution S est exécutée correctement, sinon S échoue:

Le résultat de l'exécution de la substitution S est imprévisible.

En B, chaque étape de raffinement est validée par l'établissement d'un ensemble d'obligations de preuves, générées automatiquement, qui assure la correction des différentes transformations opérées par le raffinement. La méthode B est implémentée par plusieurs outils, dont Click'n'Prove/B4Free v2.

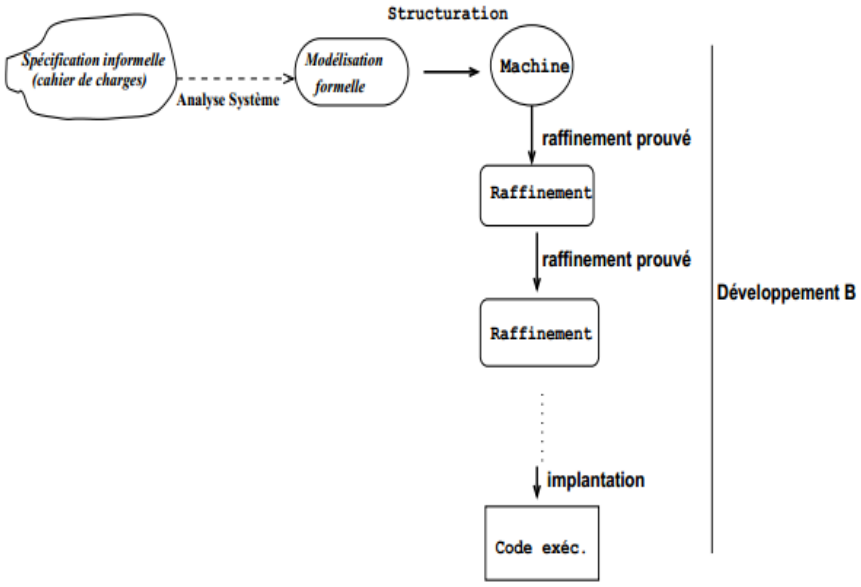

Figure IV.6 Structuration d'un développement en B

IV.2.2 Meta model et liens structurels entre spécifications B, Service Web et diagrammes de classes (Java)

En vue de faire le lien entre les aspects structurels de systèmes spécifiés en B , Service Web et classe Java, nous proposons un méta-modèle de ces concepts (Service web et Class Java) pour B. Ensuite, nous adoptons une approche interprétée où nous identifions des correspondances entre éléments du méta-modèle de (Service web et Class Java) et un sous-ensemble de constructions en B. Aussi, l'élaboration de schémas de transformation précis et explicites possible.

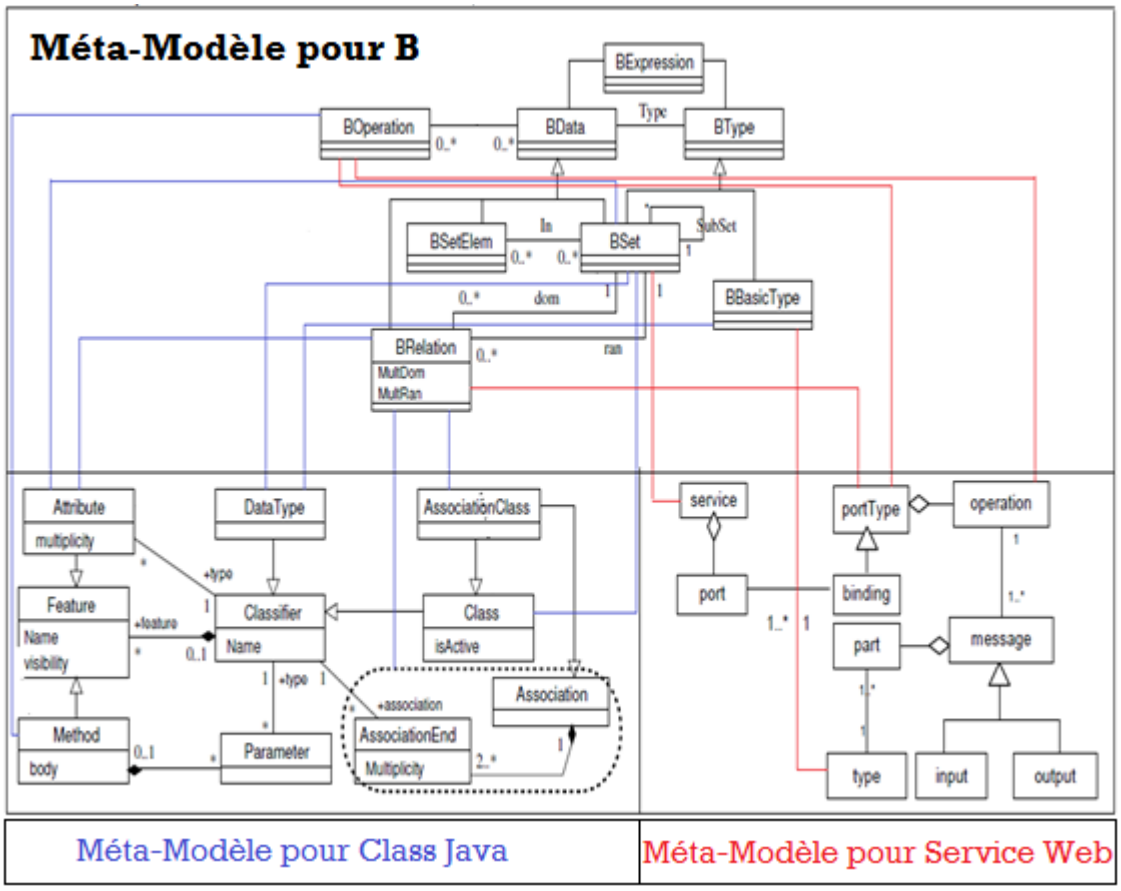

Figure IV.7 Correspondances entre méta-modèles B, Service Web et Class Java

Dans la figure ci-dessus, nous présentons le méta-modèle B (*MB*) proposé en relation avec un méta-modèle simplifié de Service web (*MS*) et Class Java (*MC*). Notons que pour des raisons de clarté et de place nous avons choisi de ne présenter que les concepts dont nous nous servons pour notre travail. Les correspondances possibles entre les éléments des ces méta-modèles sont visualisées par des flèches et sont formellement définies par les relations *R* suivantes :

$$R_{BS} \in MB \longleftrightarrow MS$$

$$R_{Bc} \in MB \longleftrightarrow MC$$

Soient les fonctions d'abstraction *AB* , *AS* et *AC* qui associent à chaque élément issu d'une machine B et à chaque élément d'un service web et chaque élément d'une classe Java respectivement un méta-concept des méta-modèles respectifs :

$$AB \in B \longrightarrow MB$$
$$AS \in S \longrightarrow MS$$
$$AC \in C \longrightarrow MC$$

Où B dénote l'ensemble des concepts issus d'une machine B, et S l'ensemble des concepts issus d'un diagramme de service web, et C l'ensemble des concepts issus d'un diagramme de classes Java. Nous nous basons sur les méta-modèles B, S et C précédents ainsi que sur les relations R$\mathcal{BS}$]⊔ R$\mathcal{B}$/ en vue d'effectuer les relations entre les spécifications B,S et C En effet, la traduction d'un élément d'une machine B en un élément d'un service web et un élément d'une classe C doit satisfaire respectivement les relation R$\mathcal{BS}$]⊔ R$\mathcal{B}$/ entre les deux méta-modèles associées aux deux éléments B et S d'une part , et B et C de l'autre part.

Les correspondances entre les concepts B ,S et C sont définies par la relation *MappingBS* et *MappingBC* comme suit :

$$MappingBS \in \mathrm{B} \longleftrightarrow \mathrm{S}$$
$$(a,b) \,.\, ((a,b) \in MappingBS \qquad (\mathrm{AB}(a),\mathrm{AS}(b)) \in R_{BS})$$
$$MappingBC \in \mathrm{B} \longleftrightarrow \mathrm{C}$$
$$(c,d) \,.\, ((c,d) \in MappingBC \qquad (\mathrm{AB}(c),\mathrm{AS}(d)) \in R_{BC})$$

- ✓ La correspondance établie de la méta-classe *BSet* vers la méta-classe *Service* la méta-classe *Class* est formalisée par : ⇐$\mathcal{BS}$]⊔⇔$\mathcal{S}$]∇⊑)/]⇒∈ $\mathcal{RB}$/ et ⇐$\mathcal{BS}$]⊔⇔$\mathcal{C}$⇳⊣//⇒∈ $\mathcal{RBC}$ et signifie qu'un ensemble abstrait peut être transformé en un *service* (cas de service web) ou une *classe* (cas class Java).
- ✓ Les opérations d'une spécification B sont traduites en *opération* de service web et en des *méthodes* de classes. Les données B manipulées par ces opérations sont spécifiées par la méta-classe *BData* et sont définies au niveau des clauses *SETS*, *VARIABLES* et *CONSTANTS*.
- ✓ Une relation en B peut être traduite par : un *portType (partner Link)* entre services web, une *association* entre classes, *un attribut* de classe ou alors *une classe associative*.

Les règles des traductions (B-BPEL4SW) et (B-JAVA) produits par notre approche à partir de spécifications B sont des instances possibles des *MappingBS et MappingBC*. L'élaboration de schémas de transformations explicites basés sur cette relation permet, d'une part, l'automatisation du processus de génération de code par vues comportementales.

IV.2.3 L'Architecture générale de l'approche proposée

L'approche proposée est conçue pour la spécification, la vérification formelle et la mise en œuvre des applications web. En effet, le développeur travaille à un niveau d'abstraction élevé en élaborant des modèles de type CTT pour décrire le comportement (architectural-BPEL4SW- et technologique –JAVA). CTT peut être critiqué pour son manque de formalisme. IL permet de réaliser des modèles précis et clairs pour décrire des systèmes selon différents points de vue, par exemple structurel ou comportemental. Cependant, la sémantique de CTT n'étant pas formellement définie, il n'est pas possible d'appliquer d'outils mathématiques sur les modèles obtenus. L'analyse et la vérification formelle des modèles CTT n'est donc pas possible directement. L'analyse formelle du code obtenu à partir des modèles n'est également pas directement possible car les langages de composition actuels tels que BPEL4SW souffrent des mêmes défauts que CTT sur ce point.

De plus, il est préférable de détecter les erreurs le plus tôt possible dans le cycle de développement, dès l'étape de spécification. Une solution proposée pour régler ce type de problème consiste à faire appel aux méthodes formelles. Nous proposons ainsi de transformer les modèles CTT en spécifications formelles. Tout langage de spécification formelle serait susceptible de convenir mais nous proposons l'utilisation de la méthode B qui est étudiée en détail dans ce chapitre dans le paragraphe IV.2.1.2. La méthode B présente l'avantage d'être standard, basé sur des règles mathématiques, formel et pris en charge par des outils de vérification formelle tel que les outils Click'n'Prove/B4Free v2, il est ainsi possible de valider de manière automatisée des propriétés comportementales (architectural-BPEL4SW- et technologique–JAVA) de l'application. Ceci permet notamment de prouver que la spécification élaborée par le développeur réalise bien le comportement désiré et donne effectivement le résultat attendu.

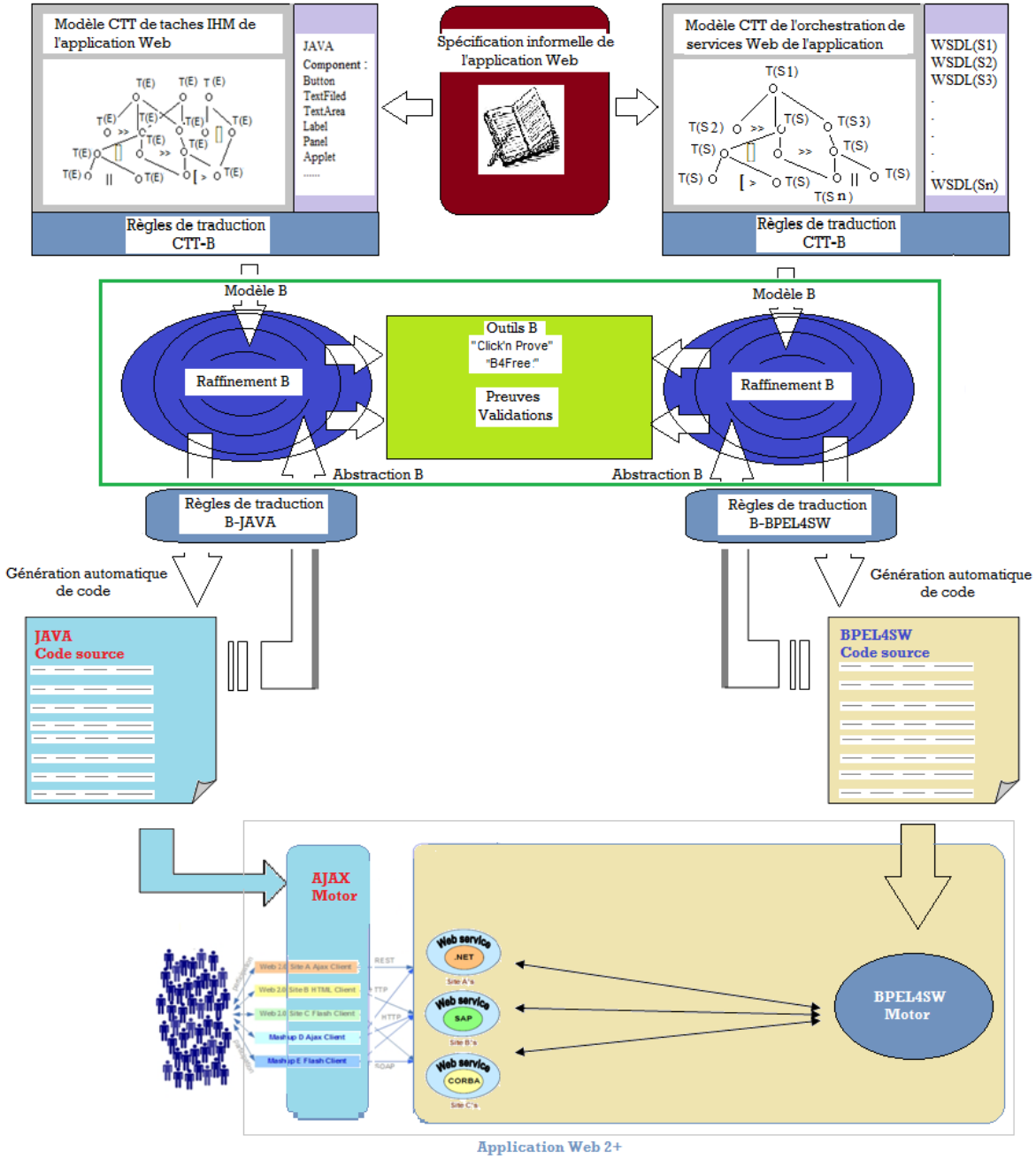

Figure IV.8 L'architecture générale de l'approche proposée

Nous allons donner dans les sections suivantes les étapes impliquées dans l'approche de développement proposée afin de mieux comprendre la manière de procéder.

IV.2.4 Modèle pour la spécification

Comme illustré dans la figure IV.8, le développeur commence par sélectionner des tâches (activités) de l'application web. D'une part un annuaire de services tel qu'UDDI est souvent utilisé pour trouver des services partenaires fournissant les fonctionnalités demandées. La découverte des services est en effet un principe important dans toute architecture de type SOA. Les services doivent être conçus pour être suffisamment descriptifs pour permettre leur découverte et leur accès à travers des mécanismes de recherche.
D'autre part les composants (IHM) Java doivent ainsi être définis pour être suffisamment descriptifs pour permettre leurs spécifications.

Une composition des services web ou des composants IHM peut être spécifiée par la composition de tâches élémentaires sous forme des expressions CTT. Ainsi, une spécification de composition des tâches se présentera sous la forme :

$$T_{appweb} = (T(E))^{*} \,|||\, (T(S))^{*}$$

où les *Ti* sont les tâches élémentaires de types :

$$(T(S))^{*} = (T(S1) op1 T(S2)\ op2 \ldots\ opm\ T(Sm))^{*}$$

$$(T(E))^{*} = (T(E1)\ op1 T(E2)\ op2 \ldots\ opn\ T(En))^{*}$$

S : Service Web

E : Composant (IHM)

T(Ei) : Activité (IHM)

T(Sj) : Activité d'un service web

opx ϵ Grammaire de langage CTT

Par exemple :

$$T^{*} = ((T1 [] T2) >> T3 >> T4)^{*}$$

Correspond une infinité de scénarios (traces d'exécution) possibles :
T1 – T3 − T4
T2 – T3 − T4
T1 – T3 − T4 – T2 – T3 − T4
T2 − T3 − T4 − T1 – T3 − T4
T1 − T3 − T4 − T1 – T3 − T4−T2−T3…..
…..

Le développement d'une application web se fera en codant l'expression de tâches CTT (arbre) par des raffinements B en utilisant les schémas de raffinement des activités présentés par CTT. Les événements apparaissant dans les feuilles de l'arbre de tâche constituent les événements de base utilisés dans la conception des messages d'orchestration au sein d'un processus de composition des services web d'une part, et du contrôleur de dialogue (IHM) d'autre part.

Les spécifications B peuvent être réparties en plusieurs machines abstraites reliées entre elles par les clauses (SEES, USES, INCLUDES et IMPORTS). Au niveau le plus abstrait, l'objectif d'une telle modularisation est de réduire la taille et la complexité des machines, d'améliorer ainsi la lisibilité de la spécification et d'alléger la phase de preuve.

IV.2.5 Modèle pour la vérification et la validation

La vérification généralement automatisée grâce aux nombreux outils logiciels existants pour la méthode formelle B. Il est ainsi possible de prouver que les tâches composées réalisent bien la fonctionnalité demandée. Dans le cas où la spécification est décrite en B, il est par exemple possible d'utiliser les outils Click'n'Prove/B4Free v2 pour la vérification de propriétés de système (L'application). Ces propriétés décrivent les caractéristiques importantes du comportement de différents composants (Services Web et composants ergonomique de Java). La méthode B permet de prouver que ces propriétés sont vérifiées (ou non), dans le cas où des erreurs sont détectées grâce à la vérification formelle, le développeur est chargé de corriger et raffiner son modèle jusqu'à arriver à un modèle prouvé correct ; et donc de valider les modèles des compositions.

En ce qui concerne la formalisation, deux approches peuvent globalement être différenciées :

La première a pour objectif d'exprimer une application (architecture et IHM) avec un langage formel CTT-B, afin de la vérifier, puis traduire cette dernière en code cible (BPEL4WS, JAVA) comme dans la figure suivante :

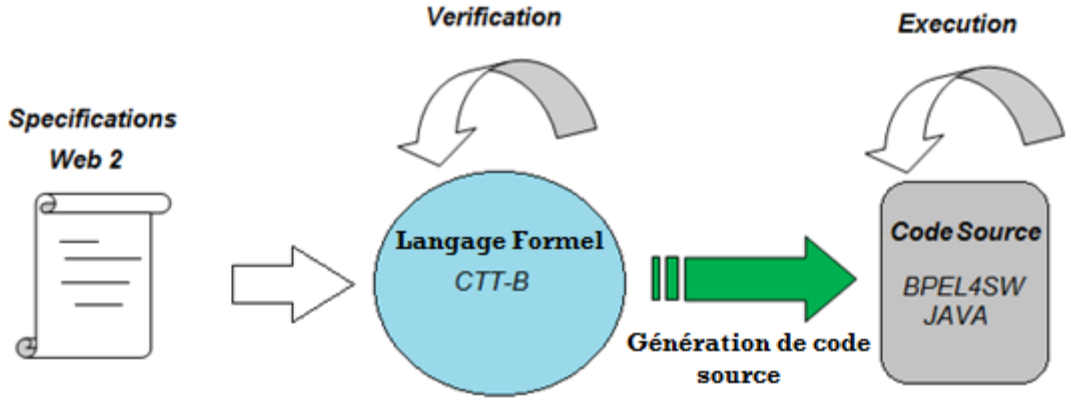

Figure IV.9: Approche descendante (B-BPEL4SW, B-JAVA)

La seconde consiste à partir des codes (BPEL4WS-JAVA) de l'application de les traduites dans un langage formel B afin de le vérifier comme dans la figure suivante :

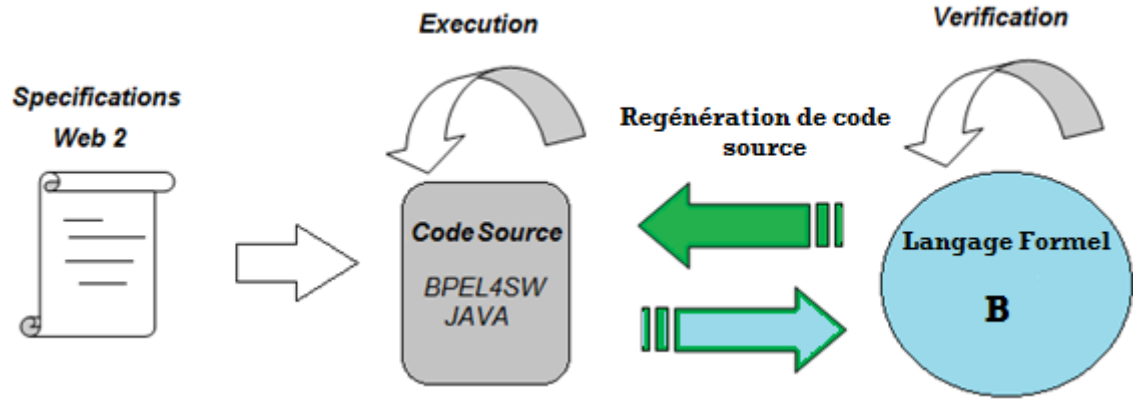

Figure IV.10: Approche Ascendante (BPEL4SW-B, JAVA-B)

A l'image de la spécification, on peut utiliser le codage des modèles de tâches CTT pour effectuer la validation. Le principe est le suivant :

Une application web est décrite par un ensemble d'événements décrivant les différentes interactions possibles (BPEL4SW, JAVA). Ils peuvent avoir été produits par un raffinement d'une tâche CTT. La validation d'une tâche consiste à produire un développement de cette tâche par raffinement selon une décomposition CTT qui atteint les événements de base décrivant le système (les événements des feuilles de l'arbre). Dans ce cas, ce développement devra respecter les propriétés du système de l'*INVARIANT* et des *ASSERTIONS.*

IV.2.6 Modélisation des propriétés statiques et dynamiques

Nous pouvons distinguer deux types de propriétés mises en évidence dans l'application: les propriétés statiques et les propriétés dynamiques :

- ✓ Les propriétés statiques représentent le cadre global du composant et doivent apparaître en *INVARIANT*. et pour cela il ya deux cas :
  - Lorsque la propriété statique à exprimer fait intervenir des variables définies dans un composant B donné et qui sont modifiées par les mêmes opérations. Elle doit être introduite au niveau de la clause *INVARIANT* de ce composant. Ainsi, l'initialisation et les opérations de ce composant doivent respecter cette propriété.
  - Lorsque la propriété statique à exprimer lie des variables provenant de composants B distincts ou affectées par des opérations différentes. Elle doit apparaître dans la clause *INVARIANT* du composant liant ces modules B entre eux.

- ✓ Les propriétés dynamiques caractérisent la loi d'évolution des données pour un service ; elles mettent donc en corrélation, les anciennes et nouvelles valeurs des données pour chaque état atteint. Ces propriétés sont locales à une opération et sont formalisées dans des *Post-Conditions*. Elles ne peuvent pas être exprimées dans des clauses INVARIANT. Ces propriétés sont satisfaites après le déroulement du corps de l'opération.

Lorsque le modèle de tâche est validé, l'étape suivante est l'implémentation. Comme expliqué précédemment, le code exécutable peut-être généré à partir de la spécification. En fonction du niveau de détails dans la spécification, le code généré sera plus ou moins complet et nécessitera donc éventuellement l'intervention du développeur. Les règles de traductions présentées dans les sections suivantes permettent de générer le code dans son intégralité, tout en fournissant des modèles clairs et lisibles.

IV.2.7 Le raffinement

Le raffinement est au cœur de la méthode B. Il permet de ne considérer dans un premier temps que l'abstraction d'un logiciel avant de prendre en compte les détails de conception et d'implémentation. L'automatisation du raffinement permet d'accélérer le développement d'un logiciel, en confiant à un outil la transformation mécanique d'un modèle abstrait complet en un modèle implémentable. Un tel outil, appelé raffineur automatique, transforme une machine en un ensemble de machines, raffinement et implémentations, Le processus de raffinement automatique se décompose alors en un grand nombre de petites transformations (qui conduisent à des obligations de preuve plus simples à prouver) [MethodeB , 2015] .

IV.2.7.1 Les types de raffinement

En B, on distingue deux types de raffinement :

- ✓ **Raffinement de données:** c'est le remplacement d'une donnée abstraite DA par une donnée concrète DC. Dans ce cas, un prédicat P, appelé invariant de collage, est à préciser. Cet invariant permet d'établir le lien entre les données DA et DC. L'invariant de collage P est spécifié dans la clause **INVARIANT** du composant raffinant.

  Dans notre domaine (les applications web) l'invariant de collage permet d'établir le lien entre les données WSDL (Partie abstraite : types, messages, portType et partie concrète : binding, service) des services composés avec les mêmes données WSDL des services composants.

- ✓ **Raffinement algorithmique:** c'est la transformation d'une substitution abstraite SA en une substitution moins abstraite SC (ex : remplacement d'une substitution simultanée par une substitution séquentielle, élimination des préconditions).

Ces deux types de raffinement ne sont pas exclusifs : ils peuvent être opérés dans la même étape de raffinement. Il est évident que tout raffinement de données entraîne un raffinement algorithmique. Dans notre domaine (les applications web) on applique ce type de raffinement sur les opérations élémentaires des activités des services, nous considérons que chaque service (tâche) comme un ensemble des opérations interagissent entre eux par un ensemble de messages.

IV.2.7.2 La dépendance des messages

La dépendance des messages définit la correspondance des messages d'inputs et d'outputs. Il y a trois types de dépendance :

- ✓ **La synthèse :** Ce type combine les messages d'output des services composants pour former les messages d'output du service composé.
- ✓ **La décomposition :** ce type décompose le message d'input du service composé pour générer les messages d'input des services composants.
- ✓ **La correspondance de messages :** permet de faire la correspondance entre les inputs et les outputs des services composants.

IV.2.7.3 Niveaux des raffinements proposés pour les applications Web2+

Le raffinement est le processus de transformation de spécifications abstraites de composition des tâches (architecturales et techniques) en spécifications plus concrètes. De manière générale, l'étape de raffinement constitue la phase la plus critique et difficile du développement d'un projet B. Il n'existe pas de méthode précise qui permettrait de raffiner n'importe quelle spécification abstraite vers un langage cible donné. En effet, une telle méthode devrait être en mesure, pour chaque implémentation possible, de déterminer les variables concrètes et aussi les invariants de collage liant ces dernières aux variables abstraites de la spécification. Néanmoins, la définition d'un processus de raffinement spécialisé pour un domaine bien précis est possible [A.Mammar] .

Pour notre domaine (les applications web 2+) nous proposons les niveaux suivants des raffinements :

Niveaux de raffinements	*Spécifications*	*Raffinements*
Processus d'orchestration	Qui décrit la manière dont les services Web peuvent interagir ensemble au niveau des messages, incluant l'ordre d'exécution des interactions (des messages) (souvent qualifiée de "logique métier"). L'orchestration décrit quant à elle comment les services Web peuvent interagir entre eux selon une perspective opérationnelle, avec des structures de contrôle. Dans l'orchestration, un seul processus, appelé orchestrateur, est responsable de la composition et contrôle les interactions entre les différents services. Cet orchestrateur coordonne de manière centralisée les différents services partenaires qui n'ont aucune connaissance de cette composition. L'identité des services Web est alors connue. L'orchestration donne une vision concrète qui permet l'expression d'un processus exécutable.	Dans le premier raffinement, l'orchestration est raffinée en des messages décrivant les activités (La partie dynamique) de l'orchestration des services web. Ces activités liées par un flot de contrôle. Ces activités peuvent être basiques, structurées ou des communications : **A .Les activités basiques :** <wait>, <assign>, <throw>, <terminate>, <empty>. **B. Les activités structurées** : Les activités structurelles définissent l'ordre dans lequel les activités imbriquées sont exécutées : <sequence>, <switch>, <while>, <pick>, <flow> , <scope>, <compensate> **C. Les activités de communication** : <receive>, <reply>, et <invoke>. Ces activités représentent les événements d'interaction avec les services partenaires et peuvent être déclenchés dans un ordre quelconque.
Service	Un Service Web est un regroupement logique d'une ou plusieurs opérations : Dans la pratique ce n'est pas stricto sensu un service Web qui est invoqué mais une opération de ce service Web.	Dans ce niveau de raffinement, chaque service partenaire est raffiné en des opérations décrivant les activités de service. incluant l'ordre d'exécution des interactions (des messages : souvent qualifiée de "logique métier") dans le service. Ces activités liées par un flot de contrôle On distingue deux types d'opérations de service Web : - *Les opérations complexes :* une opération complexe invoque d'autres opérations (du même et/ou d'autres services Web) au sein de son propre comportement. - *Les opérations atomiques* : Une opération atomique se suffit à elle-même dans le sens où elle n'a pas besoin d'invoquer d'autre opération de service Web au sein de son comportement interne. L'invocation d'une opération atomique n'excède pas un simple échange synchrone ou asynchrone de deux messages. Par contre, une opération complexe est dotée d'un comportement défini par le séquencement d'échanges de messages avec diverses autres opérations (atomiques ou complexes). Une opération complexe peut elle même, être vue comme le résultat d'une orchestration. En d'autres termes, le processus exécutable décrit par le biais d'une orchestration peut être déployé comme une opération de service Web et ainsi faciliter son utilisation et sa réutilisation au sein d'autres orchestrations. Le nombre de niveaux d'imbrication est alors

		illimitée.
Opérations sur des objets des classes.	Il est possible aussi de relier certains objets entre eux par des relations événementielles. Une opération désigne un service qu'une classe offre à ses Utilisateurs. En pratique, il ya 5 sortes d'opérations sur un objet : - *Modificateur :* une opération qui altère l'état d'un objet -*Sélecteur* : une opération qui accède à l'état d'un objet, mais qui n'altère pas celui-ci. - *Itérateur* : une opération qui permet d'accéder à toutes les parties d'un objet dans un ordre bien défini. -*Constructeur* : une opération qui crée un objet et/ou initialise son état. -*Destructeur : Une* opération qui libère l'état d'un objet et/ou détruit l'objet lui-même.	Ce niveau de raffinement détaille les événements de la récupération des paramètres et des valeurs (Exemples dans l'étude de cas) ou des autres interactions entre les objets de notre application.
Les événements IHM	Nous pourrons construire un logiciel qui réagira sur les interventions de l'utilisateur si nous arrivons récupérer dans notre application les messages que le système envoie. Java autorise aussi la consultation de tels messages. En Java, le traitement et le transport des messages associés aux événements sont assurés par des objets dans le cadre d'un modèle de communication dénommé le modèle de traitement des événements par délégation (Delegation Event Model). Les événements sont gérés par plusieurs interfaces EventListener. Les interfaces EventListener permettent de définir les traitements en réponse à des événements utilisateurs généré par un composant. Une classe doit contenir une interface auditrice pour chaque type d'événements à traiter. La gestion des événements est d'une importance capitale pour les programmes ayant une interface utilisateur graphique. En effet, en mode graphique, ce n'est plus une programmation déterministe, qui impose donc un fonctionnement séquentiel, mais plutôt une programmation qui propose un ensemble d'actions spécifiques, au moment où l'utilisateur le désire et en rapport avec des événements particuliers, comme par exemple le clic d'un bouton	Dans ce raffinement, Les événements IHM sont raffinés pour faire apparaître les modalités d'interaction de l'application avec l'utilisateur. Nous pouvons considérons l'existence de trois modalités clavier , souris et la voix . Les événements possibles dans Java sont des objets (un événement est un message contenant plusieurs informations sur les états des touches de clavier, des paramètres,...) dont les classes sont dans le package java.awt.event. -*ActionListener* : clic de souris ou enfoncement de la touche Enter -*ItemListener* : utilisation d'une liste ou d'une case à cocher -*MouseMotionListener* : événement de souris -*WindowListener* : événement de fenêtre

Tableau IV.1: Niveaux des raffinements proposées pour les applications Web2+

La méthode B propose plusieurs clauses d'architecture permettant ainsi une spécification et un développement incrémentale :

Au niveau machines abstraites :

- ✓ **SEES** : accès aux ensembles et constantes, accès aux variables en lecture dans les opérations
- ✓ **USES** : accès aux ensembles et constantes, accès aux variables en lecture dans les opérations et dans l'invariant
- ✓ **INCLUDES** : Une machine A peut être incluse au plus une seule fois dans une machine B ; accès aux ensembles et constantes, accès aux variables en lecture dans les opérations et dans l'invariant, appel des opérations en lecture.

  La clause INCLUDES permet de regrouper dans une machine abstraite les variables, les constantes, les ensembles avec leurs propriétés, provenant d'autres machines abstraites.

  La clause INCLUDES permet donc de décomposer une machine abstraite complexe en plusieurs machines abstraites, tout en facilitant le travail de preuve associé. En effet, la preuve d'un composant et des ses machines « incluses ¸ est globalement plus simple que la preuve du composant équivalent non décomposé. L'interprétation d'une inclusion est simple : les variables et constantes de la machine incluse deviennent des entités de la machine incluante, et les opérations incluses deviennent des « morceaux ¸ de spécification" utilisables.

Au niveau implémentations :

- ✓ **SEES** : accès aux ensembles et constantes, appels des opérations en lecture
- ✓ **IMPORTS** : accès aux ensembles et constantes, accès aux variables en lecture dans l'invariant, appels des opérations en lecture . La clause IMPORTS permet donc à une implantation de réaliser sa spécification en utilisant les données et services d'autres programmes vus par leur spécification. La décomposition de problèmes en sous-problèmes comme la factorisation de services utilisables par plusieurs opérations sont réalisées par l'utilisation du lien IMPORTS.

- ✓ **PROMOTES :** La clause PROMOTES n'est pas une clause d'architecture, mais elle permet à une machine abstraite (respectivement une implantation) de promouvoir des opérations appartenant à des machines incluses (respectivement importées), c'est-à-dire que les opérations concernées deviennent des opérations de la machine d'accueil, proposées à l'extérieur.

- ✓ **EXTENDS :** La clause EXTENDS est définie par les équivalences suivantes :

  Dans une machine ou un raffinement : **EXTENDS** M signifie **INCLUDES** M **PROMOTES** <toutes les opérations de M >

  Dans une implantation : **EXTENDS** M signifie **IMPORTS** M **PROMOTES** <toutes les opérations de M >‹[CLEA, ATELIER B].

Ces clauses permettent une séparation des preuves associées à différentes machines.

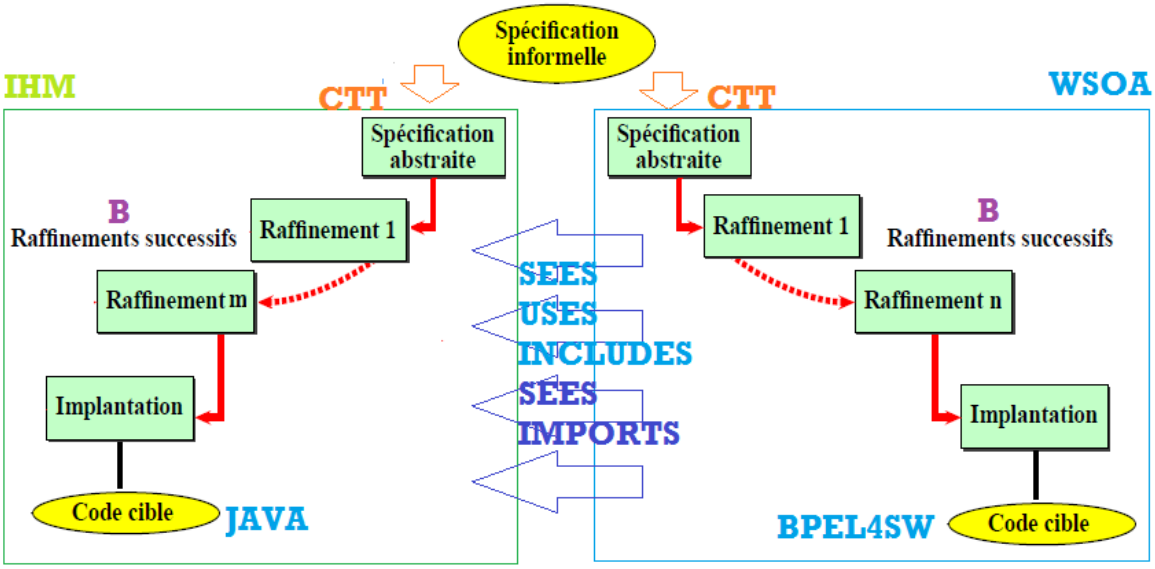

Figure IV.11: Approche de raffinement pour les applications Web 2+

Le dernier niveau de raffinement d'une machine abstraite est appelé implémentation (décrite dans un langage appelé B0). Le langage décrivant cette implémentation utilise des structures de données et programmes supportés par le langage de programmation cible choisi (BPEL4SW et JAVA). La traduction de cette implémentation vers le langage cible devient donc très naturelle et est potentiellement automatisable.

IV.2.8 La décomposition de complexités en B pour une application Web 2+

La spécification complète d'une application Web 2+ est quelque chose de complexe, contenant beaucoup de parties différentes. Même si le spécificateur a pu traduire toutes ces parties par des formules mathématiques, ces formules sont trop nombreuses pour être groupées dans une seule machine abstraite.
Une machine de regroupement qui inclut toutes les parties. La preuve de ce raffinement pose alors des problèmes de taille.

Exemple (Mauvaise composition des taches) :

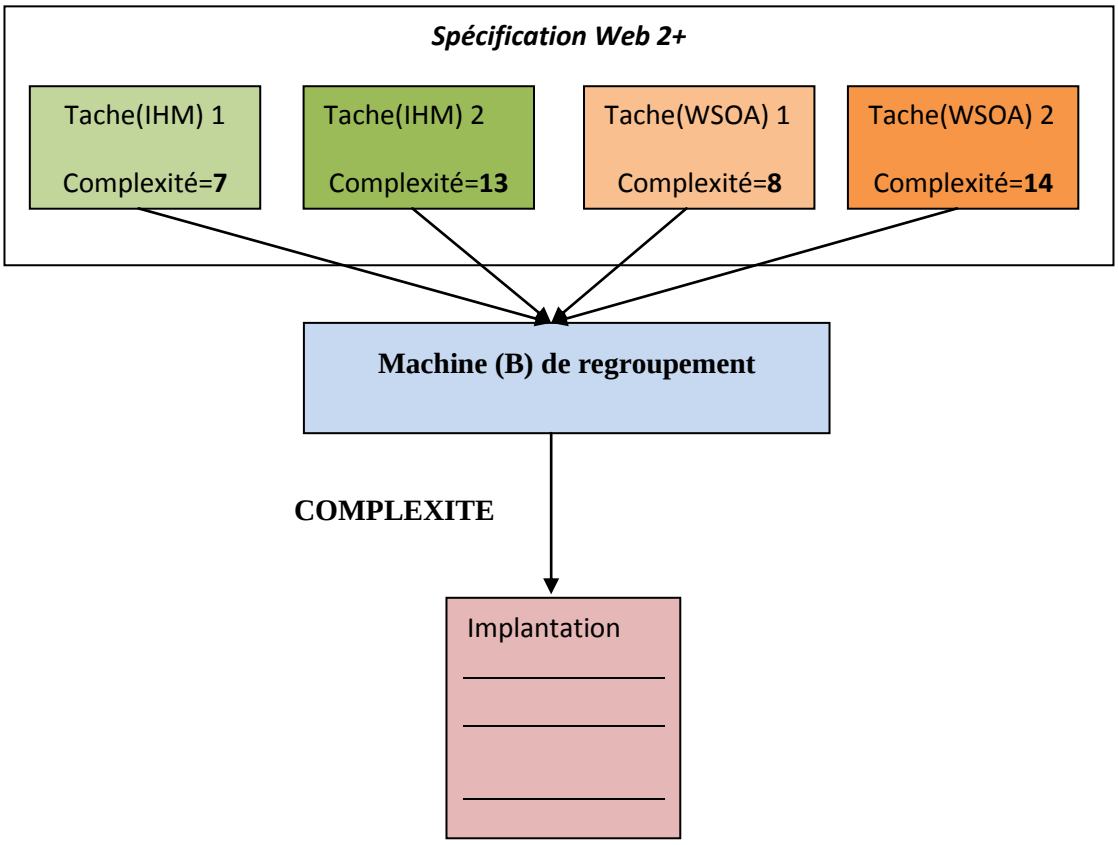

Figure IV.12: Mauvaise composition des taches

En B, ce problème se résout en utilisant des spécifications plus abstraites. Si notre spécification complète se décompose en un certain nombre des taches, supposons (exemple) que nous puissions mesurer la complexité de ces taches et leur affecter une valeur :
Nous devons trouver comment rassembler ces spécifications sans dépasser un certain niveau de complexité, disons Complexité = 16 dans notre système de mesure imaginaire. Nous allons regrouper ces spécifications en les représentant par groupes représentés chacun par une spécification plus abstraite, qui ne peut pas contenir tous les détails du groupe.

Cette représentation est faite en implantant (clause IMPORTS) la spécification plus abstraite sur les éléments du groupe :
La complexité des spécifications abstraites est inférieure à la somme des complexités de spécifications détaillées couvertes car il n'y a pas tous les détails de ces dernières. Par contre, dans ces spécifications plus abstraites il est possible de prouver des propriétés qui découlent des propriétés combinées des spécifications recouvertes. Ceci se généralise jusqu'au plus haut niveau :
L'une des conséquences de cette décomposition est que la machine de plus haut niveau est
simple, même s'il s'agit d'un logiciel très compliqué .

Exemple (Bonne composition des taches) :

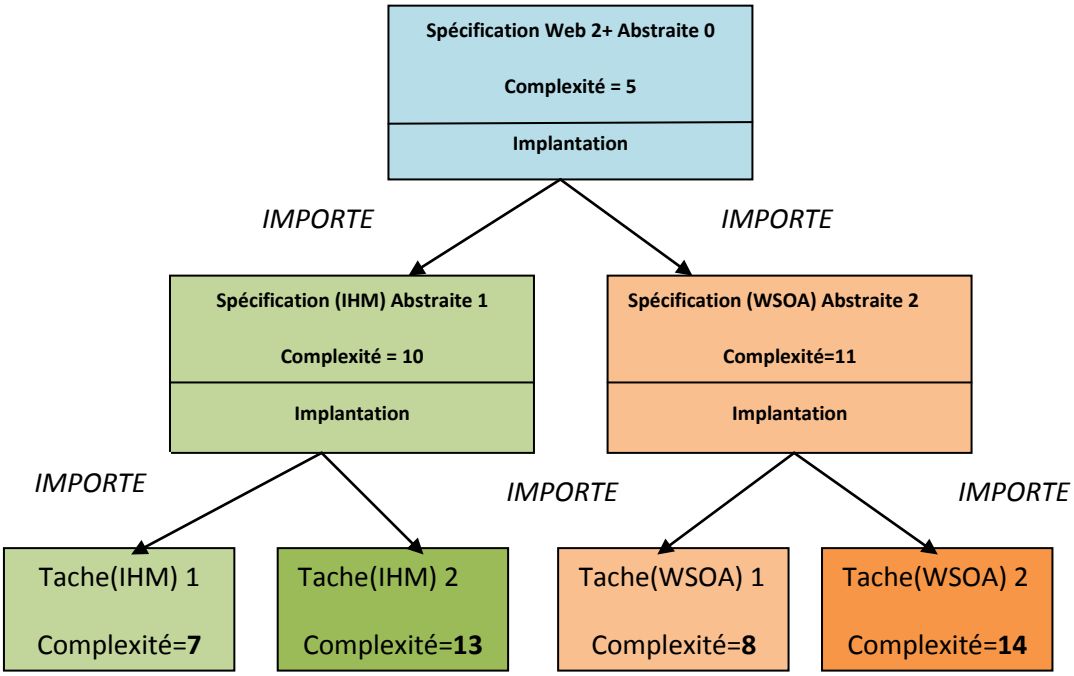

Figure IV.13: Bonne composition des taches

Ce type de démarche impose que l'on connaisse les propriétés qu'il faut prouver sur l'ensemble du l'application : ce sont les seules qui figurent dans les spécifications les plus abstraites.
Les autres détails ne doivent pas encore apparaître. Nous pensons que la détermination précoce de ces propriétés essentielles est le moyen fondamental pour maîtriser le processus de développement d'une application complexe : savoir ce que l'on veut avant de le faire.

Ces propriétés forment donc ce que l'on veut au départ : la vraie spécification de plus haut niveau de notre application. C'est en partant de ces propriétés et en faisant apparaître graduellement les détails, en construisant l'arbre de haut en bas que l'on développe la spécification complète. Cette

spécification complète est donc un arbre contenant des implantations. Le même style de démarche d'introduction graduelle de détails est également utilisé dans les raffinements.

Une spécification sans description simple et exacte, dont la seule description est une accumulation de détails, doit être réétudiée avant l'écriture des composants pour pouvoir être exprimée en B. Cette phase courante dans les projets B s'appelle la réexpression du besoin [CLEA, ATELIER B].

IV.2.9 Développement pratique pour la génération automatique de code

IV.2.9.1 Structure générale

Comme illustré dans le méta modèle (B-Service Web- Classe JAVA) dans la figure IV .7 nous pouvons proposons une correspondance entre les structures des concepts : machine B, processus BPEL4SW et classe en Java:

Machine B	Processus PBEL4SW	Class Java
MACHINE (Nom de la machine abstraite)	<Process>	Nom de la classe
ENSEMBLES	(Process PBEL c'est un model Compartmental pas des types abstraits)	Déclaration des ensembles abstraits et énumérés
VARIABLE	<Variable>	Déclaration de variables (mêmes attributs d'une classe)
INVARIANTE	<Partner Links>	Propriétés des variables
INITIALISATION		Mise en place de la valeur initiale des variables (même constructeur)
OPÉRATIONS	<Sequence>	Déclaration des opérations (mêmes méthodes).

Tableau IV.2: Génération automatique du code (Structure Générale)

IV.2.9.2 Les Règles de traduction

IV.2.9 .2.1 Les règles de codage descendant (B-BPEL4SW)

Les spécifications (architecturales) abstraites (B), sont raffinées jusqu'à l'obtention du niveau d'abstraction (en langage B) correspondant à la formalisation des concepts du code source BPEL4SW. Ce niveau d'abstraction a été défini comme un ensemble d'affectations. Ces affectations sont ordonnancées par un ensemble de structures de contrôle en BPEL4SW :

Composition séquentielle

Composition conditionnel if then else…

Boucles

IV.2.9 .2.2 La traduction ascendante (BPEL4SW-B)

Capturer l'aspect comportemental de code source (BPEL4SW) .Puis représenter l'exécution des ces instructions comme un ensemble de processus : séquentiels ou parallèles composés d'affectations en B suivant le tableau IV.2.

Le modèle abstrait (Architectural) de processus d'orchestration se compose des événements principaux qui sont : *Invoke Service 1* et *Invoke Service 2,…..*, *Invoke Service n* dont le rôle est d' invoquer une opération dans un service partenaire (Services 1, Service 2,…Service n) . Elle peut servir à la fois pour une communication synchrone ou asynchrone. Ces événements sont synchronisés par des variables : *invokeS1* , *invokeS1,… invokeSn* (qui ont un comportement booléen (1/0) pour indiquer que le service est appelé ou non) ; et les variable *S1pick*, *S2pick,…, Snpick* pour (qui ont un comportement booléen (1/0) pour indiquer que le service est en état d'attendre l'arrivée d'un événement ou non).

Specifications CTT (Semi Formelle)	Model B (Formel)	Code source (BPEL4SW)
Sequential composition: S1 >> S2>> … >>Sn	**VARIABLES** ***Invoke Service 1, Invoke Service 2,….., Invoke Service n*** **INVARIANT** ***invokeS1 ϵ (0,1)∧ invokeS2 ϵ (0,1) ∧…∧ invokeSn ϵ (0,1)*** ***S1 pickϵ (0,1)∧ S2 pick ϵ (0,1) ∧…∧ Sn pickϵ (0,1)*** ***invokeS1≠ invokeS2≠……≠ invokeSn*** ***S1 pick ≠S2 pick ≠…≠ Sn pick*** **EVENTS** … ***Invoke Service 1* = SELECT** ***invokeS1= 1^ S1pick=1*** …	**<sequence>** S1 S2 … Sn **<sequence>**

	THEN ***invokeS1:= 0 ^ S1pick:= 0 II…*** ***invokeS2 := 1 II…*** ***…*** ***END;*** ***Invoke Service 2* = SELECT** ***invokeS2= 1^ S2pick=1*** ***..*** ***THEN*** ***invokeS2:= 0 ^ S2pick:= 0 II…*** ***invokeS3:= 1 II…*** ***…*** ***END;*** **…….** ***Invoke Service n = SELECT*** ***invoke S n = 1^ S1pick=1*** ***..*** ***THEN*** ***invoke S n := 0 ^ Snpick:= 0 II…*** ***…*** ***END;*** **END;**	
Conditional compositios: **T(S1) []*T(S2)***	**VARIABLES** ***Invoke Service 1, Invoke Service 2*** **INVARIANT** ***invokeS1 ϵ (0,1)^ invokeS2 ϵ (0,1)*** ***S1 pickϵ (0,1)^ S2 pick ϵ (0,1)*** ***Cond ϵ (0,1)*** ***invokeS1≠ invokeS2*** ***S1 pick ≠S2 pick*** **EVENTS** **…** ***Invoke Service 1* = SELECT** ***invokeS1= 1^ S1pick=1^ Cond=1*** ***…*** **THEN** ***invokeS1:= 0 ^ S1pick:= 0 II…*** ***…*** ***END;*** ***Invoke Service 2* = SELECT** ***invokeS2= 1^ S2pick=1^ Cond=0*** ***…*** **THEN** ***invokeS2:= 0 ^ S2pick:= 0 II…*** ***…*** ***END;*** **…….** **END;**	**if Cond then T(S1)** **else *T(S2)***
Iterative loops composition: ***T(S1) ****	**VARIABLES** ***Invoke Service 1*** **INVARIANT** ***invokeS1 ϵ (0,1)***	***While Condi {T(S1)}***

	S1 pick∈ (0,1) ***Condi ∈ (0,1)*** **EVENTS** **…** ***Invoke Service 1* = SELECT** ***invokeS1= 1^ S1pick=1^ Condi=1*** ***…*** **THEN** ***invokeS1:= 1 ^ S1pick:= 1 II…*** ***…*** ***END;*** **…….** **END;**	

Tableau IV.3: Génération automatique du code BPEL4SW (règles de traductions) .

S : Service Web

T(S) : Tache d'un service web

IV.2.9 .2.3 Les règles de codage descendante (B-JAVA)

Les spécifications (ergonomiques) abstraites (B), sont raffinées jusqu'à l'obtention du niveau d'abstraction (en langage JAVA) correspondant à la formalisation des concepts du code source JAVA. Ce niveau d'abstraction a été défini comme un ensemble d'affectations. Ces affectations sont ordonnancées par un ensemble de structures de contrôle en JAVA :

Composition séquentielle

Composition conditionnel if then else…

Boucles

IV.2.9 .2.4 La traduction ascendante (JAVA-B)

Capturer l'aspect comportemental de code source (JAVA) .Puis représenter l'exécution des ces instructions comme un ensemble de processus : séquentiels ou parallèles composés d'affectations en B suivant le tableau Tableau IV.2.

Le modèle abstrait (d'interface IHM) se compose de des événements principaux qui sont Composant1Active, Composant2Active,…, ComposantMActive dont le rôle d'activer les composants (IHM) Java . Ces événements sont synchronisés par des variables : *déclenchC1* , *déclenchC2 ,… ,déclenchCn* (qui ont un comportement booléen (1/0) pour indiquer que le composant est déclenché ou non) ; et les variable : *C1ecoute, C2ecoute,…, Cnecoute* pour (qui ont un

comportement booléen (1/0) pour indiquer que le composant est en état d'écouter l'arrivée d'un événement ou non).

Specifications CTT (Semi Formelle)	Model B (Formel)	Code source Java
Sequential composition: C1 >> C2>> ... >>Cm	**VARIABLES** *Composant1Active, Composant2Active,..., ComposantMActive* **INVARIANT** *déclenchC1 ϵ (0,1)∧ déclenchC2 ϵ (0,1) ∧...∧ déclenchCm ϵ (0,1)* *C1ecoute ϵ (0,1)∧ C2ecoute ϵ (0,1) ∧...∧ Cmecoute ϵ (0,1)* *déclenchC1 ≠ déclenchC2 ϵ (0,1) ≠...≠ déclenchCm* *C1ecoute ≠ C2ecoute ≠...≠ Cmecoute ϵ (0,1)* **EVENTS** ... *Composant1Active* = SELECT *déclenchC1= 1^ C1ecoute= 1* ... THEN *déclenchC1:= 0 ^ C1ecoute := 0* *déclenchC2 := 1 II...* ... END; *Composant2Active* = SELECT *déclenchC2= 1^ C2ecoute=1* .. *THEN* *déclenchC2:= 0 ^ C2ecoute := 0* *déclenchC3:= 1 II...* ... *END;* ... *ComposantMActive* = SELECT *déclenchCm= 1^ Cmecoute =1* .. THEN *déclenchCm := 0 ^ Cmecoute := 0* ... END; END;	...C1...; ... C2...; ... ...Cn...;
Conditional compositios: **C1**[] ***C2***	**VARIABLES** *Composant1Active, Composant2Active* **INVARIANT** *déclenchC1 ϵ (0,1)∧ déclenchC2 ϵ (0,1)* *C1ecoute ϵ (0,1)∧ C2ecoute ϵ (0,1)* *Cond ϵ (0,1)* *déclenchC1 ≠ déclenchC2 ϵ (0,1)* *C1ecoute ≠ C2ecoute* **EVENTS** ...	**if Cond then T(C1)** **else *T(C2)***

	Composant1Active = SELECT *déclenchC1= 1^ C1ecoute= 1^ Cond=1* … THEN *déclenchC1:= 0 ^ C1ecoute := 0* … END; *Composant2Active* = SELECT *déclenchC2= 1^ C2ecoute=1 ^Cond=0* .. THEN *déclenchC2:= 0 ^ C2ecoute := 0* … END; … END;	
Iterative loops composition: ***T(S) 1 ****	VARIABLES *Composant1Active* INVARIANT *déclenchC1 ϵ (0,1)* *C1ecoute ϵ (0,1)* *Condi ϵ (0,1)* EVENTS … *Composant1Active* = SELECT *déclenchC1= 1^ C1ecoute= 1^ Condi=1* … THEN *déclenchC1:= 1^ C1ecoute := 1* … END; ……. END;	***While Condi {T(C1) }***

Tableau IV.4: Génération automatique du JAVA (règles de traductions).

Remarque: les boucles ne sont autorisées que dans les implantations. Ceci évite d'avoir des boucles spécifiant d'autres boucles, ce qui poserait des problèmes de preuves. D'autre part, il est déconseillé de faire de boucles imbriquées. La boucle interne doit être rejetée dans une machine importée [CLEA, ATELIER B].

A partir des tableaux (IV.2, IV.3 et IV.4) nous avons [Meftah,Kazar et Hani. 2016].développé un outil de transformation des codes (B-BEL4SW,B-Java) :

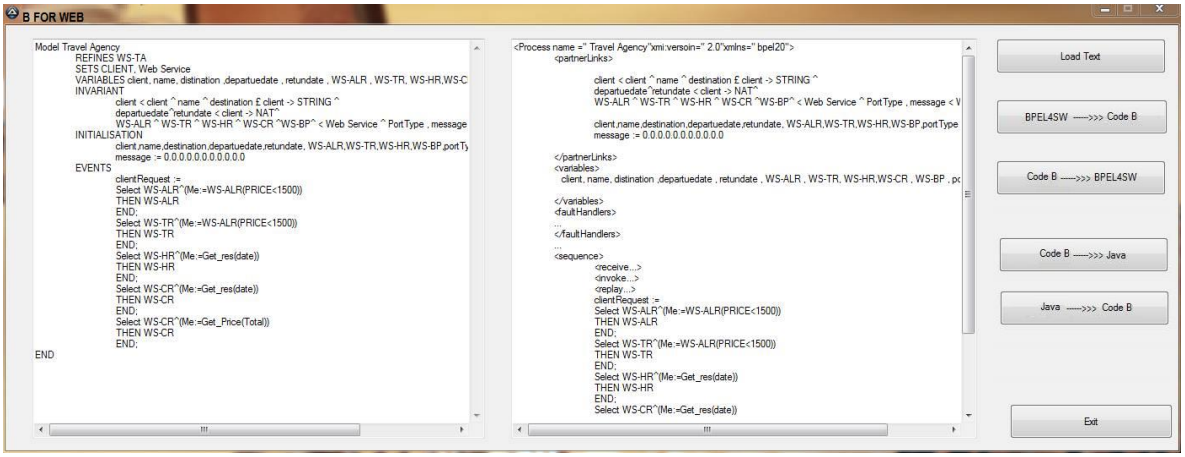

Figure IV.14: Un outil de transformation de code (B-BPEL4SW, B-JAVA)

Pour le développement de cet outil, nous avons utilisé le système (AutoIt3) est un logiciel d'automatisation de tâches. Il utilise un langage de script similaire au BASIC vous permettant d'automatiser plusieurs tâches sous Windows il peut même effectuer des recherches automatiquement, créer des fichiers, créer un disque virtuel, ouvrir un dossier, ouvrir une fenêtre, gérer les processus en cours, et bien d'autres encore. AutoIt est un logiciel gratuit compatible avec Windows 95/98/2000 et XP.

IV.3 Conclusion

Dans ce chapitre, nous avons proposé une approche formelle pour le développement des applications Web 2+ décrite architecturalement en langage BPEL4SW et technologiquement basée sur le langage JAVA. Nous avons défini un méta modèle pour les concepts (B-Service Web-Classe JAVA) sa sémantique et ses différents fondements mathématiques. Ce modèle prend en compte les aspects comportementaux des applications Web 2+ .Ainsi nous avons présenté les différentes phases de l'architecture générale de l'approche proposée ainsi leurs concepts tels que le raffinement, la corrélation des messages, les règles de traduction. L'approche proposée c'est une approche formelle descendante/ ascendante pour la spécification, la validation, la vérification formelle et la génération automatique des codes (BPEL4SW, JAVA).

Pour un système logiciel trois stratégies complémentaires sont généralement reconnues et utilisées (voir par exemple : « Software Engineering » par Ian Sommerville) : Eviter les fautes (zéro-défaut), Tolérer les fautes et Détecter les fautes. Dans le cadre du développement logiciel, la méthode B répond à la première de ces stratégies (zéro-défaut) de façon très pertinente. En effet, en utilisant le langage B il est possible de prouver que le logiciel implanté est exempt de fautes par rapport à sa spécification. Aucune faute résiduelle ne peut donc rester dans le logiciel puisque la correction est prouvée. Bien sûr la spécification peut ne pas refléter exactement le besoin du client, c'est pourquoi le (zéro-défaut) ne signifie pas que le logiciel répond totalement au besoin du client.

Dans le chapitre suivant, nous présenterons une étude cas pour le développement d'une application Web 2+ (Agence de voyage) , y compris nos concepts proposées, et les différents outils utilisés afin de nous permettre la transformation de la description d'une spécification abstraite CTT en une spécification formelle en B, et la génération automatique des code BPEL4SW et JAVA .

Chapitre 05

-Etude de cas-

V

Etude de cas

V.1 Introduction

Pour illustrer les différents idées et concepts proposés dans cette thèse nous allons présenter un exemple sous la forme d'une étude de cas afin de monter la fonctionnalité et la mise en évidence de nos propositions. Nous avons choisir les activités d'une agence de voyage comme un domaine d'application. Alors l'objectif de cet exemple est de développer une application Web 2+ (agence de voyage) selon notre approche.

Une agence de voyages (service AV) utilisé les cinq services suivants : réservation de billets d'avion (service01), réservation de billets de train (service 02) réservation de chambres d'hôtel (service03) et de location de voiture (service04). A fin de fournir ces services à ses clients, l'agence doit établir des liens avec d'autres services : compagnies aériennes, train, des hôtels, des compagnies de location de voiture et des banques (service05) pour faciliter les transactions financières entre les clients et l'agence de voyages et entre l'agence et les autres partenaires. A cet effet six services web peuvent être proposés :

- ✓ WS-AV: fournit une interface avec le client, et établir des liens avec les autres services,
- ✓ WS-VOL : permet de réserver les vols selon une date précisée, la ville de départ et d'arrivée et le nombre de personnes.
- ✓ WS-Train : permet de réserver les trains selon une date précisée, la ville de départ et d'arrivée et le nombre de personnes.
- ✓ WS-Hotel : permet de réserver les chambres d'hôtel selon la date et la ville précisé et le nombre de personnes.
- ✓ WS-LV : permet de réserver les voitures selon une date précisée et les nombres de jours.
- ✓ WS-Bank : permet à l'agence et les clients de payer les différentes réservations.

La figure suivante présente les services de notre exemple :

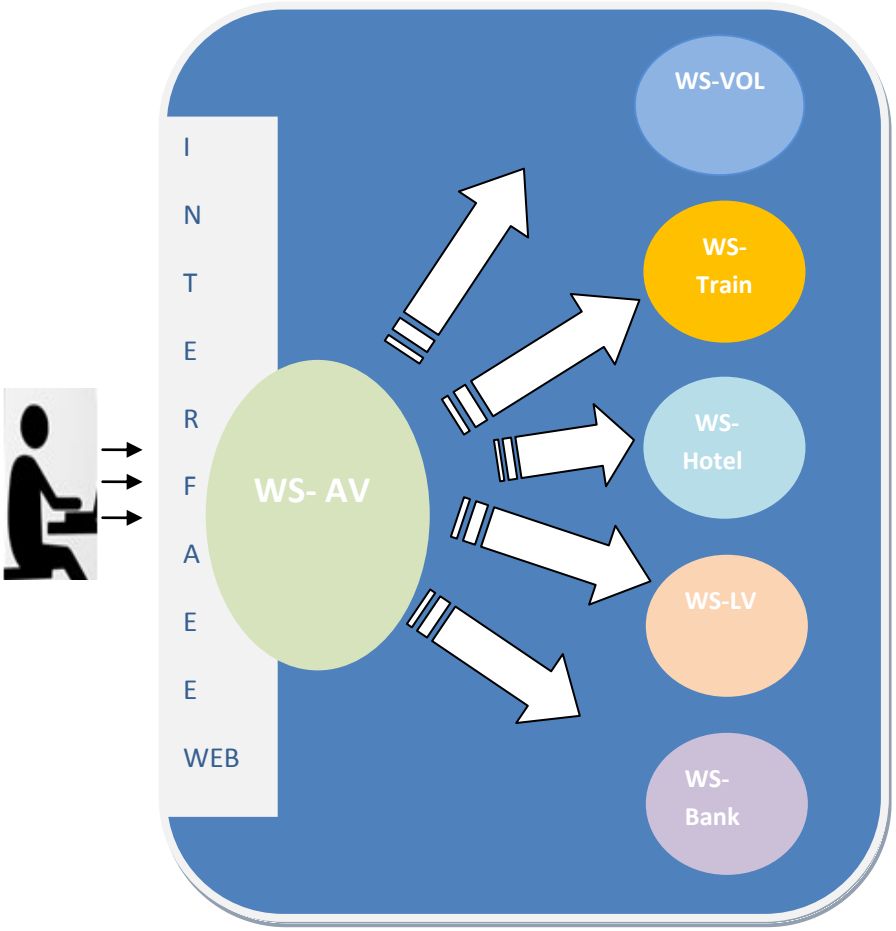

Figure V.1 Les services web d'une agence de voyage

Dans cet exemple le service (WS-AV) est la composition de cinq services (WS-VOL,WS-Train WS-Hôtel, WS-LV et WS-Bank). Donc le service WS-AV a comme rôle d'organiser les voyages par internet. Le service fournit aux clients une interface pour saisie les informations suivantes : Date de départ, date de retour, ville départ, ville arrivé et nombre de personnes et retournera : le prix total de voyage.et les billets.

Le scénario interne qui dirigera les interactions entre les services est le suivant : Le service WS-AV va d'abord connecter à la fois les services WS-Train et WS-VOL à travers ses méthodes respectivement *prixTrain()* et *prixVol()* afin de consulter le prix de billet d'avion est ce que <=*Montant* ou non Si oui Le service WS-AV appels la méthode *reserveVol()* de même service (WS-VOL).En suite et après assurer les réservations précédents, elle va connecter au service WS-Hôtel par la méthode *reserveChambre()* puis il va connecter au services *WS-LV* par la méthode *reserveVoiture().*En fin elle va envoyer le prix de chaque service au service WS-Bank pour calculer le prix total et payer par sa *méthode payer( ) . ou il va connecter avec les mêmes services par les mêmes méthodes si prixVol() > Montant mais le transport via le train (WS-Train) .*

V.2 Préparation de l'environnement de développement

Avant de commencer le développement de notre exemple, nous avons besoin d'installer des ressources logicielles suivantes:

Software and tools	Version Required
Java Development Kit (JDK) [Sun , 2015]	Version 5
NetBeans IDE [Net Beans , 2015]	Version 6.5
Glassfish Application server	(Avec NetBeans IDE)
B4free and Click'n'Prove [B4free, 2015].	Vx.x

Tableau IV.1: Outils de développement

Le serveur d'applications GlassFish doit être configuré correctement et être en exécution. Un projet de BPEL n'est pas directement déployer, nous devons ajouter un projet BPEL comme JBI (C'est Module pour le projet d'application. Ensuite, nous pouvons déployer le projet). Il définit un environnement pour les plug-in composants qui interagissent en utilisant un modèle de services reposant directement sur des Web Services Description Language (WSDL).

V.3 Composition des services de l'application (Processus BPEL4SW)

Pour développer un processus BPEL, nous passons par les étapes suivantes:

V.3.1 Définir les liens partenaires des services Web concernés (importation des fichiers WSDL externes)

Les cinq fournisseurs de services, Train, avion, hôtel, location de voiture et de la Banque des services Web ont été exposés sur le réseau et leurs fichiers WSDL peut être consultés par les URL comme suit:

Web Service	URL
Train	http://localhost:8080/TrainReservation/TrainReservationService?WSDL
Airline	http://localhost:8080/AirlineReservation/AilrlineReservationService?WSDL
Hotel	http://localhost:8080/HotelReservation/HotelReservationService?WSDL
Car rent	http://localhost:8080/CarrentReservation/CarrentReservationService?WSDL
Bank	http://localhost:8080/BankPayer/BankPayerService?WSDL

Tableau IV.2 Les URL WSDL des services Web

Netbeans outil permet d'importer les fichiers WSDL de fournisseur de service Web externe. Ainsi, nous importons les cinq fichiers WSDL pour les cinq fournisseurs de services avec leur schéma XML. WSDL du processus importe les WSDL des de services Web et définit les types Partnerlink pour eux.

Les six *partner link* sont les suivantes:

- ✓ ClientPartnerLink: utilisé pour décrire l'interaction entre le client et le processus BPEL lui-même. Cette interaction est synchrone. Ce type de lien partenaire est déclaré dans le WSDL du processus BPEL.
- ✓ AirlinePartnerLink: utilisé pour décrire l'interaction entre le fournisseur de service de la compagnie aérienne et le processus BPEL lui-même. Cette interaction est synchrone. Ce type de lien partenaire est défini dans le processus BPEL.
- ✓ TrainPartnerLink: utilisé pour décrire l'interaction entre le fournisseur de service de la compagnie de train et le processus BPEL lui-même. Cette interaction est synchrone. Ce type de lien partenaire est défini dans le processus BPEL.
- ✓ HotelPartnerLink: utilisé pour décrire l'interaction entre le processus BPEL et le service Web de Réservation d'Hôtel. Cette interaction est synchrone. Ce type de lien partenaire est défini dans le processus BPEL.
- ✓ CarPartnerLink: décrit l'interaction entre le processus BPEL et le loyer de voitures service Web. Cette interaction est synchrone. Ce type de lien partenaire est défini dans le processus BPEL.
- ✓ BankPartnerLink: décrit l'interaction entre le processus BPEL et le service Web de la Banque. Cette interaction est synchrone. Ce type de lien partenaire est défini dans le processus BPEL.

Les fonctionnalités de l'application sont décrites à l'aide des notations CTT. Une sémantique précise de ces notations a été définie dans le chapitre précédent (Figure IV.3 Grammaire de langage CTT).

V.3.1.2 Définir la logique du processus (des notations CTT):

La figure suivante (Figure V.2) présente le comportement des processus d'affaires, qui regroupe les différentes tâches qui doivent être atteints par la composition. Chaque réservation est liée à un service Web. Les services fonctionnent dans cet ordre par rapport aux conditions suivantes:

1- ***If*** (prix <Montant) ***Then*** reserveVol (WS-ALR) ***else*** reserveTrain(WS-TR).

2- reserveCHAM (WS-RH).

3- reserve voiture (WS-CR).

4- payer (WS-BP)

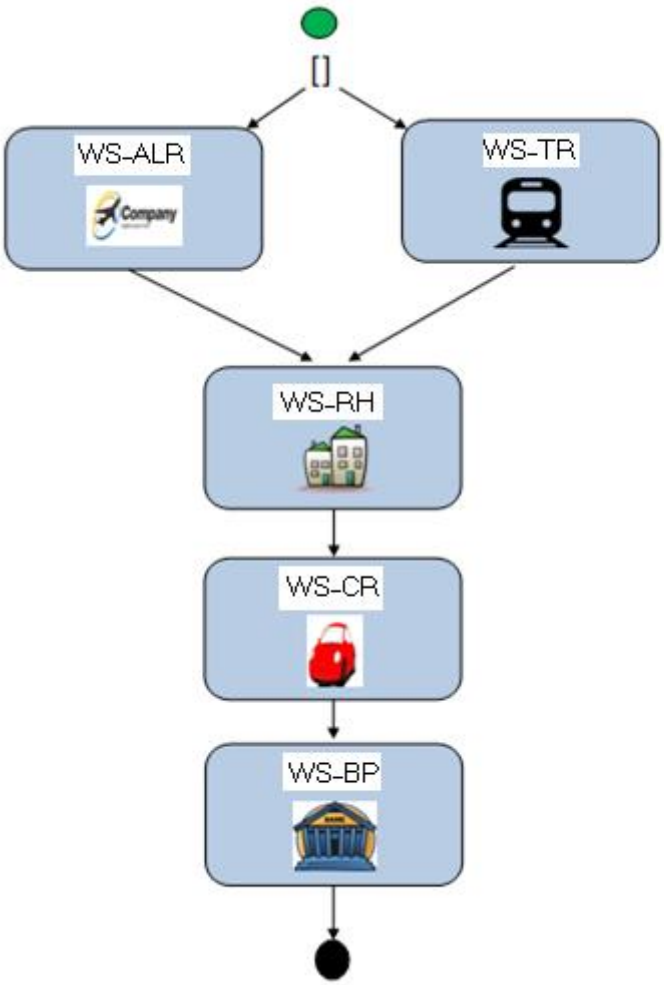

Figure V.2 La logique du processus de l'agence.

Nous fusionnons ce scénario pour produire le diagramme de séquence résultant (Sr) de cette composition, qui représente le comportement global du processus d'affaires.

Sr : if (Me :WS-ALR(PRICE<1500)) then WS-ALR else WS-TR; WS-HR ; WS-CR ; WS-BP

Sr-CTT: WS-ALR [] WS-TR >> WS-HR >> WS-CR >> WS-BP

Ces notations constituent l'entrée du processus de traduction en B. Celui-ci est décrit par un ensemble de règles formelles bien définies dans le chapitre précédent. Le développement d'une application se fera en codant l'expression des activités par des raffinements B en utilisant les schémas de raffinement des opérateurs.

V.3.1.3 La traduction en spécifications formelles B

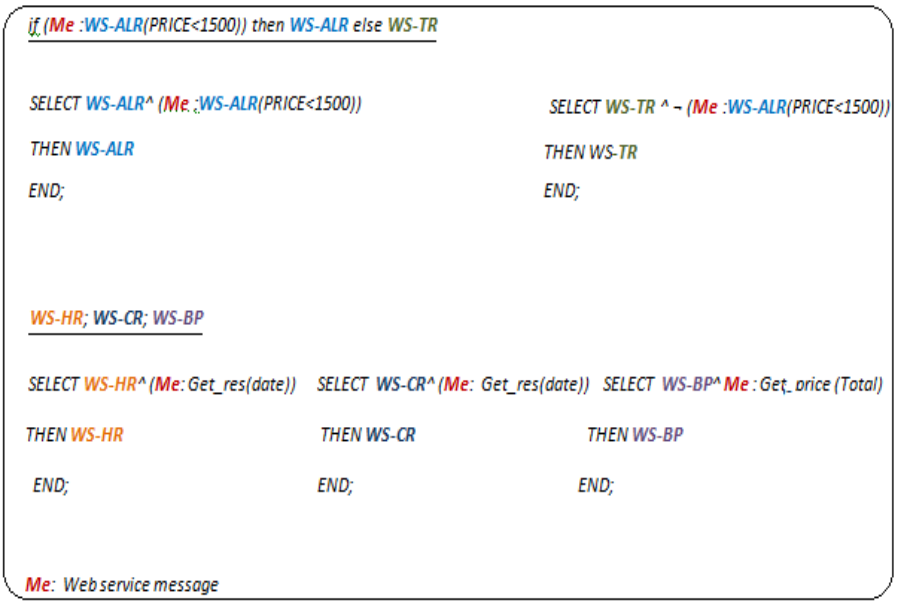

Figure V.3 La traduction en spécifications formelles B

.

```
MODEL Travel Agency
  REFINES WS-TA
 SETS CLIENT, Web Service
 VARIABLES client, name, destination, departuredate, returndate, WS-ALR ,WS-TR ,WS-HR ,WS-
 CR, WS- BP ,portType,   message
 INVARIANT
 client ⊆ Client∧ name ∧ destination ∈ client → STRING ∧
 departuredate∧returndate∈ client →NAT∧
 WS-ALR ∧WS-TR ∧WS-HR ∧WS-CR, WS-BP ⊆ Web Service∧ portType, message∈ Web Service
INITIALISATION
 client, name, destination, departuredate, returndate, WS-ALR ,WS-TR ,WS-HR ,WS-CR, WS-BP ,portType,
 message :=  Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø
EVENTS
 clientRequest :=
 SELECT WS-ALR^ (Me :=WS-ALR(PRICE<1500))
 THEN WS-ALR
 END;
 SELECT WS-TR ^ ¬ (Me :=WS-ALR(PRICE<1500))
 THEN WS-TR
 END;
 SELECT WS-HR^ (Me: = Get_res(date))
 THEN WS-HR
 END;
 SELECT WS-CR^ (Me: = Get_res(date))
 THEN WS-CR
 END;
 SELECT WS-BP^ Me := Get_price (Total)
 THEN WS-BP
 END;
 END
```

Figure V.4 Le modèle B de service web Agence de Voyage.

Les spécifications générées lors cette phase de traduction peut être réparties en plusieurs machines abstraites B reliées entre elles par la clause INCLUDES. L'objectif d'une telle modularisation est de réduire la taille et la complexité des machines, d'améliorer ainsi la lisibilité de la spécification et d'alléger la phase de preuve.

V.3.1. 4 La génération automatique de code source (BPEL4SW)

L'objectif de la phase de raffinement à produire des spécifications finales en B proches au langage de programmation cible choisie (BPEL4SW) telles que la dernière étape du codage devient plus intuitive et simple.

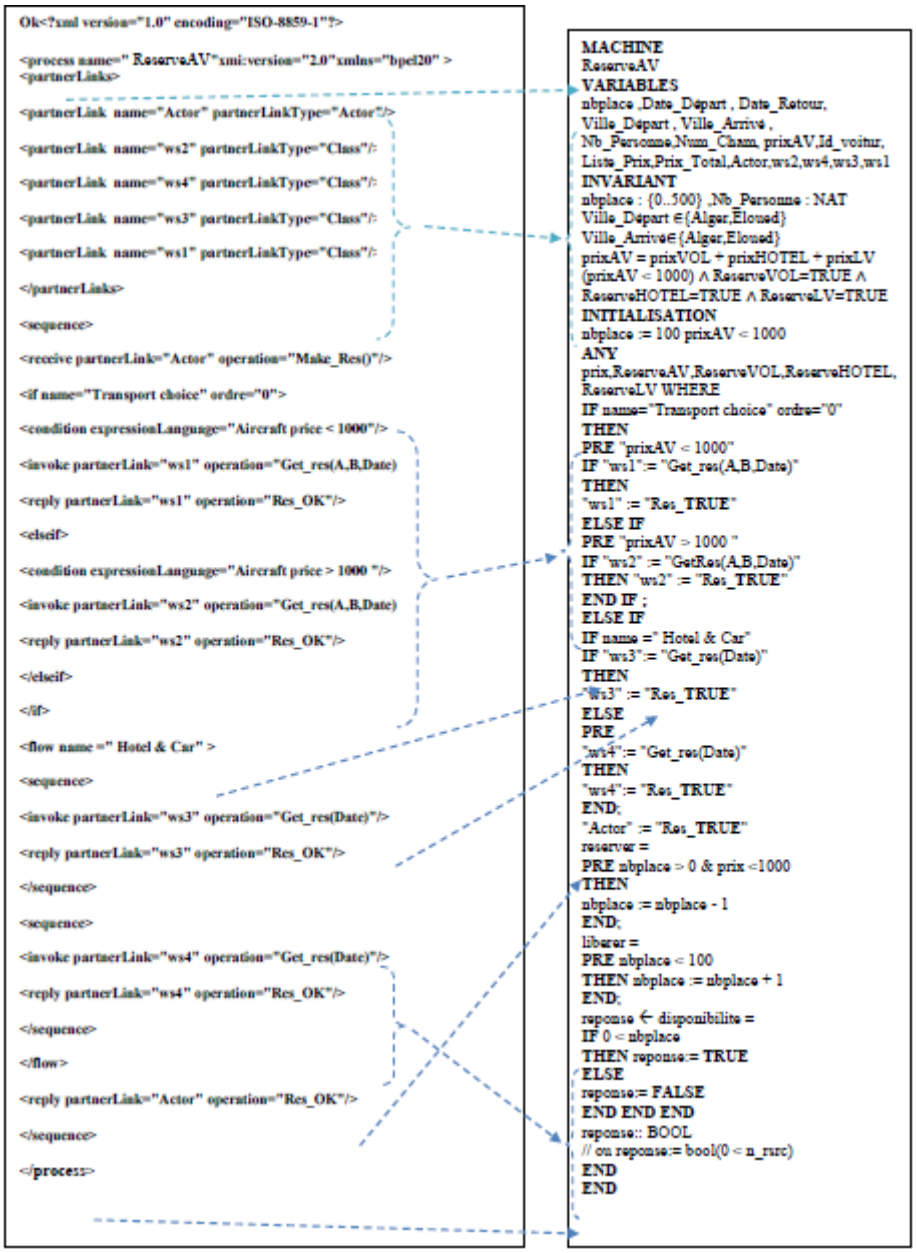

Figure V.5 Modèle B-BPEL4SW de service Web Agence de Voyage.

Les spécifications abstraites, engendrées à l'étape précédente, sont raffinées jusqu'à l'obtention du niveau d'abstraction (en langage B) correspondant à la formalisation des concepts du code source BPEL4SW. Pour cela un ensemble de règles génériques des instructions de base a été utilisées dans les règles de traduction (Tableau IV.3) :

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="TravelAgent"
         targetNamespace="http://enterprise.netbeans.org /bpel/TravelAgent/ TravelAgent_1"
         xmlns:ns1="http://localhost/TravelAgent/TravelAgent"
         xmlns:ns2="http://xml.netbeans.org/schema/TravelAgent">
    <import namespace="http://xml.netbeans.org/schema/TravelAgent"
            location="TravelAgent.xsd"
            importType="http://www.w3.org/2001/XMLSchema"/>
     <import namespace="http://localhost/TravelAgent/TravelAgent"
            location="TravelAgent.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://ws.airline/"
            location="localhost_8080/AirlineReservation/AilrlineReservationService.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://ws.hotel/"
            location="localhost_8080/HotelReservation/HotelReservationService.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://ws.car/"
            location="localhost_8080/CarReservation/CarReservationService.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
<partnerLinks>
     <partnerLink
          name="ClientPortEntry"
          partnerLinkType="ns1:partnerlinktypeforclient"
          myRole="partnerlinktyperole1">
</partnerLink>
 <partnerLink
       name="AirlinePartnerLink1"
      xmlns:tns="http://enterprise.netbeans.org/bpel/AilrlineReservationServiceWrapper"
       partnerLinkType="tns:AilrlineReservationLinkType"
       partnerRole="AilrlineReservationRole"/>
```

```
</partnerLinks>
 <variables>
      <variable name="RentCarOut" xmlns:tns="http://ws.car/"
                messageType="tns:rentCarResponse"/>
 </variables>
<sequence>
<receive partnerLink=“Client" operation=“Make_Res()"/>
<if name="Transport choice" ordre="0">
      <condition expressionLanguage="Aircraft price < 1500"/>
      <invoke partnerLink="WS-ARL" operation=“Get_res(nom, destination , departure date)"/>
      <reply partnerLink="WS-ARL" operation=“Res_OK"/>
<elseif>
      <condition expressionLanguage="Aircraft price > 1500 "/>
      <invoke partnerLink="WS-TR" operation=“Get_res(nom, destination , departure date)"/>
      <reply partnerLink="WS-TR" operation=“Res_OK"/>
</elseif>
</if>
```

Figure V.6 Code source BPEL4SW de service web Agence de Voyage.

V.4 Le comportement de l'interface de l’application agence de voyage

L’interface web de l’application (Agence de voyage) est une interface homme-machine permettant d'utiliser l’ application web par l’administrateur et les clients de l’agence. L’interface web graphique de l’application repose sur une gestion événementielle fine des actions de l’utilisateur. Cette finesse de gestion événementielle fait toute la richesse de l’application en environnement graphique. Cette interface contienne des composants (éléments de contact avec les services de partenaire : WS-VOL ,WS-Train, WS-Hotel ,WS-LV et WS-Bank) en Java , utilisent une gestion des évènements proche de « Swing » ; elles capturent ainsi les modifications et les soumissions de données.

Pour commencer à utiliser l'interface de l’application, nous créons une nouvelle application Java autonome en utilisant des outils NetBeans. Cette application contient une classe principale, où nous appelons le processus. En outre, les outils ajouter le package JAX-WS APIs qui est nécessaire pour compiler l'application.

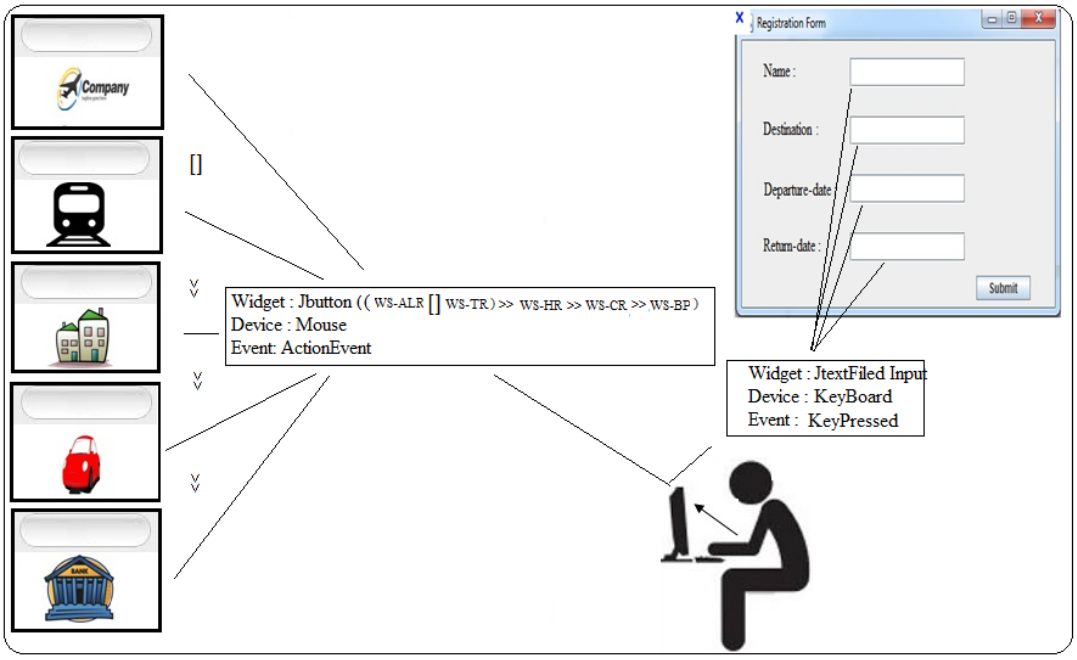

Figure V.7 L'interface de L'application web (Agence de voyage)

L'essentiel des traitements d'une application Web sont exécutés sur le serveur. L'important n'est pas ce qui est visible, mais la manière et le comportement.

Les taches des éléments (composant ergonomiques) de contacte avec les services web partenaires peuvent être formalisés comme suit :

$$WS\text{-}ALR \; [] \; WS\text{-}TR >> WS\text{-}HR >> WS\text{-}CR >> WS\text{-}BP$$

V.4.1 Modélisation formelle (B) de l'interface de l'application

MODEL WS-AT-SWING

SETS

/* Abstract Sets */

STRINGS={Text,No Text} ;

LISTENERS ; LISTENERS typ = {ActionListener, KeyListener,. . . } ;

EVENTS id = {ActionPerformed, KeyPressed, KeyReleased,. . .} ;

EVENTS typ ={ActionEvent, KeyEvent,. . .} ;

WIDGETS ;

VARIABLES

/* Sets to describe application Instances */

widgets, JButtons, JTextFields, . . ., listeners, . . .

/* Total Functions to describe Attributes of instances*/

w att, JB att, JTF att,l att,

/* Event Buffer : representation of JVM events */

User Action

INVARIANTS

/* Typing Predicates */

widgets ⊆ WIDGETS ^ JButtons ⊆ WIDGETS ^ JTextFields ⊆ WIDGETS ^ . . .

watt ∈ widgets ⟶ struct(visible :BOOL, enabled :BOOL, lists :P(listeners))^

latt ∈ listeners ⟶ struct(typs : P (LISTENERS typ))

User -Action ⟶ struct(typ :EVENTS typ, id :EVENTS id, source :widgets)

DECLARATIONS : :

Strings={no text, value} ;

JTextField input,output ;

JButton WS-ALR,WS-TR,WS-HR,WS-CR,WS-BP ;

Listener lis1 ;

INITIALIZATION : :

lis1.typs=(ActionListener, KeyListener) ;

input.visible = true ;

input.enabled = true ;

input.addKeyListener(lis1) ;

output.visible = false ;

output.enabled = false ;

WS-ALR.addActionListener(lis1) ;

WS-ALR.enabled = false ;

WS-TR.addActionListener(lis1) ;

WS-TR.enabled = false ;

WS-HR.addActionListener(lis1) ;

WS-HR.enabled = false ;

WS-CR.addActionListener(lis1) ;

WS-CR.enabled = false ;

WS-BP.addActionListener(lis1) ;

WS-BP.enabled = false ;

EVENTS

Evt ActionPerformed lis1 =

SELECT UserAction'source {WS-ALR, WS-TR}^

widgets att(UserAction'source)'enabled = TRUE^

widgets att(UserAction'source)'visible = TRUE^

UserAction'list=lis1 ^ UserAction'typ=ActionEvent ^

^ EVStart=1

THEN

Jtextfields_att(output)'text := Text || widgets_att(output)'visible := true ||

EVStart := EVStart-1

END;

Evt ActionPerformed list1-1 =

SELECT

EVStart= 0 ^ UserAction'source=WS-ALR

THEN

Widgets_att(WS-ALR)'enabled := true k widgets_att(WS-TR)'enabled := false

END;

Evt ActionPerformed list1-2 =

SELECT

EVStart= 0 ^ not(UserAction'source=WS-ALR)

THEN

Widgets_att(WS-ALR)'enabled := false || widgets_att(WS-ALR)'enabled := true

END

Evt ActionPerformed lis2 =

SELECT UserAction'source ∈ {WS-ALR, WS-RT}^

widgets att(UserAction'source)'enabled = TRUE^

widgets att(UserAction'source)'visible = TRUE^

UserAction'list=lis1 ^ UserAction'typ=ActionEvent ^

^ EVStart=1

THEN

Jtextfields_att(output)'text := Text || widgets_att(output)'visible := true ||

EVStart := EVStart-1

END;

Evt ActionPerformed list2-1 =

SELECT

EVStart= 0 ^ UserAction'source=WS-TR

THEN

Widgets_att(WS-TR)'enabled := true k widgets_att(WS-TR)'enabled := false

END;

Evt ActionPerformed list2-2 =

SELECT

EVStart= 0 ^ not(UserAction'source=WS-TR)

THENWidgets_att(WS-TR)'enabled := false || widgets_att(WS-TR)'enabled := true

END;

END

EXECUTION :

```
ActionPerformed(method of : lis1 ; react on : ActionEvent)
{
Start1 : output.text = value ;
         output.visible= true ;
         goto start1 ;
start11: if (evt1.source=WS-ALR)
          then start111
          else start112 ;
/*Inlining of invoked methods*/
start111 : WS-ALR.Enabled= false ;
            WS-ALR.Enabled= true ;
            goto end ;
start112 : EF.Enabled= true ;
FE.Enabled= false ;
goto end1 ;
end1 : return ;}
KeyPressed(method of : lis1 ;
react on : KeyEvent)
{...}
Start2 : output.text = value ;
         output.visible= true ;
         goto start1 ;
start21 : if (evt1.source=WS-TR)
          then start211
          else start212 ;
/*Inlining of invoked methods*/
Start211 : WS-TR.Enabled= false ;
            WS-TR.Enabled= true ;
            goto end ;
```

start212 : TR.Enabled= true ;

FE.Enabled= false ;

goto end2 ;

end2 : return ;}

KeyPressed(method of : lis2 ;

react on : KeyEvent)

{...}

END

V.4.2. La génération automatique de code source (JAVA) de l'interface de l'application

L'objectif de la phase de raffinement à produire des spécifications finales en B proches au langage de programmation cible choisie (JAVA) telles que la dernière étape du codage devient plus intuitive et simple. Les spécifications abstraites, engendrées à l'étape précédente, sont raffinées jusqu'à l'obtention du niveau d'abstraction (en langage B) correspondant à la formalisation des concepts du code source JAVA. Pour cela un ensemble de règles génériques des instructions de base a été défini dans les règles de traduction (Tableau IV.4).

Code source Java-Swing

```
public class ChoixTransport
extends JFrame
implements ActionListener, KeyListener
private JTextField input,output ;
JButton  WS-ALR,WS-TR ;
DC ChoixTransport dc ;
public P ChoixTransport()
{ Initialize() ;
subscribe() ;}
public void Initialize(){
/*Widgets creation*/
```

```
input=new JTextField() ;
output=new JTextField() ;
WS-ALR= new
JButton("AIRLINE RESERVATION") ;
WS-TR=new
JButton("TRAIN RESERVATION") ;
/* Widgets initialization */
        WS-ALR.setEnabled(false) ;
        WS-TR.setEnabled(false) ;
        output.setEnabled(false) ;
        this.setVisible(true) ;}
        /*listeners subscriexion*/
        public void subscribe() {
        input.addKeyListener(this) ;
        WS-ALR.addActionListener(this) ;
        WS-TR.addActionListener(this) ;} ;
        public void desactivate WS-ALR (boolean b){
        WS-ALR.setEnabled( !b) ;
        WS-TR.setEnabled(b) ;}
        public String getInput()
        {return input.getText() ;}
        public void setOutput(String s)
        {output.setText(s) ;}
        /* Listener methods */
        public void actionPerformed(ActionEvent e)
        {if (e.getSource()==WS-ALR)
        controler.WS-ALR(true) ;
        if (e.getSource()==WS-TR)
        {dc.WS-TR(false) ;}
```

```
public void keyPressed(KeyEvent e){}
public void keyReleased(KeyEvent e){}
public void keyTyped(KeyEvent e){} }
```

V.5 Outils B: B4free

B4free [B4free, 2015] est un ensemble d'outils pour le développement de modèles formels B, basé sur une version limitée de l'Atelier B et principalement destiné aux utilisateurs universitaires. Depuis 2009, l'Atelier B est disponible dans entièrement fonctionnel Community Edition. B4free ressources sont encore disponibles pour les enseignants / chercheurs qui les utilisent. Cependant la phase d'enregistrement est plus nécessaire.

B4Free permet d'utilisé la méthode B sur le plan opérationnel et offre de nombreuses fonctionnalités pour gérer les projets en langage B dans un environnement cohérent:

- ✓ la vérification syntaxique de modèles, génération automatique de théorèmes à démontrer,
- ✓ Démonstration automatiquement des théorèmes,
- ✓ Aide au développement: gestion automatique des dépendances entre les composants B, bibliothèques réutilisables, génération de documentation et métriques,
- ✓ Représentation graphique des projets, navigateur hypertexte à surfer dans un affichage projet, l'état du projet et des statistiques, la génération automatique d'un dictionnaire des termes du projet, l'archivage du projet.

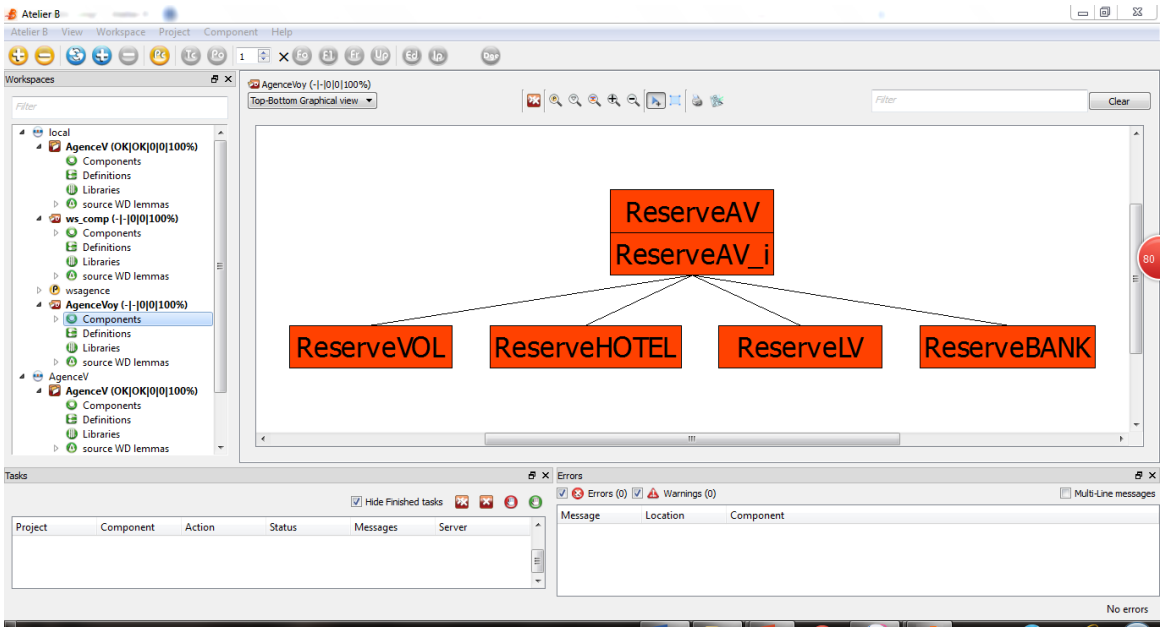

Figure V.8 Les composants B du modèle de service web Agence de Voyage (1).

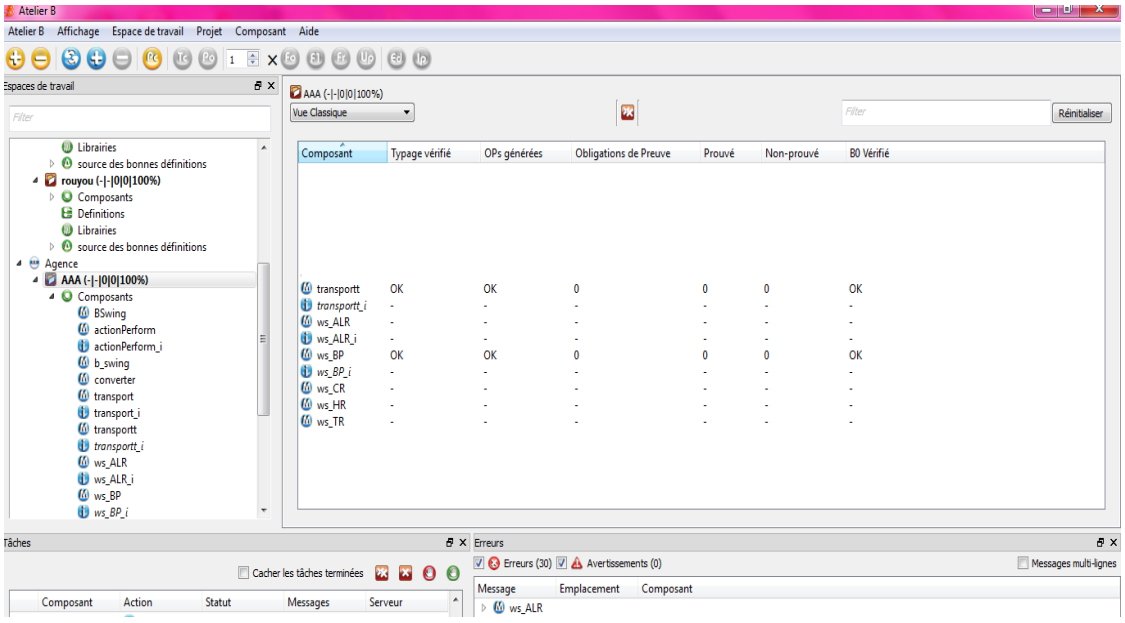

Figure V.9 Les composants B du modèle de service web Agence de Voyage (2).

V.1 Comparaisons :

Dans cette section nous proposons un modèle de comparaison entre les travaux et les approches de développements des applications web 2+. Le modèle proposé adopte les deux aspects des applications web 2+ (Architectural-coté serveur d'orchestration- et technologiques-coté clients), la formalisation et l'automatisation de processus de développement ; ainsi que la génération automatique des codes cibles (BPEL4SW et JAVA). La comparaison révèle l'importance de la contribution présentée dans cette thèse par rapport aux autres approches

Les approches	Aspect Architectural	Aspect Technologique	Ascendante	Descendante	La formalisation	L'automatisation	Code cible BPEL4SW	Code cible JAVA
Les systèmes de transitions LTSA-WS	√	×	×	√	√	×	√	×
Les réseaux de pétri	√	×	×	√	× (semi formelle)	×	√	×
L- algèbres de processus - ASDL	√	×	×	√	√	×	√	×
LOTOS/CADP	√	×	×	√	√	×	√	×
Le π -calcul	√	×	×	√	√	×	√	×
Les théories temporelles	√	×	×	√	√	×	√	×
Les logiques temporelles et La logique ACTL	√	×	×	√	√	×	√	×
L'approche Diapason	√	×	×	√	√	×	√	×
Le langage FIACRE	√	×	×	√	√	×	√	×
MDA-UML-S	√	×	×	√	× (semi formelle)	×	√	×

AAVF-IHM [Alexandre.Cor]	×	√	√	×	√	√	×	√
L'approche proposée	√	√	√	√	√	√	√	√

Tableau VI.1 Comparaison entre l'approche proposée et les autres approches de domaine

VI.2 Conclusions Générales et perspectives

Ce travail apporte une solution formelle pour le développement d'une application web sûre. En effet, beaucoup d'approches concernant la formalisation et la modélisation des services web existent dans la littérature, cependant toutes ces approches séparant entre l'aspect architectural (Orchestration des service web) et l'aspect technique (AJAX) lors de développement, et ce dernière est complètement ignoré lors de la formalisation des applications (services) web, ce qui peut réduire considérablement l'efficacité de formalisation des applications développées , , il n'y a aucune approche complet qui traite la développement des applications web tenant compte les des deux aspects architectural et technologique dans une approche unique.

Ainsi, ces approches restent des solutions partielles au problème de développement d'applications Web 2.0 (SOA) sûres. Les spécifications générées sont trop abstraites pour être directement supportées par un langage d'implémentation. Ces spécifications correspondent au niveau conceptuel du développement considéré. Une étape de raffinement (codage) de ces spécifications devient donc indispensable.

Notre approche est basée principalement sur la technique de raffinement B. Le processus de raffinement décrit dans ce travail est dédié à la génération d'une implémentation BPEL4SW et JAVA à partir d'une spécification semi-formelle CTT. Nous avons défini un méta modèle pour les concepts (B-Service Web-Classe JAVA) sa sémantique et ses différents fondements mathématiques. Ce modèle prend en compte les aspects comportementaux des applications Web 2+ .Ainsi nous avons présenté les différentes phases de l'architecture générale de l'approche proposée ainsi leurs concepts tels que le raffinement, la corrélation des messages, les règles de traduction. L'approche proposée c'est une approche formelle descendante/ ascendante pour la spécification, la validation, la vérification formelle et la génération automatique des codes (BPEL4SW, JAVA).

Pour un système logiciel trois stratégies complémentaires sont généralement reconnues et utilisées : Eviter les fautes (zéro-défaut), Tolérer les fautes et Détecter les fautes. Dans le cadre du développement logiciel, la méthode B répond à la première de ces stratégies (zéro-défaut) de façon très pertinente. En effet, en utilisant le langage B il est possible de prouver que le logiciel implanté est exempt de fautes par rapport à sa spécification. Aucune faute résiduelle ne peut donc rester dans le logiciel puisque la correction est prouvée. Bien sûr la spécification peut ne pas refléter exactement le besoin du client, c'est pourquoi le (zéro-défaut) ne signifie pas que le logiciel répond exactement au besoin du client.

L'approche présentée dans ce travail présente plusieurs avantages :

- ✓ Réduction du coût de développement : automatisation des phases de spécification, de raffinement, et de génération automatique de code
- ✓ Normalisation du code généré : les deux phases de traduction et de raffinement sont dictées par des règles précises et déterministes. Cette normalisation de code offrira une meilleure compréhension et maintenance du code ainsi produit.
- ✓ L'approche propose des métas modèles des structures (B-BPEL4SW-JAVA)
- ✓ L'approche proposée c'est une approche formelle descendante/ ascendante pour la spécification, la validation formelle et la génération automatique de code des orchestrations (BPEL4SW).

Nos travaux actuels portent sur :

- ✓ La réalisation d'un outil de raffinement automatique. Cet outil Viendra compléter cet outil, et qui permettant la génération automatique des documents WSDL associés aux services composés.
- ✓ La réalisation d'un outil de traduction automatique. Cet outil viendra compléter cet outil, permettant la traduction automatique des diagrammes UML et en spécifications B.

Ainsi ces outils, aura pour but d'assister le concepteur durant le processus de développement de ses applications web. Un tel outil déchargera le concepteur des différentes phases manuelles et surtout coûteuses du processus de développement.

Bibliographie

[Abrial, 1996] : Abrial.J.R : The B-Book, Combridge University Press, 1996.

[Alain SSII SQLI] : Alain Lefebvre, co-fondateur de la SSII SQLI et à l'origine de la création du service de réseaux sociaux synergies.

[Alexandre.Cor] : Alexandre Cortier : Une approche ascendante pour la vérification formelle des IHM (ONERA - Office National d'´etude et de Recherche Aérospatiales)

[A.Mammar] :Amel Mammar, (Une Approche Formelle par Raffinement pour le Développement d'Applications Bases de Données Sûres) , CEDRIC-IIE (CNAM)

[Andrews et al , 2003] : T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services version 1.1. http://www- 128.ibm.com/developerworks/library/specification/ws-bpel, 2003.

[Ankolenkar and al , 2001] : A. Ankolenkar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, SA. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. Daml-s : Semantic markup for web services. International Semantic Web Working Symposium (SWWS), 2001.

[Ankolenkar and al , 2002] : A. Ankolenkar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. Daml-s : Web service description for the semantic web. First International Semantic Web Conference (ISWC), 2002.

[Arkin et al 2002] : A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek. Web service choreography interface (wsci) 1.0. World Wide Web Consortium, http://www.w3.org/TR/wsci, 2002.

[Ass et Ali, 2007] : Le Web 2.0 pour la veille et la recherche d'information»Asselin. C et Alii, Ed. Digimind, Juin 2007, 113 p.

[Bartel et al , 2002] : M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. Xml-signature syntax and processing. World Wide Web Consortium, http://www.w3.org/TR/xmldsig-core/, 2002.

[Belhajjame et al 2001] : K. Belhajjame, G. Vargas-Solar, and C. Collet. A flexible workflow model for processoriented applications. 2nd International conference on Web Information Systems Engineering (WISE 2001), 2001.

[Belhajjame et al 2001] : K. Belhajjame, G. Vargas-Solar, and C. Collet. Defining and coordinating open-services using workflows. Eleventh International Conference on Cooperative Information Systems (COOPiS03), 2003.

[Booth et al] : D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture. World Wide Web Consortium, http://www.w3.org/TR/wsarch. - 178 -

[Bruno, 2009] Bruno Catteau - Nicolas Faugout,*Ajax - Le Guide complet*,Editions Micro Application - 2009,(ISBN 978-2-300-02202-9).

[Butler et Ferreira 2005] M. Butler, C. Ferreira, and M.Y. Ng. Precise Modelling of Compensating Business Transactions and its Application to BPEL. j-jucs, 11(5) :712–743, 2005.

[CADP] INRIA. CADP : Construction and Analysis of Distributed Processes. Software Tools for Designing Reliable Protocols and Systems. http://www.inrialpes.fr/vasy/cadp.html.

[Cau et al 2006] : A. Cau, B. Moszkowski, and H. Zedan. Interval temporal logic. http://www.cse.dmu.ac.uk/STRL/ITL/, 2006.

[Chanliau , 2006] : M. Chanliau. Web services security : What's required to secure a service-oriented architecture. Oracle White Paper http://www.oracle.com/technology/tech/standards/pdf/security.pdf, 2006.

[Chinnici 2007] : R. Chinnici, J.J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (wsdl) version 2.0. World Wide Web Consortium, http://www.w3.org/TR/wsdl20, 2007.

[Chiriello et Salaun 2005] : A. Chirichiello and G. Salaun. Encoding abstract descriptions into executable web services : Towards a formal development negotiation among web services using lotos/cadp. IEEE/WIC/ACM International Conference on Web Intelligence (WI 2005), 2005.

[Christensen 2001] : E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. World Wide Web Consortium, http://www.w3.org/TR/wsdl/, 2001.

[CLEA, ATELIER B] Document é tabli par CLEARSY. ATELIER B Langage B Manuel Utilisateur : version 1.2 CLEARSY Maintenance ATELIER B Europarc de PICHAURY 1330 Av. J.R. Guilibert Gauthier de la Lauzi˙ere - Bât C2 13856 Aix-en-Provence Cedex 3 France.

[Clements 1999] : L. Bass, P. Clements, and R. Kazman. Software architecture in practice. Addison Wesley, ISBN 0-201-19930-0, 1999.

[Clement et al , 2004] : L. Clement, A. Hately, C. von Riegen, and T. Rogers. Uddi version 3.0.2. http://uddi.org/pubs/uddi v3.htm, 2004.

[Collet 2006] : P. Collet. Etat de l'art sur la contractualisation et la composition. RNTL FAROS - Livrable F-1.1 – http://www2.lifl.fr/faros/pub/uploads/Main/RNTL-FAROS-F1-1.pdf, 2006.

[Cox et al] : W. Cox, F. Cabrera, G. Copeland, T. Freund, J. Klein, T. Storey, and S. Thatte. Web services transaction (ws-transaction).

[Curbera et al , 2003] : T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services version 1.1. http://www- 128.ibm.com/developerworks/library/specification/ws-bpel, 2003.

[Della-Libera et al] : G. Della-Libera, M. Gudgin, P. Hallam-Baker, M. Hondo, H. Granqvist, C. Kaler, H. Maruyama, M. McIntosh, A. Nadalin, N. Nagaratnam, R. Philpott, H. Prafullchandra, J. John Shewchuk, D. Walter, and R. Zolfonoon. Web services security policy language (ws-securitypolicy).

[Domi , Fil] : Dominique FILIPPONE, JDN Solutions (PageWeb : Ajax, technologie au cœur du Web 2.0)

[Emilia , Gregorio 2007] M. Emilia Cambronero, Gregorio Diaz, J. Jose Pardo, and Valentin Valero. Using UML Diagrams to Model Real-Time Web Services. In Proceedings of the Second International Conference on Internet and Web Applications and Services (ICIW '07), page 24, 2007.

[Fallside and Walmsley, 2004] : D. Fallside and P. Walmsley. Extensible markup language schema (xml schema) 1.0.World Wide Web Consortium, http://www.w3.org/TR/xmlschema-0, 2004.

[Farail.Gaufillet 2008] P. Farail, P. Gaufillet, F. Peres, J.-P. Bodeveix, M. Filali, B. Berthomieu,S. Rodrigo, F. Vernadat, H. Garavel, and F. Lang.FIACRE : an intermediate language for model verification in the TOPCASED environment. In European Congress on Embedded Real-Time Software (ERTS), 2008.

[Fay, 2010] Fayçal ABOUZAID Thèse : ANALYSE FORMELLE D'ORCHESTRATIONS DE SERVICES WEB , DÉCEMBRE 2010

[Ford, 2001] : W. Ford, P. Hallam-Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein, and J. Lapp. Xml key management specification (xkms). World Wide Web Consortium, http://www.w3.org/TR/xkms/, 2001.

[Foster et al 2003]: H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. IEEE Automated Software Engineering (ASE), 2003.

[Foster et al 2005]: H. Foster, S. Uchitel, J. Magee, and J. Kramer. Tool support for model-based engineering of web service compositions. IEEE International Conference on Web Services (ICWS), 2005.

[Foster et al 2006]: - H. Foster, S. Uchitel, J. Magee, and J. Kramer. Ltsa-ws : A tool for model-based verification of web service compositions and choreography. IEEE International Conference on Software Engineering (ICSE 2006), 2006.

[Frédéric ; 2007] : Frédéric POURRAZ These : Diapason : une approche formelle et centrée architecture pour la composition évolutive de services Web - le 10 décembre 2007.

[Graham 2004] : S. Graham, D. Davis, S. Simeonov, G. Daniels, P. Brittenham, Y. Nakamura, P. Fremantle, D. Koenig, and C. Zentner. Building web services with java : Making sense of xml, soap, wsdl, and uddi, 2nd edition. 2004.

[Gudgin et al , 2003] : M. Gudgin, M. Hadley, N. Mendelsohn, J.J. Moreau, and H.F. Nielsen. Simple object access protocol (soap) 1.2. World Wide Web Consortium, http ://www.w3.org/TR/soap, 2003.

[Hamadi et Benatallah 2003] : R. Hamadi and B. Benatallah. A petri net-based model for web service composition. Fourteenth Australasian Database Conference (ADC2003), 2003.

[H. Mounier] H. Mounier Support de cours Java : Structures de données Notions en Génie Logiciel et Programmation Orientée Objet Université Paris Sud.

[Henzinger.Manna 1991] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In REX Workshop, pages 226–251, 1991.

[IBM, 2003] IBM. BPEL4WS (version 1.1), May 2003. http://www.ibm.com/developerworks/library/ ws-bpel.

[Imamura et al , 2002] : T. Imamura, B. Dillaway, and E. Simon. Xml-encryption syntax and processing. World Wide Web Consortium, http://www.w3.org/TR/xmlenc-core/, 2002.

[Imamura et al , 2005] : T. Imamura, M. Tatsubori, Y. Nakamura, and C. Giblin. Web services security configuration in a service-oriented architecture. 14th international conference on World Wide Web, 2005.

[ISO/IEC , 1989] : ISO/IEC. Lotos : A formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, International Organization for Standardization - Information Processing Systems - Open Systems Interconnection, 1989.

[Jean.M 2002] : Jean-Michel DOUDOUX. : Livre(Développons en Java v 2.00 -décembre 2002)

[Jesse, 2006] : Jesse James Garrett (l'auteur du livre très largement référencé The Elements of User Experience) *Article original:* Ajax, a New Approach to Web Applications.

[KangChan et al] :L. KangChan, J. JongHong, L. WonSeok, J. Seong-Ho, and P. Sang-Won. Qos for web services : Requirements and possible approaches.

[Kayantzas 2005] : N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. World Wide Web Consortium, http://www.w3.org/TR/ws-cdl-10, 2005.

[Koshutanski and Massacci, 2003] : H. Koshutanski and F. Massacci. An access control framework for business processes for web services. ACM workshop on XML security (XMLSEC 2003), 2003.

[Krishnaswamy et Loke] : S. Kalepu, S. Krishnaswamy and SW. Loke. Verity : A qos metric for selecting web services and providers.

[Kowalski et Sergot 1986]: R. Kowalski and M.J. Sergot. A logic-based calculus of events. New generation computing 4(1), pages 67-95, 1986.

[Krishnaswamy et al a] S. Krishnaswamy, SW. Loke, and A. Zaslavsky. Application run time estimation : A qos metric for web-based data mining service providers. a.

[Krishnaswamy et al b] - S. Krishnaswamy, SW. Loke, and A. Zaslavsky. Efficient prediction of quality of service for data mining web services. b.

[LALLALI, 2009] LALLALI M Mounir ,Thèse : Modélisation et Test Fonctionnel de l'Orchestration de Services Web présentée pour l'obtention du grade de Docteur de l'INSTITUT NATIONAL DES TELECOMMUNICATIONS Soutenue le 20 Novembre 2009

[Liu et al] : Y. Liu, A. Ngu, and L. Zheng. Qos computation and policing in dynamic web service selection.

[LOTOS 1989] : ISO. LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, 1989.

[Luc Van., 2007] : Luc Van Lancker, AJAX — Développez pour le Web 2.0 : Entrez dans le code : JavaScript, XML, DOM, XMLHttpRequest…, Editions ENI - 2007 (ISBN 978-2-7460-3707-6).

[Manna and Pnuel , 1990] : Z. Manna and A. Pnueli. A hierarchy of temporal properties. Proceedings of the 9th ACM Symposium on Principles of Distributed Computing (PODC'90), 1990.

[Manna and Pnuel , 1992] : Z. Manna and A. Pnueli. The temporal logic of reactive and concurrent systems. Volume I : Specification. Springer-Verlag, 1992.

[Martin et al , 2004] : D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing semantics to web services : The owl-s approach. First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), 2004b.

[Mateescu , 2003] : R. Mateescu. Logiques temporelles basées sur actions pour la vérification des systèmes asynchrones. Projet VASY - Rapport de recherche 5032, 2003.

[Mateescu , 1998] : R. Mateescu. Vérification des propriétés temporelles des programmes parallèles. Thèse -Institut National Polytechnique de Grenoble, 1998.

[McDermott , 2002] : D. McDermott. Estimated-regression planning for interaction with web services. Sixth International Conference on AI Planning and Scheduling, 2002.

[MethodeB , 2015] GLEARSY SYSTEM ENGINEE RING : http://www.methode-b.com/outils/.

[Meftah et Kazar, 2015] : Meftah Mohammed charaf eddine et Pr.Kazar Okba .Article "Web Applications Development by Formal Refinement Approach" publié chez : International Journal of Software Engineering and Its Applications Vol. 9, No. 12 (2015), pp. 73-98 http://dx.doi.org/10.14257/ijseia.2015.9.12.07

[Meftah,Kazar et Hani. 2016] : Meftah Mohammed charaf eddine , Pr.Kazar Okba et Hani Nabil. Application AutoIt pour une transformation des codes (CTT-B-BPEL4SW-JAVA).2016.

[Michael., 2006] : Michael Mahemoff, Ajax design patterns, O'Reilly Media, Inc. - 2006 (ISBN 978-0-596-10180-0).

[McIlraith and Son , 2002] : S. McIlraith and T. Son. Adapting golog for composition of semantic web services. Eighth International Conference on Knowledge Representation and Reasoning (KR2002), 2002.

[Milner , 1989] : R. Milner. Communication and concurrency. Prentice Hall, 1989.

[Milner , 1999] : R. Milner. Communicating and mobile systems : the _-calculus. Cambrige University Press, 1999.

[Mon, 2005] : Le Monde Informatique, no 1139 .

[Myerson ; 2002] : JM. Myerson. Web services architectures. http://www.webservicesarchitect.com/content/articles/webservicesarchitectures.pdf, 2002.

[Myerson ; 2003] : S. Mysore. Securing web services - concepts, standards, and requirements. Sun White Paper - http://www.sun.com/software/whitepapers/webservices/securing webservices.pdf, 2003.

[Nadalim et al, 2004] : A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo. Web services security : Soap message security 1.0 (ws-security 2004).

[Nakamura et al, 2005] : Y. Nakamura, M. Tatsubori, T. Imamura, and K. Ono. Model-driven security based on a web services security architecture. IEEE International Conference on Services Computing (SCC 2005), 2005.

[Nathaniel,2006] : Nathaniel T. Schutta - Ryan Asleson,*Pro Ajax and Java Frameworks*,Apress, 2006,(ISBN 978-1-59059-677-7)

[Nau et al , 2003] : D. Nau, T.C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. Shop2 : An htn planning system. Artificial Intelligence Research Journal, 2003.

[Nicola and Vaandrager, 2010] : Christophe Dumez Thèse Doctorat : Approche dirigée par les modèles pour la spécification, la vérification formelle et la mise en œuvre de services Web composés (MDA-UML-S) de l'Université de Technologie de Belfort-Montbéliard (2010))

[Nicola and F.W. Vaandrager, 1990] : R. De Nicola and F.W. Vaandrager. Action versus state based logics for transition systems. In Semantics of Concurrency, volume 469 of Lecture Notes in Computer Science - Springer Verlag, 1990.

[Oasis, 2007] OASIS Standard. WSBPEL Ver. 2.0, April 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[OMG 2008] Object Management Group. UML, Unified Modeling Language, version 2.1.2, 2008. http: //www.omg.org/technology/documents/formal/uml.htm.

[Orchard et al 2004] : D. Orchard, F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, J. Shewchuk, and T. Storey. Web services coordination (ws-coordination). http://dev2dev.bea.com/pub/a/2004/03/ws-coordination.html, 2004.

[Patr Chass] : (Patrick Chassany, fondateur d'Amen et co-fondateur de plusieurs sociétés et services en ligne Web 2.0, dont Fotolia et EveryFeed).

[Papazoglou 2003a] : M. Papazoglou. Web services and business transactions. Technical Report 6, Infolab, Tilburg University, Netherlands, 2003a.

[Papazoglou 2003b] : M.P. Papazoglou. Service-oriented computing : Concepts characteristics and directions.

In I. CS, editor, (WISE-03), 2003b.

[Papazoglou 2004] : MP. Papazoglou. Web services : Concepts, architectures, and applications. In Springer

Verlag, 2004.

[Papazoglou 2006] : MP. Papazoglou and WJ. van den Heuvel. Service-oriented design and development methodology.

Int. J. of Web Engineering and Technology (IJWET), 2006. et -M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and Kr¨amer B. J. Service-oriented computing : A research roadmap. Service Oriented Computing (SOC), number 05462, 2006.

[Peltz 2003] : C. Peltz. Web services orchestration : A review of emerging technologies, tools, and standards. http://devresource.hp.com/drc/technical white papers/WSOrch/WSOrchestration.pdf, 2003.

[Perrin 2005] : S. Bhiri, C. Godart, and O. Perrin. Reliable web services composition using a transactional approach. e-Technology, e-Commerce and e-Service (EEE 2005), 2005.

[Pourraz et Verjus 2007b] : F. Pourraz and H. Verjus. Diapason : An engineering environment for designing, executing and evolving service-oriented architectures. Second International Conference on Software Engineering Advances (ICSEA 2007), 2007b.

[Pourraz et Verjus 2007] : H. Verjus and F. Pourraz. A formal framework for building, checking and evolving service oriented architectures. 5th IEEE European Conference on Web Services (ECOWS 2007), 2007.

[Paterno, 2000,2003] Paterno, F. (2000). Model-Based Design and Evaluation of Interactive Applications. Applied Computing Series, Springer. Paterno, F. (2003). ConcurTaskTrees: An Engineered Notation for Task Models. Chapter 24, in Diaper, D., Stanton, N. (Eds.), The Handbook of Task Analysis for Human-Computer Interaction, Lawrence Erlbaum Associates, 483-503.

[Radu 2009] Radu Mateescu and Sylvain Rampacek. Formal Modeling and Discrete-Time Analysis of BPEL Web Services. Int. J. of Simulation and Process Modelling, 4 :183–194, 2009.

[RmdiScala 2006]Rm di Scala : Premier pas dans .Net - Programmation événementielle et visuelle (rév. 28.08.2006).

[Rouchard et Godart , 2006] : M. Rouached and C. Godart. Securing web service compositions : Formalizing authorization policies using event calculus. 4th International Conference on Service-Oriented Computing (ICSOC 2006), 2006.

[Rouchard et al , 2006a] : M. Rouached, W. Gaaloul, W.M.P. van der Aalst, S. Bhiri, and C. Godart. Web service mining and verification of properties : An approach based on event calculus. OTM Confederated International Conferences, 2006a.

[Rouchard et al , 2006b] : M. Rouached, P. Perrin, and C. Godart. Towards formal verification of web service composition. 4th International Conference on Business Process Management (BPM 2006), 2006b.

[Salaun et al , 2004] : G. Salaun, A. Ferrara, and A. Chirichiello. Negotiation among web services using lotos/ cadp. European Conference on Web Services (ECOWS 04), 2004.

[Solanki et al 2006] : M. Solanki, A. Cau, and H. Zedan. Asdl : A wide spectrum language for designing web services. 15th International World Wide Web Conference (WWW2006), 2006.

[Srivastava and J. Koehler , 2003] : B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. International Conference on Automated Planning and Scheduling (ICAPS 2003), 2003.

[Skogsrud et al , 2007] : H. Skogsrud, B. Benatallah, F. Casati, and F. Toumani. Managing impacts of security protocol changes in service-oriented applications. 29th International Conference on Software Engineering (ICSE 2007), 2007.

[Tao et al] : Y. Tao, Z. Yue, and L. Kwei-Jay. Modeling and measuring privacy risks in qos web services.

[Tim, 2005] Tim O'Reilly (2005). What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software . Web 2.0 Conférence 2005. 30 septembre 2005.

[Tim et Batte 2005] Dans l'exposé d'ouverture de leur conférence (Web 2.0 Conférence 2005). 30 septembre 2005.

[Wohed et al , 2002] : P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-based analysis of bpel4ws. QUT Technical report, FIT-TR-2002-04.

[Wohed et al , 2003] : P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of web services composition languages : The case of bpel4ws. 22nd International Conference on Conceptual Modeling (ER 2003), 2003.

[Wu et al , 2003] : D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating daml-s web services composition using shop2. Second International Semantic Web Conference (ISWC2003), 2003.

[Wass, 1994] :Wasserman et Faust, auteurs de Social Network Analysis: Methods and Applications publié en 1994

[W3C., 2002] : W3C Note 8 Web Service Choreography Interface (WSCI) 1.0 (August 2002)

[W3C., 2005] : Web Services Choreography Description Language Version 1.0 W3C Candidate Recommendation 9 November 2005

[Yuan,Zhong 2006]: Yuan Yuan, Zhongjie Li, and Wei Sun. A Graph-Search Based Approach to BPEL4WS Test Generation. In Proceedings of the International Conference on Software Engineering Advances (ICSEA '06), pages 14–14, Oct. 2006.

Sites Web

[B4free, 2015] : http://www.b4free.com/public/installLinuxSun.php.

[Net Beans , 2015] : NetBeans. NetBeans Tool. http://www.netbeans.org.

[Sun , 2015] : Sun. Java SE Downloads-JDK 5. http://java.sun.com/javase/downloads/index_jdk5.jsp.