SOMMAIRE

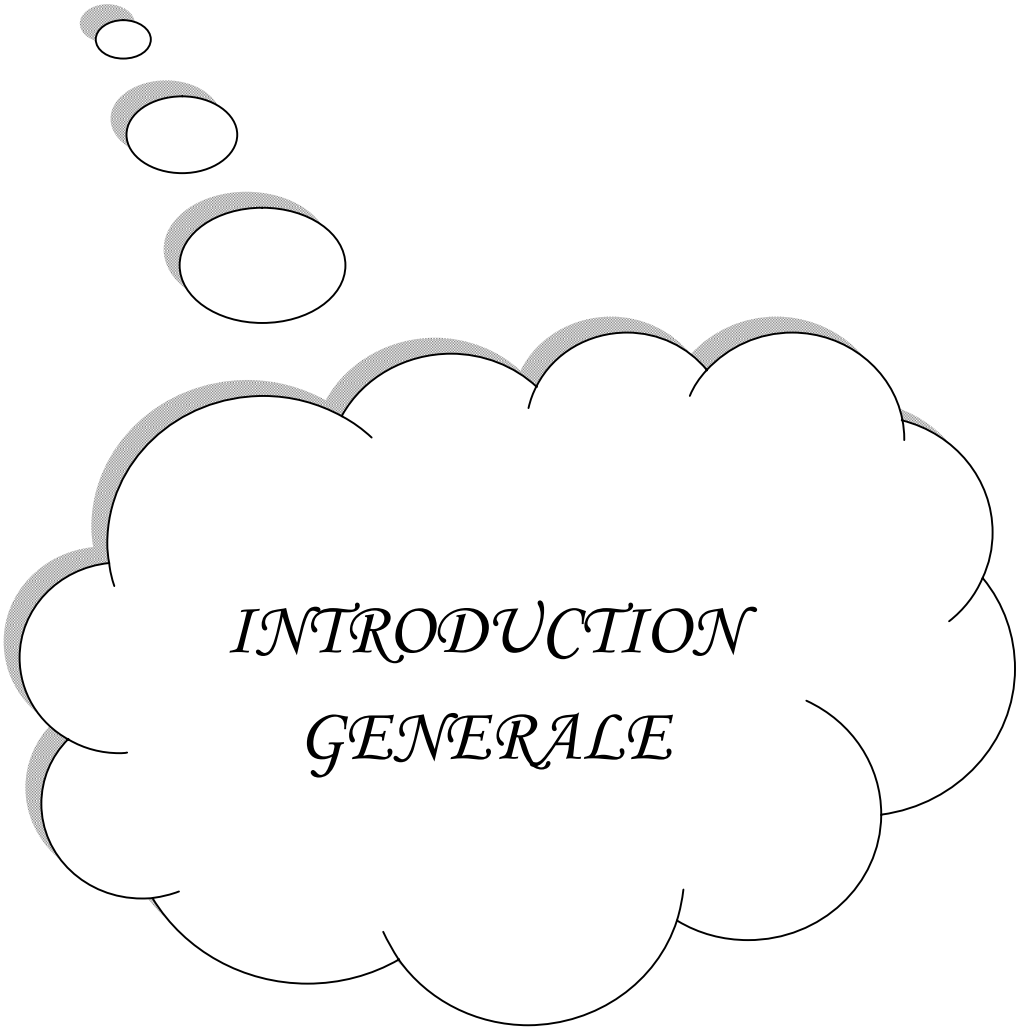
CHAPITRE 3 : LES SYSTEMES MULTI_AGENTS

CHAPITRE 4 : LA CONCEPTION DU MODELE DE DEVELOPPEMENT

CHAPITRE 5 : ETUDE DE CAS

CHAPITRE 6: CONCLUSION GENERALE

Chapitre 1



1. CONTEXTE GENERAL

Dernièrement, l'importance de l'Internet s'augmente de façon remarquable. Les applications basées Web s'évoluent de plus en plus. On s'oriente actuellement vers des contenus basés services où l'Internet, au lieu d'être une simple source d'information souvent difficile à gérer, devient un acteur dans le processus de réalisation des objectifs d'une application donnée. Avec l'apparition des nouveaux standards et technologies, qui ont changé l'infrastructure de l'Internet de son architecture classique client-serveur à l'architecture services Web, plusieurs tendances de recherche sur les bénéfices qui peuvent être gagnés de l'utilisation de l'Internet sont émergées. Ils ont adopté, pour réduire le coût et le temps de réalisation et de maintenance des applications Web, l'approche basée composant pour le développement et le déploiement de ces applications.

Le Développement Basé Composants (DBC) [14] est l'une des techniques la plus utilisée à l'heure actuelle en génie logicielle. Il facilite la construction des applications à grande échelle supportant la composition de modules simples et élémentaires dans des applications complexes. Cette approche présente notamment les avantages de mieux séparer l'interface et l'implémentation et de rendre l'architecture des applications explicite. Ainsi, un composant logiciel ne montre que ses interfaces requises et fournies ; et des contrats basiques qui s'établissent lors des assemblages de composants.

La transposition de l'approche de développement basée composant dans le cadre du Web donne la naissance d'un nouveau paradigme appelé les services Web. Un service Web est un composant logiciel qui offre des services à travers une interface standardisée. La particularité des services Web réside dans le fait qu'elle utilise la technologie Internet comme infrastructure pour la communication entre les composants logiciels (les) et ceci en mettant en place un cadre de travail basé sur un ensemble de standards. Le processus de standardisation touche trois couches du modèle de fonctionnement global des services Web : un protocole de communication permettant de structurer les messages échangés entre les composants logiciels, une spécification de description des interfaces des services et enfin une spécification de publication et de localisation des services. Ce modèle supporte principalement des composants logiciels qui présentent des services sous forme d'une collection d'unités de traitement (opérations) dont l'invocation n'excède pas un échange simple de messages (généralement requête-réponse). A ce niveau de complexité des composants logiciels, le

paradigme des services Web est suffisant pour mettre en place des composants interopérables et facilement intégrables.

On remarque aujourd'hui un rapprochement entre les recherches en SMA et les services Web. Ce rapprochement se manifeste sous différents aspects. On en distingue essentiellement deux : d'une part l'utilisation des agents en tant que cadre conceptuel pour mettre en place des services Web sophistiqués, ou encore des outils de planification et composition de services Web. D'autre part l'utilisation des services Web comme un cadre architectural et/ou technologique pour mettre en place des systèmes multi-agents ou des agents communicant à travers Internet.

2. PROBLEMATIQUE

Le Web a été établi comme une plateforme majeure pour les différentes applications, bien que les modèles d'implémentations sous-tendant compliquent le développement et l'évolution de ces applications Web. L'augmentation de la complexité des applications Web fait émerger une approche disciplinée pour le développement et l'évolution mandataire.

Puisque le développement des applications Web est étroitement lié à des multiples disciplines ayant une relation avec les technologies de l'informatique et les réseaux, il peut être assez complexe, coûteux et lent, s'il ne supporte pas une méthodologie pratique et efficace [9]. En réalité, le développement des applications Web est complexe parce elles requièrent une maintenance régulière afin de mettre à jour le contenu des pages, de suivre des groupes de travail particulièrement de commerce, d'introduire des nouvelles caractéristiques pour supporter des nouvelles tâches ou/et de nouveaux clients etc.

En plus, le développement Web a subi des problèmes de plusieurs origines. Parmi ces problèmes, le développement des applications Web est souvent fait par plusieurs personnes qui suivent des manières ad hoc dans leur processus de développement. Un autre problème influant le développement des applications Web est le temps de leur livraison et le temps de leur mise à jour qui peut être quelques heures de travail. D'un autre côté, les concepteurs sont obligés de changer leurs conceptions presque tout le temps pour satisfaire les besoins variables et instables des utilisateurs ce qui rend le développement d'un projet Web a un haut niveau d'évolution dans sa nature [92].

Pour s'occuper de la complexité de développement Web, le soutien par la modélisation est essentiel puisque elle fournit une vue abstraite de l'application. Les méthodes et les modèles traditionnels sont employés avec succès dans le développement des systèmes hypertexte mais ils ont montrés leurs limites dans la conception des applications Web vue leurs caractéristiques spécifiques. Même si les applications Web sont des parties des logiciels, qui est le cas dans la plus part du temps, le processus de développement utilisé est en commun de plus avec les magasines de publications qu'avec les logiciels de développement conventionnels. Cependant, les méthodes de développement des applications Web requièrent, de plus en plus, un certain type d'intégration avec les anciennes méthodes de développement. Il devient clair que les méthodes appropriées pour le développement des applications Web doivent combiner la conception traditionnelle des logiciels et les particularités de la conception des applications Web.

Dans les années récentes, quelques modèles de développement des hypermédias et les systèmes d'informations pour le Web ont été utilisés pour le développement des applications Web [92]. On peut citer, par exemple, les méthodes OOHDM, UWE, WebML, JESSICA et d'autres. Les méthodes influées par les systèmes hypertextes et les méthodologies orientées objet proposent des solutions fournissant des modèles abstraits décrivant l'ensemble des classes des éléments d'informations et les structures navigationnelles sans donner l'importance aux détails d'implémentation. Les méthodes basées sur les approches incluant les constructions de haut niveau peuvent augmenter la productivité. Néanmoins, ces méthodes ne peuvent pas être adaptées pour prendre en compte un aspect très important qui est la nature serviable des applications Web. En plus, elles ont plusieurs problèmes dont les plus significatifs sont :

- Le manque de séparation entre la modélisation et l'implémentation.
- L'inadaptation de ces méthodes pour répondre aux besoins des concepteurs.
- Le manque de la détection des besoins ambiguës et leur résolution.
- L'augmentation rapide de complexité des conceptions obtenues avec les tailles des applications Web.
- Le manque d'une manière structurelle et fiable qui assure le bon passage de la conception à l'implémentation.
- La lenteur des durées de développement et la difficulté de maintenance.

- Le manque de réutilisation où les concepteurs réalisent souvent des maquettes jetables et non utilisables dans les applications finales.
- Certaines méthodes imposent un type particulier d'implémentation qui limite la liberté des concepteurs et leur demande le savoir des connaissances spécifiques.
- Ces méthodes sont soit orientées données soit orientées présentation et aucune méthode ne prend en compte l'aspect service des applications Web.

3. OBJECTIF DU PROJET

Dans le but d'augmenter et de faciliter la productivité de développement des applications Web, de minimiser le temps de développement et de faciliter la maintenance, de prendre en compte l'aspect serviable des applications Web (des applications qui ne font que rendre des services à leurs utilisateurs) et de rendre ces applications maintenable, nous allons proposer un modèle pour le développement des applications Web.

L'objectif de ce travail est de présenter un modèle générique, général et extensible pour le développement, le déploiement, l'évolution et le soutien des applications Web. Ce modèle doit construire des applications Web évolutives et maintenables par le regroupement d'un ensemble de services Web. Dans ce modèle, nous avons basé sur la réutilisation des services et on a permis l'extensibilité par la possibilité d'ajout des nouveaux services. Dans notre modèle, les agents sont les acteurs qui vont utiliser les services Web et les ontologies pour réaliser ces objectifs. Chaque agent a un rôle et une tâche spécifique à réaliser. Les agents collaborent, interagissent et inter-opèrent pour assurer le fonctionnement global du système.

4. ORGANISATION DU MEMOIRE

Ce mémoire est composé de cinq chapitres organisés de la manière suivante :

Dans le deuxième chapitre, nous présentons l'approche de développement à base de composant ; à savoir son émergence, en focalisant sur ses concepts principaux de base tel que la notion de composant. Ensuite, nous penchons vers l'application de cette approche pour le développement des applications Web pour obtenir enfin l'ingénierie basée composant pour le développement Web.

Dans le troisième chapitre, nous décrivons les systèmes multi-agents tout en discutant les différents aspects des agents tel que l'interaction, la communication etc. Puis, nous discutons

les possibilités de la composition entre l'approche muti-agents, l'approche basée composant et les services Web.

Dans le quatrième chapitre, nous allons définir et détailler un modèle pour le développement, le déploiement, l'évolution et le soutien des applications Web. Ce modèle est basé sur la composition des services Web pour générer des applications Web finales. Nous commençons par la présentation des différents composants tels que les ontologies, les services Web sémantique et les systèmes multi-agents. Puis, nous présentons l'architecture de notre modèle toute en illustrant ses éléments.

Dans le cinquième chapitre, nous poursuivons avec une étude de cas afin de valider notre proposition. On a choisi comme cas à étudier l'organisation des voyages car elle est considérée comme un exemple typique pour la composition des services Web.

Nous terminons par une conclusion générale sur notre travail et les perspectives envisagées.

Chapitre 2

1. INTRODUCTION

À l'heure actuelle, les applications sont de complexité croissante. Leur réalisation nécessite des investissements de plus en plus importants avec la prise en compte le délai de mise sur le marché qui est l'un des facteurs déterminant de succès. Pour réduire le temps de développement des applications, la composition et la réutilisation représentent des thèmes de recherche importants pour les organisations de développement.

- La composition est une manière de structurer et de construire des applications à partir des entités logicielles. Avec la composition, la profession d'informaticien s'évolue de plus en plus depuis le statut de « programmeur » d'applications vers le statut d'« intégrateur » d'éléments logiciels.
- La réutilisation est la construction des nouvelles applications à partir des éléments logiciels existants. La réutilisation permet de profiter des fonctionnalités déjà écrites, déboguées et maintenues par les autres.

Il y a de nombreux travaux qui cherchent à trouver des nouveaux et efficaces paradigmes de développement des logiciels avec un coût raisonnable tout en introduisant des contraintes de réutilisation de code, d'administration, de maintenance et d'évolution d'applications. La programmation modulaire puis la programmation orientée objet sont apparus pour résoudre les problèmes de complexité et réduire le temps de développement [07].

Au cours des dernières décennies, la programmation orientée objet (POO) s'est montrée comme un paradigme puissant pour la réalisation d'applications évolutives et à grande échelle. Elle fournit les abstractions nécessaires à la représentation modulaire des éléments du problème modélisé. Selon le principe d'encapsulation, chaque élément apparaît sous la forme d'un objet possédant des données (représentées par les attributs) et des opérations accessibles à travers une interface (les méthodes). Ce principe garantit la "bonne manipulation" des objets. Les objets ayant des caractéristiques et des comportements communs sont des instances d'une même classe. Ainsi, il s'agit d'un modèle de conception permettant la réutilisation. Néanmoins, la programmation orientée objet a démontré plusieurs limites [01, 03] : la structure de l'application est généralement peu visible, la difficulté de la modification (ajout/suppression) de fonctionnalités et la construction de l'application est prise totalement en

charge par le programmeur (construction des différents modules, définition des instances et interconnexion des modules),… etc.

Une solution prometteuse aujourd'hui c'est l'approche de développement du logiciel à base de composant. L'apparition de cette approche est venue pour faire face aux défauts de l'approche objet [12]. Cette approche prouve son efficacité dans le génie logiciel car elle vérifiée les conditions de construction des bons logiciels tels que la réutilisabilité, l'indépendance, l'extensibilité, etc.

Cette approche est vue comme une technologie adopté pour :

- développer des logiciels complexes avec succès (plus de réutilisation du code),
- augmenter la productivité de développement qui est le facteur le plus important

Avec l'expérience positive du développement basé composant des logiciels, il est souhaitable d'adapter l'utilisation de cette approche pour le développement et l'évolution des applications Web [19, 20]. Le résultat de la combinaison de l'approche de développent basé composant et le Web est concrétisé dans une nouvelle approche appelée l'ingénierie de Web basé composant « Component Based Web Engineering ».

Dans ce chapitre, nous allons présenter l'approche de développement à base de composant en donnant une définition à la notion du composant et son architecture. Puis, on citera les différentes catégories des composants et une classification des modèles à base de composant. Ensuite, on présentera l'approche de services Web qui est un modèle basé composant adopté pour donner une nouvelle architecture à l'Internet. Enfin, nous allons citer deux modèles de développement des applications Web basés composant avec une définition de leurs concepts et une présentation de leurs modèles.

2. LA REUTILISATION

La réutilisation de la conception et du code est un clé de succès et une tâche primordiale du génie logiciel [24, 25, 26]. La notion de réutilisation n'est pas un nouveau concept. Depuis le début de la programmation, on cherche à réutiliser ce qui a été déjà fait afin de le combiner et de l'intégrer. Les méthodes de réutilisation s'évoluent et se simplifient. En effet, au départ,

le seul moyen de réutilisation du code était de le recopier dans un autre programme. Puis, les premières bibliothèques de fonctions sont apparues suivies des bibliothèques de classes. Mais, depuis 1995, l'émergence des composants logiciels permet une réutilisation plus simple et plus efficace grâce aux interfaces de composant et des protocoles de réutilisation.

Nous allons citer quelques exemples pratiques montrant comment la réutilisation facilite le développement durant le cycle de vie des applications :

- La réutilisation des applications par le soutien des outils d'aide de différents systèmes ou machines.
- La réutilisation des fonctionnalités fournies par les bibliothèques standard comme les bibliothèques mathématiques.
- La réutilisation des logiciels à base de composant suivi par Java Haricots ou DCOM/COM composants.
- La réutilisation de modélisation de connaissance comme les *patterns* et les *frames*.

Cependant, ces pratiques sont moins familières dans le développement des applications Web. En plus, et encore plus mauvais, la maintenance de grandes applications Web est souvent effectuée par des ingénieurs des sites au lieu d'auteurs authentiques du contenu d'application. Souvent, ces ingénieurs finissent inconsciemment par reconstruire les concepts de haut niveau de la conception originale à partir de l'implémentation elle-même pour rendre la conception plus faisable et réalisable selon leur point de vu. Certainement, le résultat de cette erreur de stratégie est la perte de l'intégrité des applications et l'aboutissement à des systèmes contradictoires.

En résumé, il y a deux manques principaux qui nécessitent l'intervention par étude de la part du génie logiciel, en utilisant ses différentes techniques, pour le développement des applications Web :

- La réutilisation de la conception ou du code pour le développement des applications Web est insatisfaisante.
- Le manque d'utilisation des concepts d'héritage de l'orienté objet dans le développement des applications Web [27].

3. DEFINITIONS

L'approche à base de composants est une méthodologie qui soutient l'idée que la construction de logiciels peut être réalisée à partir de l'assemblage d'entités logicielles autonomes réutilisables appelées composants [13] pour définir une architecture globale du logiciel. Elle permet d'assimiler à terme le développement du logiciel à un simple processus d'assemblage. Un composant définit de manière explicite un ensemble de provisions et de dépendances pour permettre de réaliser sa composition [02,47].

Suivant cette approche, une application logicielle est construite en interconnectant plusieurs composants. Chacun d'entre eux représente un aspect fonctionnel de l'application (un ensemble de services) [12]. Le cycle de vie et le modèle de génie logiciel de l'approche à base de composant sont différents de l'approche traditionnelle car l'approche à base de composant suit le principe d'assemblage des composants pour construire une application finale. Cette approche a connu un intérêt dans les communautés de recherche et les industries de logiciel [10].

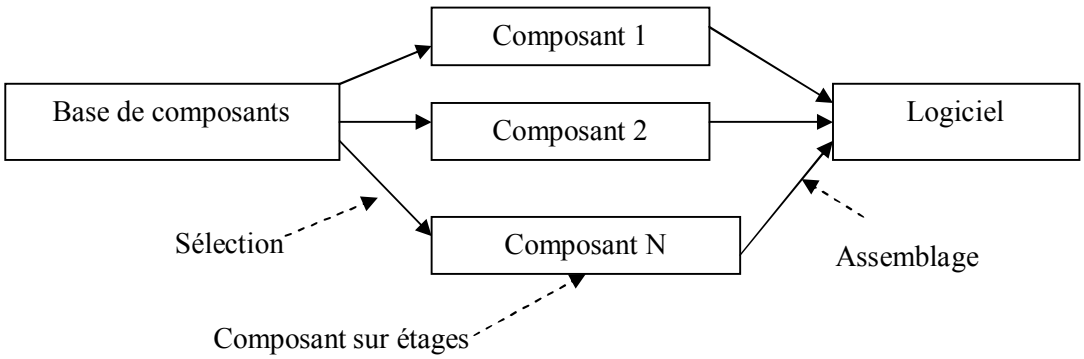


Figure 1. Développement de logiciel à base de composant [10].

Pour mettre en œuvre une technologie de composants, on doit définir un modèle de composants, une infrastructure à composants, une implémentation de référence ainsi que de la documentation et des outils de développement.

Le modèle de composants définit les entités ainsi que leur mode d'interaction et de composition. Il doit fournir les propriétés suivantes :

- l'encapsulation : séparer l'interface de son implémentation, utiliser l'interface comme seul point d'entrée des fonctionnalités d'un composant, pouvoir définir des interfaces multiples qui reflètent les divers comportements du composant ;
- la composabilité : expliciter les ressources requises et fournies pour stablir la composition;
- la capacité à décrire l'architecture de manière plus ou moins formelle notamment par l'utilisation de langages de description de composants ;
- la réutilisabilité et l'évolution : définir des modèles génériques de composants, pouvoir adapter les composants par des interfaces de contrôle, pouvoir les reconfigurer.

L'infrastructure de composants met en œuvre le modèle de composants et permet de construire, déployer, administrer et exécuter des applications conformes à ce modèle. Elle couvre et gère le cycle de vie des composants en permettant leur installation et leur activation, la réaction aux événements, la collecte des données et la configuration ainsi que la gestion des propriétés non fonctionnelles (e.g la persistance, la sécurité, les transactions) [11].

4. LES BENEFICES DE DEVELOPPEMENT A BASE DE COMPOSANT

La programmation à base de composant a pour but d'améliorer la réutilisabilité, la sûreté, la flexibilité et la productivité des applications, etc [14]. Pour cela, elle se base sur la notion de composant et par là même la notion de composition ou d'assemblage. Elle propose de concevoir une application comme un assemblage de briques logicielles préfabriquées.

D'après Councill et Heineman dans [15], le but de l'ingénierie logicielle basée composant (CBSE) est de *"créer des assemblages précis de composants bien documentés, de qualité et dignes de confiance"*. Ainsi, les auteurs soulignent le fait que ce type de conception s'intéresse non seulement aux phases de développement des composants, mais aussi à leur phase de conception, de modélisation (basée sur des outils de conception, outils visuels, UML, Model Driven Architecture de l'OMG) et d'assemblage (basée la plupart du temps sur des langages de description ADL). Ce qui facilite une fiable réutilisation de composants. Certains modèles permettent même le réassemblage, pendant l'exécution, des composants, ce qui permet d'adapter les applications [03].

Donc les bénéfices attendus sont multiples [04] :

- Diminuer considérablement les coûts et le temps de conception en généralisant la réutilisation en donnant la possibilité aux entreprises de développement d'achat et de vente de composants déjà réalisés.
- Améliorer totalement la qualité du système.
- Augmenter la fiabilité des logiciels par l'utilisation de composants largement testés.
- Faciliter et réduire les coûts de maintenance et l'évolution des systèmes par le remplacement des composants causant des problèmes.
- Enfin, rentabiliser les développements par la vente de COTS Component (Commercial-off-the-shelf Component ou composants sur étagère).

5. NOTION DE COMPOSANT

Le développement de logiciels à base de composants est une approche qui vise à rendre disponibles des composants logiciels sur étagères. L'objectif, comme cela peut être le cas en électronique, est de les réutiliser afin de concevoir des applications logicielles par composition. Mais c'est quoi, en réalité, un composant logiciel ?

D'un point de vue ingénierie informatique, la notion de composant n'est pas encore totalement bien définie. Il existe plusieurs définitions du composant où chacune focalise sur un aspect élémentaire des aspects décrivant ces composants. Elles ont tendance à converger vers une solution unique mais ce n'est pas encore aussi bien placée que la notion d'objet [08].

Une définition du concept du composant logiciel dans [05] est donnée comme suit *: Les composants logiciels se présentent comme des boîtes exécutables réutilisables.*

Suivant [13], *un composant logiciel est une entité autonome de déploiement qui encapsule du code informatique. Cette entité est décrite par des interfaces définissant les interactions autorisées avec d'autres composants.*

Cependant, suivant Jed harris président du CI Lab [06, 16] : *Un composant logiciel est un morceau de logiciel assez petit pour que l'on puisse le créer et le maintenir et assez grand*

pour que l'on puisse l'installer et en assurer le support. De plus, il est doté d'interface standard pour pouvoir interopérer.

En d'autres termes, les composants logiciels sont des entités logicielles autonomes qui présentent clairement les services qu'elles offrent et ceux qu'elles requièrent chez d'autres composants pour fonctionner [12]. Ainsi, un composant peut-être utilisé facilement sans qu'un développeur n'ait besoin d'en connaître son fonctionnement (en allant regarder son code). Les services offerts par les composants doivent être génériques afin de pouvoir être réutilisés simplement.

Il existe une autre définition qui prend sa source dans l'approche orientée objet. Elle définit un composant logiciel comme:

- Une entité commercialisable : c'est un logiciel autonome prêt à l'emploi, qu'il est possible d'acheter dans le commerce;
- Ce n'est pas une application complète : il doit être combiné à d'autres composants pour former une application complète (il dispose d'un nombre limité de fonctionnalités dans un domaine d'application particulier);
- Il peut être utilisé en agencements non prévus à l'avance : comme les objets, il peut être utilisé dans des combinaisons que le développeur n'avait pas prévues à l'avance;
- Il possède une interface définie au travers de laquelle il peut être manipulé.
- C'est objet qui sait interopérer : un composant peut être appelé comme objet au travers d'espaces d'adresse, de réseaux, de langages, de systèmes d'exploitation, et d'outils variés. C'est une entité logicielle indépendante des systèmes;
- Un objet au sens où il supporte l'encapsulation, l'héritage et le polymorphisme.

Une autre définition d'un composant présentée dans [06], où il y a la traduction de la définition donnée par les participants à la première édition du Workshop sur la programmation orienté composant : *Un composant logiciel est une unité de composition spécifiant, par contrats, ses interfaces (fournies et requises) et ses dépendances explicites aux contextes. Un composant logiciel peut être déployé indépendamment et peut être sujet de composition par un tiers pour la conception d'applications logicielles.*

Il en résulte de cette définition que :

- Un composant est une unité de composition spécifiant, par contrat, ses interfaces (fournies et requises) et ses dépendances explicites aux contextes ;
- Un composant logiciel peut être déployé indépendamment de plate-formes (l'installation sur différentes plates-formes) ;
- Un composant est capable de s'auto-décrire par son interface, ce qui permet aux constructeurs d'applications de l'utiliser facilement sans qu'ils aient besoin d'en connaître son fonctionnement.

6. ARCHITECTURE D'UN COMPOSANT

L'architecture typique d'un composant logiciel figure 2 [06] spécifie ses entrées et ses sorties afin de faciliter la description de son comportement (services offerts) quels que soient les langages de programmation utilisés.

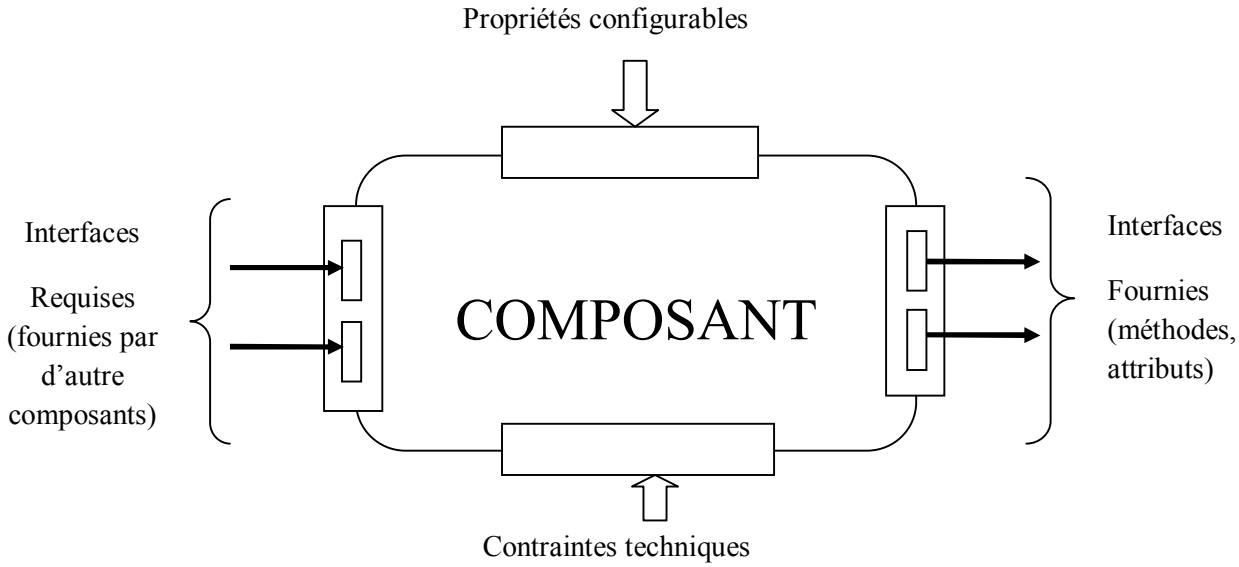


Figure. 2. Architecture d'un composant logiciel [06].

Comme le présente la figure 2, un composant logiciel possède, principalement, les cinq éléments suivants :

- Les interfaces fournies (sorties) et les interfaces requises (entrées), en mode synchrone ou asynchrone : ce qui définit ses moyens mis en œuvre pour coopérer. Ces moyens peuvent être des opérations (des fonctions promises aux clients) ou des propriétés ;
- Les propriétés configurables : généralement, ce sont des attributs. Elles permettent d'adapter et de personnaliser le composant dans des contextes d'exécutions spécifiques ;
- Les contraintes techniques (QoS : Quality of Services), qui peuvent être : la sécurité, la persistance, les transactions, etc.
- le corps de composant : contient les détailles de fonctionnalité décrit dans l'interface.

7. LES DIFFERENTES CATEGORIES DE COMPOSANTS

On peut distinguer les composants logiciels suivant deux plans : conceptuel et fonctionnel selon [07, 08]. D'un point de vue conceptuel, On distingue, principalement, les composants boîte noire et les composants boîte blanche. Les composants boîte blanche sont fournis en code source et sont donc modifiables avant l'intégration. Cependant, les composants boîte noire, à l'inverse, sont des codes compilés ou binaires, leurs fonctionnalités ne sont accessibles qu'au moyen de leurs interfaces publiques et ils ne peuvent être modifiés directement. Pour étendre ou redéfinir certaines de leurs fonctionnalités, il est nécessaire d'utiliser des adaptateurs. Néanmoins, une solution intermédiaire entre les deux précédentes consiste à considérer des composants boîte grise. Dans ce cas, un sous-ensemble des modalités de fonctionnement d'un composant serait ouvert à la modification au moyen d'un ensemble de mécanismes prédéfinis.

Suivant un point de vue fonctionnel, on peut distinguer deux catégories de composants : les composants métiers et les composants non métier ou système.

La classification des modèles à composants est dictée par l'architecture trois tiers des systèmes d'information. Ainsi on distingue couramment deux grands types de modèles à composants : les modèles à composants de présentation (ex : JavaBeans) et les modèles à composants métiers (ex : EJB, CCM, .NET), la partie donnée étant gérée par les services techniques des composants métiers. Cependant, un troisième type de composants est

récemment apparu : les modèles à composants qu'on pourrait dénommer "génériques" (ex : Fractal, ArcticBeans, Avalon) [03].

Les composants métiers sont développés dans le cadre d'une profession ou d'une entreprise et satisfont des besoins liés aux activités d'un utilisateur. Cependant, les composants non-métiers sont des composants développés pour résoudre un problème correspondant à un besoin rencontré par un ensemble d'applications de métiers différents (composants de communication, de sécurité, de persistance transactionnelle, …).

8. LES ENVIRONNEMENTS A COMPOSANTS

Il existe plusieurs environnements à composants. On va présenter quelques uns tels que COM+, CCM et EJB. Puis, nous présentons une comparaison entre ces environnements.

8.1. LE MODELE DE COMPOSANTS COM+

Au début des années 90, le modèle COM (Component Object Model) s'est imposé comme une technologie Windows, importante et évolutive [09]. Le but est de rendre, tout objet COM capable d'être distribué. En d'autre terme, il peut être instancié à partir d'une autre machine via un simple appel réseau. On parle alors de DCOM (Distribueted COM), qui est l'extension de COM prenant en compte l'aspect distribué (il permet la communication entre objets situés sur des machines différentes).

L'architecture Internet proposée par Microsoft, pour développer et supporter les applications réparties, construite autour de DCOM s'appelle DNA (Distribueted Network Application). Sous Windows 2000, DCOM change de nom pour devenir COM+. On peut dire alors que COM+ est l'évolution de COM. En suite, Microsoft a développé un environnement de création d'application Web nommé « .NET ».

Le modèle COM+ inclut une série de fonctionnalités comme le "data binding " assurant la liaison entre certains objets et les champs d'une base de données, des fonctionnalités d'événements et des messages asynchrones (COM+ Event Services et Queued Components), et la gestion de la distribution, etc.

Un composant COM+ peut avoir une ou plusieurs Interfaces de l'Objet qui dérivent de l'interface IUnknown. Les interfaces sont fortement typées. L'interface IUnknown est l'interface de base que doit implémenter tout composant COM+. C'est par cette interface que l'on peut instancier un composant COM+. Elle offre trois méthodes : QueryInterface qui permet aux clients de découvrir les interfaces fournies (supportées par le composant COM+), et ainsi, de naviguer entre elles. La classe Factory fournit l'interface IClassFactory qui contient deux méthodes : la méthode CreateInstance a pour but d'instancier cette classe pour obtenir des Objets qui soient distants ou locaux. La méthode LockServer permet de mettre l'Objet en mémoire afin de l'instancier au plus vite.

8.2. LES COMPOSANTS "JAVA BEANS" ET "ENTERPRISE JAVA BEANS" (EJB)

Avec le succès remarquable du langage Java, Sun MicroSystems a standardisé, au sein de J2EE, la technologie EJB qui consiste en deux partis : JavaBeans pour le développement de composants côte client et Enterprise JavaBeans (EJB) pour le développement côté serveur [10]. La technologie EJB définit un modèle abstrait de composants Java côté serveur qui décrit la structure des composants EJB et des modèles de développement et de déploiement des applications à base de composants. Dans le modèle EJB, les composants sont des composants Java non visuels, portables, distribués, réutilisables et déploieables [06].

Ce modèle EJB propose un ensemble de bibliothèques (classes) Java pour fabriquer des applications distribuées à base de petits serveurs logiciels. Les spécifications EJB fournissent un cadre et des règles pour construire ce type d'applications. La version actuelle des spécifications est "2.1". La version 3 est encours de définition [08].

Selon le modèle EJB, une application répartie est constituée de composants appelés Beans qui sont implémentés en Java. Les Beans n'ont qu'une seule interface d'entrée et peuvent être de trois types : session, entité et à messages. Au sein des applications, les Beans de type session et entité sont accessibles à travers une communication synchrone basée sur Java RMI. Les Beans à messages, introduits par la dernière version de la spécification, sont un moyen d'intégration d'un service d'événements dans les EJB. Ils communiquent de manière asynchrone en utilisant JMS.

L'établissement d'un lien vers un Bean consiste en la récupération de sa référence. Elle peut être passée en paramètre par un autre composant ou obtenue à l'aide d'un service de nommage.

Les instances de Beans s'exécutent dans des conteneurs EJB qui sont chargés de leurs aspects non fonctionnels. Les conteneurs peuvent héberger plusieurs instances de plusieurs types de Bean. Ils gèrent les transactions, la sécurité et la persistance selon trois types de configurations correspondant aux trois types de composants (session, entité et à message). Les conteneurs sont inclus dans des environnements d'exécution, des serveurs EJB, qui fournissent les services système nécessaires [09].

8.3. LES COMPOSANTS "CORBA COMPONENT MODEL"

Le modèle "Corba Component Model" a vu ses premières spécifications apparaître en 1998, a été finalisé par l'OMG en 2002 et est basé sur la norme CORBA. Le middleware CORBA (Common Object Request Broker Architecture) spécifie une architecture fonctionnelle et des services qui permettent de créer des applications objets distribuées avec des environnements hétérogènes. Dans le modèle CCM, le on définit un ensemble d'interfaces qui exploitent le bus à objets répartis de CORBA pour réaliser un composant [08].

Les composants CCM sont définis par un ensemble d'attributs et par leurs interfaces fournies et requises, appelés respectivement ports d'entrée et ports de sortie. Les attributs de composants sont utilisés pour la configuration au déploiement ou à l'exécution. Les ports définissent les points de communication des composants et sont utilisés pour leur invocation et leur interconnexion. En effet, il n'est pas possible d'appeler un composant sans référencer un de ses ports d'entrée.

Pour que les composants d'une application puissent communiquer, ils doivent être interconnectés à l'aide de leurs ports. Les connexions sont établies entre des ports de sortie et des ports d'entrée qui représentent des interfaces identiques et qui sont de la même nature synchrone ou asynchrone. En d'autres termes, les ports ne peuvent pas être connectés si l'interface qui est requise par le composant avec le port de sortie n'est fournie par le composant avec le port d'entrée [09]. En résumé, un composant CCM est constitué de :

- Une référence de base ;
- Des interfaces : 4 types de port (la facette, le réceptacle, la source et le puits d'événements) ;
- Des attributs ;

À ces éléments constituant un composant CCM est associé une Fabrique (home) qui constitue un gestionnaire pour les instances de composants. Chaque composant CCM a une référence, dite référence de base, et il peut avoir un ou plusieurs attributs qui représentent ses propriétés configurables. En plus, il est doté de multiples interfaces appelées "ports". Ces interfaces se classent en deux types : d'entrés et de sorties (fournies et requises).

Deux modes "ports" fournies : facettes (facets) pour les invocations synchrones, et puits d'événements (event sinks) pour les notifications asynchrones. Les facettes ont le même cycle de vie que celui du composant qui les encapsule. Elles permettent d'avoir des points de vue différents sur un même composant. A partir de la référence de base d'une instance de composant, il est possible de naviguer entre les différentes facettes au cours d'exécution.

Chaque composant CORBA est associé à une Fabrique (home). Une Fabrique est un gestionnaire des instances de composants qui permet de créer, à l'exécution, des instances de même type, et il permet également, la recherche d'une instance en se basant sur des clés. Néanmoins, il est possible de définir différents types de Fabrique pour un type unique de composants. Elle est aux composants ce que l'opérateur "new" est aux objets.

Les types de composants sont définis indépendamment des types des Fabriques. Une définition de Fabrique, cependant, doit indiquer exactement un type composant qu'elle contrôle. Différents types de Fabrique peuvent gérer le même type de composant, bien qu'ils ne puissent pas gérer le même ensemble d'instances de composants [06].

8.4. COMPARAISON DES MODELES DE COMPOSANTS

Après cette brève étude des trois modèles, le tableau illustré dans la figure 3 résume les principales caractéristiques de chaque modèle [10, 06].

Critère de comparaison	Composants		
	EJB	**COM+**	**CCM**
Architecture	J2EE	. NET	CORBA
Editeurs de Composant	Plus de 30	Microsoft	OMG (850 membres)
Interpréteur	JRE (Java Run-time Engine)	CLR (Common language Runtime)	IR (Interfaces Repository)
Intégration	MV	Par un système de type commun +MV pour ce type de système	Au niveau IDL
Langages de rogrammation	Mono-langage (Java)	Multi-langage	(27 langages) Multi-langage
Implémentation	Bien pour les entreprises traditionnelles	Bien pour les applications des bureaux traditionnels	Bien en générale dans client Web
Plate-forme	Multi -plate-forme	Mono-plate-forme	Multi-plate-forme
Connecteur	Non	Non	Oui
Relation de composition	Non	Non	Non

Figure 3. Une comparaison entre COM+, CCM et EJB [06].

9. LES SERVICES WEB

9.1. LES SERVICES WEB ET LE COMPOSANT

Les nouvelles technologies des systèmes à base de composants distribués offrent des moyens de plus en plus puissants pour bâtir et de réaliser une interopérabilité entre les systèmes et les applications de l'entreprise [12]. Parmi celles-ci les services web apparaissent comme une nouvelle possibilité exploitant plus directement et plus simplement les possibilités offertes par l'infrastructure Internet. Un service Web est un composant logiciel, bien défini, qui expose son interface sur Internet. Par rapport aux technologies CORBA/COM/DCOM et Java RMI, elles ne prend pas en compte la contrainte d'une API unique pour l'invocation des

services de distribution par les applications désirant interopérer et reposent sur des invocations de service par l'échange de messages en s'appuyant sur la technologie XML. Cette caractéristique et l'utilisation directe des protocoles Internet qui la suggèrent comme une technologie de référence particulièrement indiquée pour l'interopérabilité de systèmes hétérogènes au sein de l'entreprise et plus encore lorsqu'ils sont situés dans des entreprises différentes.

9.2. DEFINITION

Le paradigme des services Web repose sur une architecture à base de composants qui utilise des protocoles Internet comme infrastructure pour gérer la communication entre les composants. Il offre un modèle à base de composants (les services Web) en ligne encapsulant une application logique derrière une interface interactive uniforme et standardisée. Le terme «services Web» est utilisé pour désigner aussi bien le paradigme que l'idée d'un composant accessible à partir du Web. La définition du service Web divise clairement la vision académique et industrielle. Certains restreignent les composants services Web à ceux qui utilisent les standards de la technologie services Web tandis que d'autres proposent une définition plus générale.

Définition [73] : Un service Web est une application accessible à partir du Web. Il utilise les protocoles Internet pour communiquer et utilise un langage standard pour décrire son interface.

Nous choisissons cette définition qui est la plus générale et la plus complète puisqu'elle souligne les aspects clés qui donnent à toute application en ligne un accès généralisé.

9.3. ARCHITECTURE

La définition de l'architecture services Web consiste à mettre en évidence les concepts, les relations entre ces concepts ainsi qu'un ensemble de contraintes qui assurent l'objectif premier des services Web à savoir l'interopérabilité. Ces éléments sont structurés à travers un modèle de déploiement et de fonctionnement des services Web. Le concept des Web Service s'articule actuellement autour des trois acronymes suivants :

- SOAP (Simple Object Access Protocol) est un protocole d'échange inter-application indépendant de toute plate-forme, basé sur le langage XML. Un appel de service SOAP est un flux de caractères en code ASCII encadré dans des balises XML et transporté dans le protocole HTTP.
- WSDL (Web Services Description Language) donne la description au format XML des Web Services en précisant les méthodes pouvant être invoquées, leur signature et le point d'accès (URL, port, etc..). C'est, en quelque sorte, l'équivalent du langage IDL pour le modèle distribuée CORBA.
- UDDI (Universal Description, Discovery and Integration) normalise une solution d'annuaire distribué de Web Services, permettant à la fois la publication et l'exploration. UDDI se comporte lui-même comme un Web service dont les méthodes sont appelées via le protocole SOAP.

10. DEFINITION DE L'INGÉNIERIE BASEE COMPOSANT POUR LE WEB

La construction et le déploiement à grande échelle des applications Web complexes en utilisant les anciennes technologies fait apparaître quelques difficultés résumés par les chercheurs en :

- la maintenance et l'évolution des applications Web prennent beaucoup de temps. Ce qui la rend une tâche fastidieuse.
- processus de création des applications Web est souvent ad hoc et en désordre.
- nombreux projets Web soufrent des exigences de changement continu.

Il devient clair que la construction et l'évolution des applications Web requièrent un support similaire à celui qui est disponible pour les applications traditionnelles : les modèles, les méthodes et les principes du génie logiciel. Le Web avec ses caractéristiques particulières et ses propriétés devient un nouveau domaine d'application du génie logiciel. Cela a toujours besoin des bases théoriques saines. Cette nouvelle discipline connue sous le nom d'ingénierie du Web « *Web engineering* » promet à réduire les coûts et augmente la qualité durant le développement et l'évolution des applications Web. Elle est définie comme suit :

L'ingénierie de Web est une approche quantifiable, discipliné et systématique pour le développement, la mise en service et la maintenance des applications hypermédias [22].

L'ingénierie de Web est l'application des études du génie logiciel pour le développent Web [43 ,44 ,45].

L'ingénierie du Web, n'a pas concentré sur comment développer des nouveaux sites Web autant que sur comment les développeurs, dans un systématique et quantifiable chemin, puissent réduire le coût de développement en supportant les attribues de qualité tel que les exigences de changement, les futures extensions et les évolutions [29].

Un des problèmes les bien connus de l'ingénierie du Web réside dans le fait que son modèle d'implémentation basé ressources (fichier) n'a été jamais vraiment destiné aux types d'applications Web complexes [22, 23]. Le modèle d'implémentation du Web, fondé sur les fichiers comme ressources, ne permet pas de modéliser les concepts de haut niveau de la conception qui passe au-delà de volumes supportés par les fichiers. Les structures telles que les dialogues dans une session d'application, les objets d'interface utilisateur présentés dans une application interactive, les éléments d'identités corporatives représentées par une seule entité de conception devraient être copiés dans de nombreux fichiers (ressources). Cela rend la maintenance de l'application difficile et impose une évolution ponctuelle de l'application en détruisant les concepts originaux de la conception.

Bien qu'il y a des méthodes de conception et des systèmes disponibles qui soutiennent la projection des concepts de haut niveau et les entités de petites tailles au développement Web (tel que OOHDM [39], RMM [38] ou Jessica [40]), la projection inverse et la possibilité de la distribution et la gestion de ces concepts pour la maintenance et la réutilisation sont moins supportés. De plus, à cause du manque de la structure dans le code des applications Web, il est difficile de :

- réutiliser le code dans l'application,
- définir un code pour la réutilisation,
- réutiliser un code pour d'autres systèmes cibles.

Ces restrictions dans tout le processus de développement empêchent une production et une maintenance moins coûteuses tout en garantissant la qualité [17].

L'expérience positive sur le développement des logiciels à base de composants et ses avantages [41, 42, 13] rend cette approche souhaitable pour adapter l'utilisation de la technologie basée composant dans le développement et l'évolution des applications Web. C'est aussi une condition pour importer entièrement les avantages d'application de processus de l'ingénierie moderne orientée réutilisation dans les technologies du Web. D'ailleurs, une technologie appropriée basée composant pour le Web implique un modèle de processus de développement de logiciels qui décrit la construction et l'évolution basée composant des applications Web en prenant en compte les bases et les principes fondamentaux du Web [28].

La nécessité de construction d'une application Web à partir des composants a introduit l'ingénierie de Web Basée Composant «*Component Based Web Engineering* » comme une approche de l'engineering du Web. Ce qui soutient la réutilisation des composants et des connaissances [29]. Dans [28], la définition de CBWE est la suivante :

> L'ingénierie de Web Basée Composant (CBWE) : C'est la production des applications Web par la composition des composants existant en utilisant un processus bien défini; celui-là inclut la réutilisation systématique des composants et les connaissances du domaine.

Le but de construction des applications Web en suivant CBWE est de considérer un composant comme une unité de composition. On a besoin aussi à un processus qui définit le développement avec un soutien dédié à la réutilisation des solutions précédentes (composantes existant et éprouvés) [19, 20].

11. LES APPROCHES DE DEVELOPPEMENT DES APPLICATIONS WEB BASES COMPOSANT

Dans cette section, nous allons présenter deux travaux sur le développement à base de composant pour le Web. Le premier est l'approche WebComposition qui permet de modéliser les applications Web par des composants. Ce modèle ne supporte ni un modèle conceptuel

réel ni des niveaux hauts d'abstraction. Le deuxième est un modèle basé composant d'applications Web. Ce modèle est dirigé par le contenu et ne prend pas en compte l'aspect service des applications Web**.**

11.1. WEBCOMPOSITION

L'approche WebComposition permet de modéliser les applications Web par des composants. Elle fait une passerelle entre la conception et l'implémentation en permettant l'interprétation complète des objets de la conception dans des composants de différentes tailles. La résolution d'un composant (la nature de contenu et sa taille) n'est pas préréglée mais elle est très dépendante du niveau de détail demandé par les concepts de conception en question (page, lien, texte, etc.). Un composant peut représenter, par exemple, une caractéristique atomique tel que la police ou la taille d'attribue, une structure complexe de navigation ou simplement une composition d'autres composants.

De cette façon, l'approche WebComposition a permis la réduction de trou entre le modèle de conception et celui d'implémentation et le passage entre eux en offrant une haute résolution de modèle d'implémentation qui trouve pour chaque élément de conception son correspondant dans l'implémentation. Des fichiers complets en langage de balise sont générés par la compilation des compositions de ces composants [22].

L'approche WebComposition vise à composer la réutilisation de la conception et la réutilisation du code. Il est bien connu, comme il a été dit au paragraphe 2 de ce chapitre, que la réutilisation facilite le développement durant le cycle de vie de l'application [30].

L'approche WebComposition consiste en un modèle de processus pour le développement et l'évolution en plus d'un modèle de composant. Démarrant par un cadre réutilisable vide, l'application grandit selon l'évolution du modèle de processus WebComposition (Process WebComposition Modeler WCPM). Ce cadre est la base de n'importe quelle application et son squelette reste comme un support de cette application évoluée durant son cycle de vie. Ce squelette est appelé le bus d'évolution "Evolution bus" dans le modèle de WebComposition.

Les parties ajoutées à l'application sont également des composants. Les nouvelles parties de codes sont développées indépendamment des ressources (les fichiers) sous-jacentes comme des composants uniformes [32].

WebComposition définit un modèle semblable à celui de l'orienté-objet qui se base sur la décomposition des applications Web en composants modélisant les entités Web de granularité arbitraire et à un certain niveau d'abstraction [33]. Le modèle de WebComposition est basé sur un paradigme de prototype-instance [35], par opposition au modèle orienté-objet qui se base sur les classes et nécessite la définition des classes d'instanciation des objets. Dans le modèle WebComposition, un composant peut utilisé comme une classe abstraite e. i. chaque composant peut être un prototype pour un autre composant. Les nouveaux composants peuvent être construits comme une composition des propriétés ou une composition des composants existants à travers le prototypage et le référencement.

Le prototypage est un mécanisme de base de WebComposition pour implémenter le partage du code entre les objets par l'héritage. Une autre possibilité de partage le code des composants est de permettre les références multiples sur le même composant. Le partage du code est fondamental non seulement pour la réutilisation mais aussi pour la maintenance comme il aide à garder les modifications locales. Cela est par opposition aux autres modèles orientés objet proposé pour le Web, par exemple. WOOM [30].

Les composants peuvent référencer un autre composant par le modèle d'agrégation (has-part) ou de spécialisation (inherits-from). Par exemple, un composant modelant une page pourrait référencer un autre composant modelant une partie de page ; un composant modelant la structure de navigation pourrait référencer des composants modelant les liens et les ancres concernés. Par le moyen de référence par type, les composants peuvent référencer les composants du prototype à partir héritent les états et les comportements. Le model WebComposition soutient l'ordre de l'héritage multiple, cela permet au composant d'avoir plus d'un parent [34].

Dans le Web, selon l'approche WebComposition, les entités sont modélisées comme des composants avec un état et un ensemble d'opérations spécifiant le comportement de ces composants [26]. Un composant peut modéliser les petites entités (par exemple les liens

individuels), comme il peut modéliser plusieurs fragments d'une ressource. Naturellement, un composant peut être aussi associé avec une ressource complète. Un composant peut modéliser un groupe de ressources, par exemple une séquence de dialogue implémentée par un certain nombre des hyperliens.

11.1.1 LE MODELE DE PROCESSUS

L'ingénierie à base de composant pour le Web dépend d'une définition adéquate du processus de développement. Les processus de développement proposés, dans le passé, incluant OOHDM [39] ou JESSICA [40], manquent de soutien de la réutilisation et souvent ils conviennent seulement à certains types d'applications Web. En plus, ces processus et ces méthodes sont concentrés sur le paradigme orienté-objet et ainsi ils ne provoquent pas directement l'usage des composants. Pour faire face à ces problèmes, Gaedke [28] présentent le modèle de processus de WebComposition « Process Web Composition Modeler » (WCPM) pour supporter la réutilisation et la maintenance. C'est un modèle de processus ouvert et convenable pour une large variété d'applications. Un autre facteur influant l'apparition de WCPM est le changement des caractéristiques des applications Web de celles des systèmes d'informations normaux à des applications logicielles qualifiés avec les changements rapide du Web [29].

Le WCPM ne définit pas exactement un processus spécifique de développement, mais il est plutôt un frameworks dans lequel les processus arbitraire de développement comme OOHDM peuvent être intégrés. En plus, WCPM soutient directement la réutilisation à travers sa gestion de réutilisation des composants. C'est un modèle de processus ouvert qui convient aux grandes variétés d'applications [09].

WCPM consiste en plusieurs phases. Ses phases sont dérivées à partir des phases commun du modèle de processus moderne (orienté objet) aussi bien que des solutions abordant les besoins de la réutilisation de logiciel [28].

Basé sur le modèle en spiral [45], le modèle de processus de WebComposition « WebComposition Process Model » soutient l'évolution d'application Web dans un cycle de vie de développement court. La figure 4 représente ce cycle suivant WCPM. La construction et la maintenance d'une application sont un cycle infini d'évolution de l'analyse et de la

planification, d'évolution de la conception et d'évolution de la réalisation. Cependant, l'évolution du cycle n'est pas toujours un processus homogène. Il est généralement divisé en plusieurs sous processus par une décomposition hiérarchique.

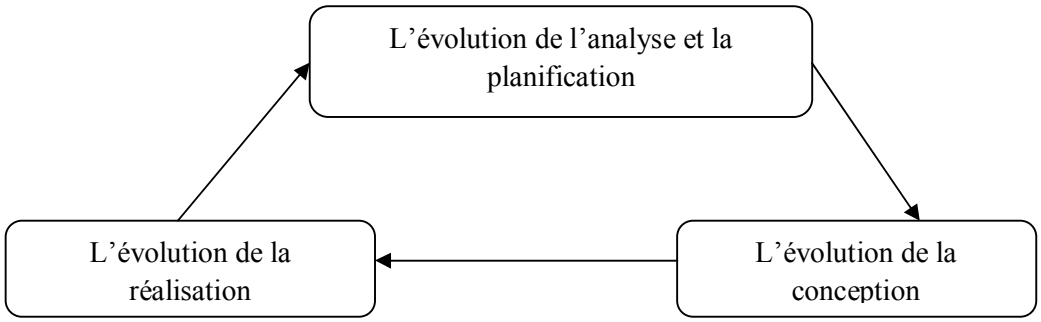


Figure 4. L'évolution spirale du modèle de processus de WebComposition [28].

WCPM considère le développement de chaque composant comme un sous-processus de développement. Il n'y a pas une restriction sur le type de processus utilisé comme sous-processus. Par exemple, un composant peut être développé selon le modèle en cascade ou n'importe quel autre type de processus de développement [44]. Ainsi, WCPM est une extension fractale du modèle en spiral. Il admet le processus de développement le plus approprie pour être utilisé au niveau du composant. Donc, il s'agit d'un modèle de processus ouvert. Alors, WCPM n'est pas restreint à certaines classes d'applications Web.

Un processus de réutilisation peut être utilisé également comme un sous-processus. Néanmoins, un mécanisme de gestion est nécessaire pour résoudre le problème de synchronisation des sous-processus et d'administration des composants existants. Un tel mécanisme est donc responsable de maintenir un processus global de développement et une réutilisation coordonnée et explicite. La coordination des modèles est possible par une gestion de réutilisation explicite et coordonnée. La figure 5 présente la coordination des différent processus avec le modèle de processus de WebComposition, ce qui illustre son ouverture.

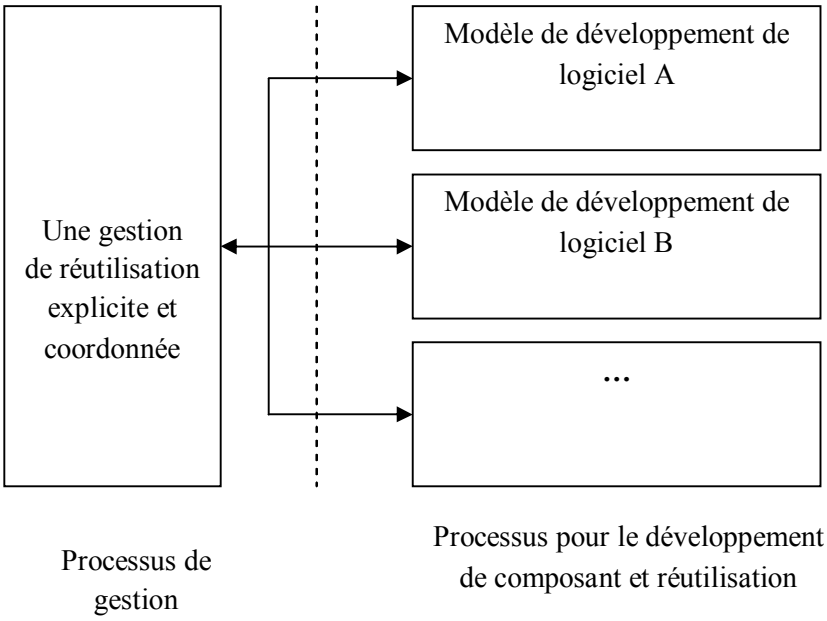


Figure 5. La coordination des processus orthogonaux dans le modèle de processus WebComposition [28].

Dans [46], un entrepôt de composant « component repository » était présenté comme un mécanisme de gestion de la réutilisation. Ce mécanisme administre l'entrepôt des composants et offre une variété de méthodes pour l'accès et la récupération des composants. Donc, il se décide au début du processus de développement du composant comment ce composant sera développé. C'est la première demande à travers l'interaction avec la gestionnaire de la réutilisation. Si un composant existe déjà, il peut être utilisé par la réutilisation de composant ou son ajustement. Seulement si aucun genre de réutilisation est possible, un processus de développement approprié est choisi pour le développement du composant concerné. La figure 6 présente ce concept par la représentation de deux composants où le premier a été développé selon le modèle de cascade (choute d'eau) et le deuxième par la réutilisation. Tous les deux processus se collaborent avec la gestion de réutilisation pour l'échange des composants existants et donc ils peuvent coordonner et synchroniser entre eux.

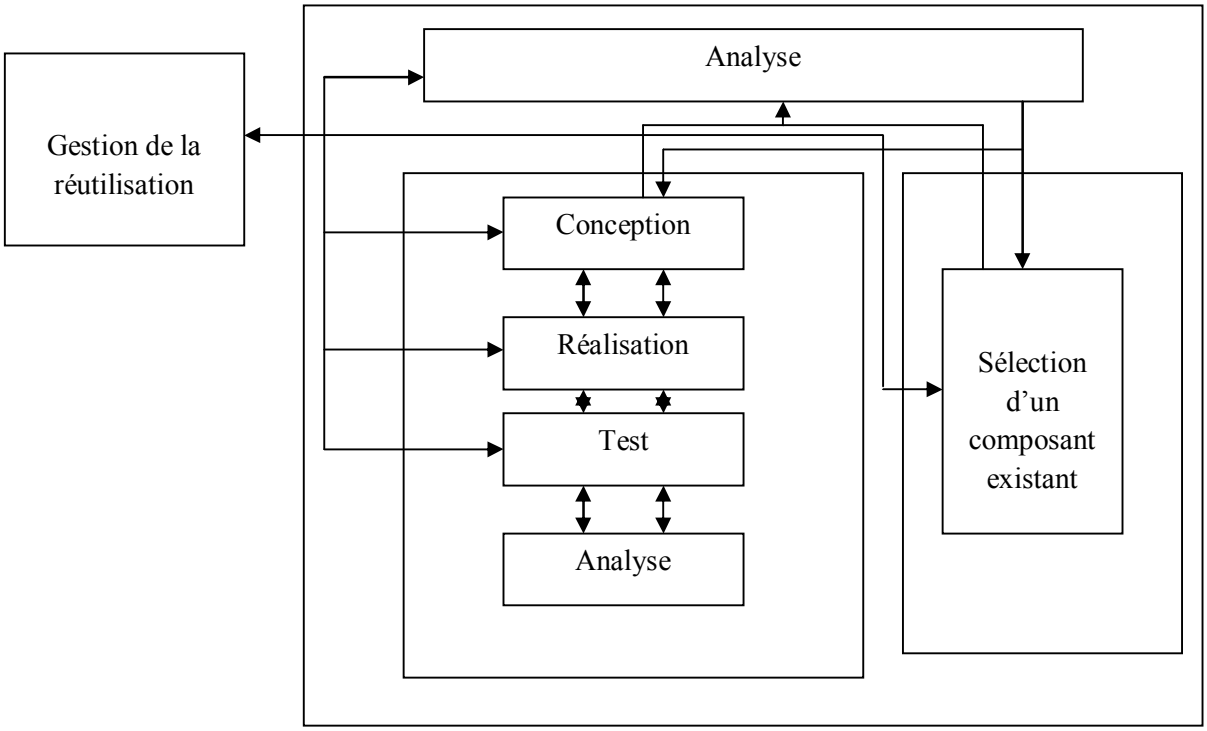


Figure 6. Un exemple de WCPM [19].

Le problème de récupération de composant pour une réutilisation potentielle dans l'entrepôt des composants a été résolu. Les caractéristiques des composants et les propriétés forment ensemble un méta-données de composants. En cas de la réutilisation, la personne demandant la réutilisation et cherchant un composant existant, (consommateur de composant « component consumer ») est capable de formuler une simple requête sur le méta-données et non pas sur le composant lui-même. C'est parce que le méta-données contient les conditions satisfaisantes le composant désiré et non pas le composant lui-même. La gestion de la réutilisation stocke le composant et son méta-données à la fois. Le modèle de méta-données cependant ne doit pas se limiter seulement à la représentation dont le but est de permettre des requêtes performantes sur les composants dans le modèle approprié. Donc, la représentation du composant dans l'entrepôt de composants et son modèle de méta-données sont découplés.

Le producteur des composants fournit plusieurs représentations d'un composant donné dans la méta-données pour augmenter la probabilité de la récupération du composant par le consommateur [09].

11.1.2. LES CONCEPTS DE WEBCOMPOITION

a) COMPOSANT

L'approche WebComposition est basée sur le concept de composant. Elle soutient les composants de différents niveaux d'abstraction. Les composants sont de granularité arbitraires ; ils peuvent être aussi petits qu'un seul attribut (par exemple l'attribut de police pour une balise HTML) ou aussi grand que des pages Web complètes.

Un composant est un simple assemblage des paires nom-valeur (name-value) qui sont appelées propriétés. De plus, comme une propriété peut être implantée par une simple fonction, le concept de polymorphisme de l'approche orientée objet est soutenu. Dans ce sens, un composant qui représente une page Web peut être composé par des composants de référence qui sont responsables de livraison du code de l'entête, du corps, et du pied. Les objets instanciés peuvent hériter les capacités des composants par une simple utilisation de ces composants comme des prototypes [30].

Les composants sont définis par leur état et leur comportement. L'état est défini par une liste des pairs des propriétés de type nom-valeur. Par exemple, un composant modélisant un élément HTML doit avoir des propriétés concernant les attributs de cet élément HTML. Le comportement est défini par un ensemble d'opérations.

Tous les composants doivent fournir des opérations *getProperties, setProperties* et *generateCode.* La première opération réalise l'accès en lecture et la deuxième en écriture pour manipuler l'état du composant. L'opération *getProperties* et *setProperties* implémentent ensemble la persistance du composant, la sérialisation et la désérialisation de l'état du composant pour poursuivre le stockage dans le l'entrepôt de composants. L'opération *generateCode* implémente le fonctionnement du composant définissant son état comme une représentation dans le Web. Généralement la représentation est en HTML. Le rapport entre le composant et ses représentations dans le Web est une relation modèle-affichage (model-view) [34].

Un composant est appelé composant primitif s'il n'est pas décomposable ultérieurement; un composant composite consiste en un ou plusieurs autres composants (primitifs ou

composites) [13]. Pour un composant primitif, par exemple texte, sa représentation est directe. Dans le cas d'un composant composite sa représentation implique une invocation des représentations de ses composants enfants. Pour les composites simples tels qu'une liste, l'opération *generateCode* peut être une simple itération de ses composants enfants, en invoquant leurs opérations respectives. Dans des cas plus complexes, par exemple les tables, le composite doit engendrer son propre code en plus du code de ses composants enfants [34].

b) SERVICE

Un domaine du composant dans le modèle WebComposition est appelé service. Les services sont les modélisateurs primaires de l'abstraction du WebComposition. Un service doit intégrer tous les aspects d'une application Web. Dans le WebComposition, cinq différentes caractéristiques d'application Web sont considérées. Chaque caractéristique est exprimée dans un composant de WebComposition. Les caractéristiques peuvent être utilisées pour la classification des composants et leur division en groupes. Un service peut être composé par plusieurs autres composants. Un service est composé d'un composant (il peut être un composant vide) de chaque groupe. Dans le WebComposition la correspondance groupes/caractéristiques suivante est utilisée :

- Contenu : les composants de ce groupe fournissent les données aux services. Ils ne définissent pas comment les données sont accumulées et comment elles sont récupérées.
- Présentation : les composants de ce groupe sont des modèles de mise en page. La caractéristique commune entre ces composants est l'encapsulation du code (tel que HTML or WML).
- Navigation : la séparation entre la navigation et le modèle permet une modélisation navigationelle des patterns
- Processus : les composants implémentent le fonctionnement nécessaire pour l'exécution d'un service interactif [32].

c) WEB COMPOSITION MARKUP LANGUAGE (WCML)

Les composants du WebComposition sont définis dans WebComposition Markup Langue (WCML) [27, 23, 29]. La WCML est une extention du langage XML (eXtensible Markup

Langue) qui prépare le chemin pour l'ingénierie orienté objet du Web basé sur l'approche WebComposition. La WCML permet aux développeurs la réutilisation d'une conception et des fragments de code en fournissant une notation simple capable de définir les objets et leurs relations [27].

Comme tous les documents XML, le document WCML consiste en une balise de début et d'un contenu. Le document WCML de définition de type décrit une notation de balises pour les composants. Les composants décrits dans WCML résident dans un document WCML, qu'on appelle un entrepôt virtuel de composant (virtual component store) [30].

La livraison de composants WCML, qui sont décrits sémantiquement en utilisant XML, est possible d'être établis par la livraison des documents WCML correspondants. Le déploiement à travers le Web et la génération du code au moyen de compilateur de WCML. C'est ce qui illustré dans la figure 7.

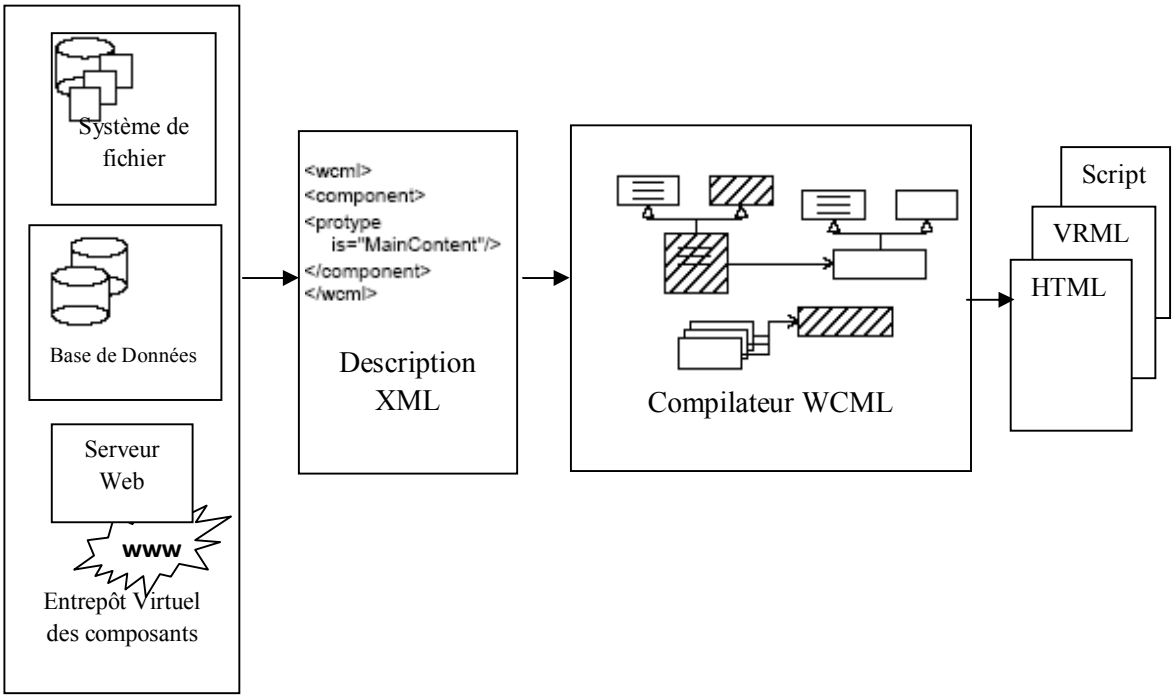


Figure 7. Intégration et transformation de WCML [23].

Le compilateur de WCML est implémenté en Java en utilisant le parseur standard XML, qui est responsable de la lecture et l'analyse des documents XML/WCML. Ainsi, la tâche

restreinte du compilateur WCML est d'accomplir l'opération de présentation des composants et d'associer les références des composants à leurs propriétés avec le maintien de ces références aux ses composants originaux dans les différents documents HTML.

L'entrepôt virtuel des composants représenté par des documents WCML contenant un ensemble de composants d'une granularité arbitraire. Il peut être implémenté en utilisant un système de base de données, un système de fichier, ou un serveur Web. Le compilateur de WCML transforme les composants, qui sont stockés dans l'entrepôt virtuel, au un modèle d'implémentation du Web [30].

Un composant a un identificateur unique, un ensemble d'attributs, peut hériter à partir d'un ou plusieurs composants prototype et lui même peut être un prototype pour d'autres composants.

Le WCML en plus soutient la définition des hyperliens au niveau conceptuel par les liens avec des autres composants. Ces liens peuvent être définit en dehors des composants eux-mêmes, donnant comme résultat un concept puissant pour définir les différentes structures navigationnelles pour un modèle de composant donné. Le pattern décorateur est utilisé pour étendre le contenu des composants existant avec l'information du spécifique dispositif de modèle. Donc le même contenu peut être présente différemment dépendant du dispositif cible [29].

11 .2. MODELE BASEE COMPOSANT POUR LES APPLICATIONS WEB

Le travail de Xin propose un modèle basé composant pour le développement des applications Web avec un ensemble d'outils supportant sa méthodologie proposée [72]. Ce modèle exprime la composition de composants dans une collection particulière appelée unité Web formant une application Web. Donc l'application Web consiste en des unités Web. L'unité Web peut être également un élément de déploiement.

11.2.1. UNITE WEB

Les unités Web représentent les éléments d'implémentation et d'abstraction dans une application Web. Elles composent le squelette principal d'une application. Au niveau de l'implémentation, une Unité Web est l'unité fondamentale de déploiement sur laquelle les structures physique et logique d'implémentation sont basées. Au niveau abstrait, une application Web est un groupement d'unités Web qui collaborent ensemble pour implémenter les fonctionnalités de l'application. Une unité Web encapsule la présentation et la fonctionnalité intacte dans un composant. Les unités de web communiquent à travers les événements. La communication entre l'utilisateur et les unités de Web est aussi à travers les événements. Ces événements fournissent un format unifié pour les interactions entre les utilisateurs et les unités Web et l'interopérabilité entre les unités Web. Une unité Web doit satisfaire quatre exigences :

1. reconnaître son environnement externe;
2. fournir *une présentation* à cet environnement externe;
3. fournir des facilités de *auto-maintenance*;
4. accéder à ses données par une interface standard de données.

11.2.2. ARCHITECTURE D'UNE APPLICATION WEB

L'architecture d'une application Web est restreinte à une structure d'arbre. Les nœuds dans cet arbre sont homogènes. Chaque nœud est une partie du contenu de l'application. Le contenu désigne ici le contenu dans l'organisation abstraite d'une application Web. Il n'est pas associé avec le contenu qui est délivré à l'utilisateur du navigateur. Quand le contenu constitue un élément visible alors, il saurait affiché sur l'écran de l'utilisateur. Sinon, il restera que dans l'architecture de l'application.

L'arbre de contenu et les relations entre les nœuds peuvent avoir différents sens quand elles sont appliquées à des situations différentes. Par exemple, il peut simuler l'organisation d'une entreprise dans laquelle l'arbre représente la hiérarchie de la structure des services dans l'entreprise. La relation entre deux nœuds représente la relation de gestion entre deux

services. Dans un site Web orienté contenu, cette structure d'arbre représente l'organisation de contenu dans le site.

11.2.3. PROCESSUS GENERAL DE LA METHODOLOGIE BASEE COMPOSANT

En se basant sur le modèle basé Composant, le composant de l'application Web, c'est-à-dire l'unité Web, est le centre du développement dans toutes les phases de développement du logiciel. La méthodologie de développement est composée de l'analyse et la conception, l'implémentation et la maintenance. On utilise le cycle de vie en spirale pour décrire la méthodologie de développement.

Durant la phase de conception, l'application est décomposée et transformée en un arbre de contenus avec laquelle l'architecture de l'application est établie. Les nœuds de contenu dans l'arbre sont conçus par les unités Web. Chaque unité Web est spécifiée par ses fonctions, ses événements et ses contenus. La navigation entre les unités Web est conçue après la conception des unités Web. Après ces étapes, le résultat de conception est passé à un créateur graphique pour concevoir la présentation et l'interface utilisateur de l'application. Cette conception est basée sur la spécification d'unités Web et la navigation. La conception graphique produit le prototype de la présentation de l'application Web. À travers l'intégration dans le processeur de présentation d'unité Web, le prototype de l'application est produit. La phase de conception est un traitement itératif à partir de la construction de l'arbre du contenu à la création de prototype.

L'implémentation de l'application Web est basée sur des unités Web conçues dans la phase de conception. Dans l'étape de l'implémentation, les unités Web sont configurées pour qu'elles soient utilisées dans la génération de l'application. Quand l'unité Web devient un composant prêt à être utilisé, sa présentation sera configurée.

12. CONCLUSION

Le développement à base de composant peut être vu comme l'évolution logique de l'approche orientée objets. Cela était formulé dans le cadre d'une approche à base de composant. Cette approche a comme objectif primordial la réutilisation et l'intégration de modules logiciels conçus et développés par des personnes différents tout en appuyant sur la

construction par assemblage. Elle agit pour diminuer les coûts de production et augmenter la productivité et la qualité du produit final.

Dés son apparition, le Web ne cesse pas à s'évoluer en faisant évoluer et améliorer comme ses différentes technologies. Pour ce faire, les chercheurs ont rencontré plusieurs problèmes qu'ils ont essayé de dépasser en appliquant et en adaptant des méthodes éprouvées de développement de génie logiciel pour le Web afin d'obtenir des résultats satisfaisants. C'est le cas de l'approche à base de composant qui était projetée sur le Web pour faire face à certains problèmes. On a obtenu comme résultat l'approche de développement à base de composant pour le Web.

Dans ce chapitre, nous avons vu la motivation de l'apparition de l'approche de développement à base de composant ainsi que ses buts, ses bénéfices attendus et les définitions de la notion de composant. Ensuite, nous avons présenté l'architecture d'un composant logiciel et ses différentes catégories. En plus, nous avons discuté la relation entre les services Web et l'approche à base de composant. Nous avons vu les insuffisances des méthodes de conceptions des applications Web telles que le soutien faible de la réutilisation, et comment l'approche à base de composant pour le Web propose des solutions à ces problèmes. Enfin, nous avons présenté deux exemples sur l'approche à base de composant pour le Web avec ses modèles de processus de développement.

On remarque que les méthodes proposées sont soit orientées données soit orientées présentation mais la demande actuelle et en vogue du Web, après sa nouvelle architecture adoptée ; basé services Web, est le développement des méthodes de développement d'applications Web basé service pour bien profiter de cette nouvelle architecture dans la minimisation de coût et le temps de développement.

Chapitre 3

1. INTRODUCTION

Les volumes de connaissances humaines avaient augmenté, les sciences et les techniques étaient évoluées. L'IA devenait incapable à résoudre ces problèmes, et encore la plupart des travaux, dans le monde réel, sont réalisés par groupe. De ce fait qu'il est apparu une nouvelle approche : Intelligence artificielle distribué (IAD) qui est un issu de la rencontre de l'Intelligence Artificielle et des Systèmes Distribués [74]. Cette émergence a conduit à l'apparition d'une nouvelle technique de conception (programmation) adéquate à l'IAD. Qui est l'approche multi-agents.

Les travaux en IAD se différencient de celles de l'IA classique car la majorité des travaux en IA se sont efforcés de développer les capacités d'un agent unique censé représenter un être humain dans l'accomplissement d'une tâche requérant des connaissances et des raisonnements, alors que en IAD, un système est conçu comme une société d'agents autonomes collaboratifs ou coopératifs.

Le Web, comme une nouvelle orientation des différents systèmes dans différents domaines d'applications, ouvre les portes aux chercheurs pour présenter tout travail qui tend à rendre ce monde plus efficace pour le soutien de ses systèmes. Les chercheurs, après une longue recherche, ont orienté vers l'utilisation des multis-agents dans le développement des applications Web. L'utilisation des systèmes multis-agents dans la réalisation des applications a prouvé son efficacité en terme de performance et de qualité de développement.

2. AGENT

Aujourd'hui, il n'existe pas de définition universelle de l'agent. L'existence de plusieurs définitions d'un agent est due essentiellement à la multiplicité des facettes et caractéristiques que renferme ce concept. Chaque définition met en avant des aspects différents de l'agent. Elles sont généralement influencées par une vision particulière de l'agent ou par un contexte d'étude. Nous en avons choisi quelques unes, qui sont cité dans [73]:

Définition 1: Un agent qualifie toute entité capable de percevoir son environnement à travers des unités sensorielles et qui agit sur celui-ci à travers des unités d'actions.

Définition 2: Les agents autonomes sont des systèmes de calcul plongés dans un environnement complexe et dynamique, ils perçoivent et agissent dans cet environnement de manière autonome, leurs actions sont dirigées par des objectifs ou des tâches pour lesquelles ils sont destinés.

Définition 3: Un agent est une entité logicielle qui présente les propriétés suivantes:

- autonomie : les agents fonctionnent selon leurs propres directifs. Ils ont un contrôle sur leurs actions et leurs états internes.
- sociabilité : ils interagissent entre eux en utilisant certains langages de communication.
- réactivité : les agents perçoivent leurs environnements et réagissent en conséquence dans un intervalle de temps acceptable.
- pro-activité : ils prennent l'initiative d'un comportement afin d'atteindre leurs objectifs.

Un agent est une entité informatique qui :

- Se trouve dans un système informatique ouvert comprenant un ensemble d'applications, des réseaux et de systèmes hétérogènes
- Peut communiquer avec d'autre agent.
- Est mue par un ensemble d'objectifs propre.
- Possède des ressources propres.
- Ne dispose que d'une représentation partielle des autres agents.
- Possède des compétences (services) qu'elle peut offrir aux autres agents.
- A un comportement tendant à satisfaire ses objectifs, en tenant compte d'une part des ressources et des compétences dont elle dispose, et d'autre part de ses propres représentations et des communications qu'elle reçoit.

3. MOTIVATION

Les raisons (les plus généraux) qui poussent cette approche à apparaître sont [75]:

- Les limites de l'intelligence artificielle (IA) classique sur le plan de structure et d'organisation de la connaissance,

- a nécessité de trouver des techniques performantes de modélisation en simulation,
- La robotique miniature, c'est-à-dire faire coopérer plusieurs petits robots au lieu d'un seul plus grand,
- Le développement de l'informatique et des systèmes distribués; ce qui signifie la capacité de faire coopérer plusieurs logiciels pour la réalisation d'un objectif commun.

4. TYPE DES AGENTS

Il y a d'agents plusieurs types d'agent. Ils sont définis [75]:

- Agent collaborateur (AC) : c'est un agent qui possède, à la fois, les deux caractéristiques : l'autonomie et la coopération.
- agent d'interface (AI) : c'est un: l'apprentissage et l'autonomie. Cet agent est nécessaire à chaque fois que l'utilisateur est présent.
- agent mobile (AM).
- agent d'information/Internet (AII) : il peut être de l'une des deux première catégories (statique, mobile, cognitif et réactif.
- agent réactif (AR).
- agent hybride (AH) :c'est un agent qui combine plus de deux philosophies d'agent.
- agent smart (AS) : c'est un agent qui possède, à la fois, les trois caractéristiques (la coopération, l'apprentissage et l'autonomie).

5. SYSTEME MULTI-AGENT

Il est plutôt rare que les concepteurs d'agents n'aient besoin que d'un seul agent dans l'environnement qu'ils construisent. Lorsque plusieurs agents se retrouvent dans un même environnement et que ces agents ont besoin d'interagir entre eux, on parle alors de système multi-agent. Ainsi, un système multi-agent (SMA) est un ensemble agents regroupés dans un environnement et capables d'interagir ensemble en vue de coopérer, de coexister ou de se concurrencer.

Définition 1: Un SMA est une connexion d'entités (agents) faiblement couplées qui interagissent afin de résoudre des problèmes qui dépassent les capacités et les connaissances de chacun.

Cette définition souligne les aspects et les objectifs des SMA. Deux points sont importants. Le premier est que les entités ou les agents doivent être faiblement couplés. Le deuxième point est la distribution des capacités et des connaissances entre les entités face aux exigences des problèmes traités par le système. Les agents doivent être conçus en totale indépendance les uns des autres et indépendamment de la problématique ou du type d'application. Seules les capacités ainsi que les connaissances de l'agent doivent être prises en considération lors de la phase de conception [73].

> Définition 2: Un SMA est un système composé d'entités appelées agents qui communiquent afin de se coordonner pour la résolution d'une tâche globale. Les agents agissent dans un environnement. L'organisation des agents apparaît par l'intermédiaire de la coordination. Le système est donc composé d'agents, d'environnement, d'interactions et d'organisation.

Cette définition est plus spécifique, elle définit des aspects importants tels que l'importance de la représentation de l'environnement dans un système multi-agent. Elle souligne également des concepts nécessaires à la coopération telles que la coordination, l'interaction, l'organisation ainsi que la communication. Le domaine multi-agent couvre plusieurs champs de recherche qui mobilisent, au-delà de l'informatique et de l'IA, plusieurs autres disciplines.

Les agents sont, en général, situés dans un environnement contenant également des entités passives, manipulées par les agents (par exemple, des ressources, des données, des objets physiques, etc) et communément appelées objets. Chaque agent n'a qu'une connaissance partielle de son environnement et des autres agents. Un système multi-agents est donc intrinsèquement décentralisé [77].

Et d'après Jacques Ferber [76] : " Un SMA est défini comme:

- Un ensemble **B** d'entités plongées dans un environnement **E** (**E** est caractérisé par l'ensemble des états de l'environnement **S**).
- Un ensemble **A** d'agents avec **A** est inclus ou égale à **B**.
- Un système d'action (opérations) permettant à des agents d'agir dans **E**
- Un système de communication entre agents (envoi de messages, diffusion de signaux,…)

- Une organisation **O** structurant l'ensemble des agents et définissant les fonctions remplies par les agents (notion de rôles et éventuellement de groupes)
- Eventuellement: une relation à des utilisateurs **U** qui agissent dans ce SMA via des agents interfaces **U** est inclus ou égale à **A** ".

6. INTERACTIONS ET COOPERATION DES AGENTS

Un système multi-agents (SMA) se distingue d'une collection d'agents indépendants par le fait que les agents interagissent en vue de réaliser conjointement une tâche ou d'atteindre conjointement un but particulier. Les agents peuvent interagir en communiquant directement entre eux ou par l'intermédiaire d'un autre agent ou même en agissant sur leur environnement.

Chaque agent peut être caractérisé par trois dimensions : ses buts, ses capacités à réaliser certaines tâches et les ressources dont il dispose. Les interactions des agents d'un SMA sont motivées par l'interdépendance des agents selon ces trois dimensions: leurs buts peuvent être compatibles ou non. Les agents peuvent désirer des ressources que les autres possèdent; un agent X peut disposer d'une capacité nécessaire à un agent Y pour l'accomplissement d'un des plans d'action de Y [77,78].

7. COORDINATION DES AGENTS

Les agents s'accordent pour partager leurs ressources respectives afin d'atteindre un but commun. Les ressources échangées ont un prix explicite ou implicite. Lorsqu'un contrat est conclu, il y a un accord pour que l'agent contacteur devienne le superviseur de l'agent contractant. De nombreuses organisations mettent en œuvre des processus de coordination mixtes basées sur l'ajustement mutuel, la supervision directe et la standardisation. Ces cadres organisationnels peuvent être des sources d'inspiration quand on doit établir une structure organisationnelle pour un SMA.

La coordination est une question centrale pour les SMA et la résolution de systèmes distribués. En effet, sans coordination un groupe d'agents peut dégénérer rapidement en une collection chaotique d'individus. On pourrait penser que la façon la plus simple de s'assurer un comportement cohérent du groupe d'agents serait de le faire par un agent centralisateur qui détiendrait des informations de haut niveau sur ces agents. Ainsi, l'agent centralisateur pourrait créer des plans d'action et assigner les tâches aux divers agents du groupe. Cette

approche est pratiquement impossible à mettre en œuvre dans des applications réalistes en raison de la difficulté de réaliser un tel agent centralisateur qui puisse tenir compte des buts, des connaissances et des activités de chaque agent : la charge en communication serait énorme, sans compter qu'on perdrait les avantages d'un SMA composé d'agents autonomes. Le contrôle et les informations doivent alors être distribués parmi les agents [77].

8. NEGOCIATION DES AGENTS

Dans un système multi-agents les agents interagissent en vue de réaliser des tâches ou d'atteindre des buts. L'interaction a lieu, d'habitude dans un environnement commun où les agents ont diverses zones d'influence, notamment diverses parties de l'environnement sur lesquelles ils peuvent agir. Ces zones peuvent être disjointes mais, dans la plupart des cas, elles se superposent l'environnement est partagé par les agents. En interagissant dans un tel environnement partagé, les agents doivent coordonner leurs actions et avoir des mécanismes pour la résolution des conflits. La coordination et la résolution des conflits sont surtout nécessaire dans le cas des agents égocentrés (des agents ayants leurs propres buts, désirs, préférences, etc.) ou compétitifs mais aussi bien, parfois, dans le cas des agents coopératifs pour la communication des changements des plans ou l'allocation des tâches. Le mécanisme favori pour la résolution des conflits et la coordination, inspiré du modèle des humains, est la négociation. La négociation est une composante de base de l'interaction surtout parce que les agents sont autonomes; il n'y a pas de solution imposée à l'avance et les agents doivent arriver à trouver des solutions dynamiquement, pendant qu'ils résolvent les problèmes [55].

Pour modéliser la négociation dans un logiciel multi-agents il faut alors prendre en compte les aspects suivants:

- **Le langage de négociation :** le langage utilisé par les agents pour échanger des informations pendant la négociation; le langage de négociation est composé d'un ensemble de primitives de communication.
- **Le protocole de négociation** : l'ensemble des règles qui régit la négociation: les participants possibles dans la négociation, les propositions légales que les participants peuvent faire, les états de la négociation (par exemple l'état initial où commence la négociation, l'état où on accepte des soumissions ou la fin de la négociation) et une règle pour déterminer quand on est arrivé à un accord ou quand il faut s'arrêter parce qu'aucun accord n'a pas n'a pu être trouvé .

- **L'objet de négociation** : un objet abstrait qui comprend les attributs qu'on veut négocier.
- **Le processus de décision** : le modèle que l'agent utilise pour prendre des décisions pendant la négociation. La partie la plus importante de la prise des décisions dans ce cas est la stratégie de négociation qui permet de déterminer quelle primitive de négociation l'agent doit choisir à un certain moment [55].

9. COMMUNICATION DES AGENTS

Les agents peuvent interagir soit en accomplissant des actions linguistiques (en communiquant entre eux), soit en accomplissant des actions non-linguistiques qui modifient leur environnement. En communiquant, les agents peuvent échanger des informations et coordonner leurs activités. Dans les SMA deux stratégies principales ont été utilisées pour supporter la communication entre agents: les agents peuvent échanger des messages directement ou ils peuvent accéder à une base de données partagées (appelée tableau noir ou “blackboard”) dans laquelle les informations sont postées. Les communications sont à la base des interactions et de l’organisation sociale d’un SMA [77].

10. L’ONTOLOGIE ET LES SYSTEMES MULTI-AGENTS

Pour qu’un agent puisse effectuer une tâche automatiquement, il doit connaitre la procédure correspondant à ce qu’il doit effectuer. Cette correspondance est soit un appel à la procédure inscrite dans le code de l’agent logiciel, soit une correspondance entre la description de la tâche à effectuer et la description de la procédure. Pour écrire ces descriptions, il faut les représenter à l’aide d’un langage formel, c’est-à-dire une représentation ontologique. Cette dernière consiste à définir les primitives de représentation du domaine d’application et leurs significations qui seront utilisées pour la représentation des connaissances [54]. Dans un système multi-agent et pour coordonner l'activité d'un ensemble hétérogène d'agents autonomes, il faut que chaque agent communique dans un langage compréhensible par tous les autres. On observe que dans un système ouvert un tel langage peut constituer une interface entre les agents.

Un langage de communication agent (ACL Agent Communication Language) doit être conçu comme un langage de haut niveau qui assure l'échange d'états mentaux et le sens du vocabulaire. Le format utilisé pour l'échange des connaissances est donné par un langage de

contenu, indépendant du langage ACL (KIF, FIPA-SL, FIPA-CCL). Le vocabulaire commun concerne les définitions précisées dans une ontologie. Donc l'ontologie joue un rôle très important dans la communication dans un système multi-agents. La figure 1 représenté les composants d'un langage de communication entre les agents [55].

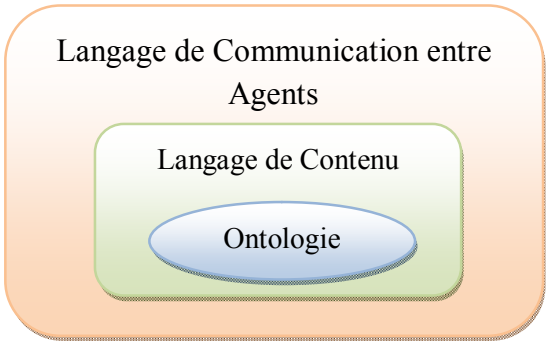


Figure 1. Modèle des Langages de Communication entre Agents [55].

Pour que le système multi-agent garantir une communication fiable, il est nécessaire d'établir des standards de communication incluant des protocoles de communication, les langages de communication et des ontologies. Dans les dernières années, des standards de communication d'agents tels que KQML (Knowledge Query and Manipulation Language) et FIPA-ACL (FIPA Agent Communication Language) sont développés. Des plateformes pour la communication des agents distribués tel que JADE et Jini de Sun Microsystems, et des langages de description telle que KIF et FIPA-SL (Content Language Specification) sont aussi créés [56].

11.L'APPROCHE AGENTS ET L'APPROCHE COMPOSANT

Les systèmes multi-agents proposent un ensemble de concepts, modèles et outils pour modéliser, développer des systèmes coopératifs, complexes, hétérogènes, évolutifs, décentralisés et ouverts pour la simulation, la résolution ou l'ingénierie des systèmes. Ils ont acquis un impact important dans le domaine de l'intelligence artificielle et des systèmes distribués. Ils ont été utilisés avec succès dans la supervision, le commerce électronique, les services Web, etc. Parallèlement, dans le domaine du génie logiciel, les composants logiciels font l'objet de propositions variées, qualifiées d'industrielles ou plus académiques. Les applications concernent plus particulièrement le domaine du génie logiciel : la programmation, la vérification ou les infrastructures logicielles. Des applications réussies ont

été conduites dans le domaine des lignes de produit multimédia, du commerce électronique, des systèmes d'exploitation, des télécommunications, etc.

Des besoins croisés émergent actuellement comme l'utilisation des composants pour les agents. Il s'agit d'utiliser la notion de composant pour aider à la conception, la construction et le déploiement de systèmes multi-agents, modulaires et réutilisables. Inversement d'autres tentatives essaient de transposer des propriétés des agents vers les composants pour concevoir les applications réparties du futur. Dans ce cas il s'agit de donner plus d'autonomie aux composants de l'application : capacités d'adaptation, de prise de décision, d'auto-assemblage (incluant le match-making), de coordination (avec d'autres composants), et donc de s'inspirer de concepts (autonomie, adaptation, coordination) des systèmes multi-agents. Comme on le voit avec les deux tendances précédentes il existe un champ d'investigation à la frontière des multi-agents et des composants [57].

Des travaux relativement récents se sont intéressés à la construction d'agents par assemblage de composants [58, 59]. La programmation par composants est en effet intéressante pour ce type de développement car les agents présentent souvent des capacités proches d'une application à une autre (communication, perception, planification,...). Dans ce cas, la construction d'un agent consiste à assembler des composants pré-existants qui sont chacun chargé d'implémenter une partie bien spécifique du comportement de l'agent. En d'autres termes, un agent est une entité logicielle contenant un assemblage de composants qui matérialise son comportement. Le développement d'applications multi-agents bénéficierait grandement de l'utilisation d'une bibliothèque de composants implantant ces capacités partagées. Plusieurs environnements de développement ont ainsi adopté une approche par composants.

Parmi les derniers travaux les plus récents l'article de Grondin et al [58] qui propose de construire des agents avec utilisation de modèle MADCAR1. MADCAR est un modèle abstrait qui a pour objectif la reconfiguration dynamique et automatique d'applications à base de composants.

12. LES AGENTS ET LES SERVICES WEB

On assiste aujourd'hui à un rapprochement entre les recherches en SMA et les services Web. Ce rapprochement se manifeste sous différents aspects. Il ya plusieurs travaux traitent les agents et les services Web des différentes côté soit de point de vue d'intégration ou de point de vue des standards utilisés. On en distingue essentiellement deux catégories : d'une part l'utilisation des agents en tant que cadre conceptuel pour mettre en place des services Web sophistiqués, ou encore des outils de planification et composition de services Web. D'autre part l'utilisation des services Web comme un cadre architectural et/ou technologique pour mettre en place des systèmes multi-agents ou des agents accessibles à travers Internet. Il y a deux grandes classes de travaux :

- La première catégorie correspond à l'utilisation des agents comme modèle conceptuel pour développer des services Web effectuant des tâches sophistiquées. Dans [65], une implémentation des services Web suivant un modèle agent est proposée afin de faciliter la gestion des transactions entre services Web commerciaux. Le travail proposé dans [66] utilise un ensemble de primitives de communication empruntés des langages de communication agent afin de mettre en place des services Web qui présentent un comportement d'interaction dynamique (défini en cours d'interaction). Il présente un ensemble de modèles de conversation pour introduire la négociation ainsi que des exigences à respecter dans l'invocation des services Web.
- La deuxième catégorie de travaux concerne l'utilisation des SMAs pour mettre en place des systèmes de composition ou de planification automatique et semi-automatique de services Web [18, 67, 68]. Pour le dernier travail, les aspects sémantiques prennent une place importante où il se base essentiellement sur des services Web qui définissent leurs sémantiques utilisant des ontologies définie par DAML-S. DAML-S (un langage de description sémantique des services) offre une spécification qui permet de décrire les services par rapport à un référentiel sémantique (ontologies) [69,70]. Les services Web comme environnement technologique et architectural pour les systèmes multi-agents. Les travaux sur l'adaptation du modèle technologique et architectural pour intégrer les systèmes multi-agents ne sont pas aussi présents que dans la première catégorie des travaux.

La plupart des travaux concernent essentiellement le cadre technologique des services Web c'est à dire les standards tels que l'utilisation de SOAP pour une implémentation des standards FIPA KQML et ACL.

13. CONCLUSION

Nous avons présenté dans ce chapitre une vision générale sur les systèmes multi-agents. Ces systèmes , qui ont porté des résolutions aux problèmes de l'IA classique , sont organisés dans des sociétés d'agents qui interagissent, communiquent et coopèrent entre eux pour accomplir une tâche bien déterminée. Afin d'augmenter les capacités des systèmes multi-agents et de faciliter la conception de ces derniers, les chercheurs appliquent les approches de génie logicielle telle que l'approche composant dans la conception et la composition des systèmes multi-agents.

Toutes les approches mentionnées sont complémentaires et couvèrent différents aspects d'un agent et les services Web. Nous essayons, dans ce projet, de présenter un modèle pour le développement des applications Web. Ce modèle prend les avantages à la fois des agents et les services Web sémantique tout en appuyant sur les ontologies qui fournissent un moyen standard de sémantique.

Chapitre 4


LA CONCEPTION DU
MODELE DE
DEVELOPPEMENT
DES APPLICATIONS
WEB

1. INTRODUCTION

Face à la difficulté de maîtriser le développement des logiciels, il est apparu une nouvelle tendance pour la construction de logiciels. *C'est le développement des logiciels en tant que services.* La définition du logiciel comme un ensemble des services a prouvé son efficacité [72]. Cette approche repose sur la réorganisation des applications en des ensembles fonctionnels appelés services. Elle est conçue principalement pour améliorer la rapidité ainsi que la productivité des développements. Elle est basée principalement sur le principe de réutilisation des composants représentés comme des services. D'un autre côté, qui nous intéresse, cette approche convient parfaitement au développement des applications Web. En effet, le but principal des applications Web est de rendre des services aux utilisateurs. Donc l'orientation service est une caractéristique naturelle des applications Web [72].

Dans ce chapitre nous allons décrire notre contribution qui est la proposition d'un modèle pour le développement des applications Web utilisant l'approche basée service. Ce modèle sert à construire des applications Web à partir de la composition des services existants en utilisant l'approche agent, les ontologies et les services Web sémantiques. Pour mieux présenter ce modèle, nous allons présenter les différents composants nécessaires tels que les ontologies, les services Web sémantiques et les agents. Enfin, nous allons fournir une architecture agent d'un système de construction des applications Web à base de service.

2. PROPOSITION

Un modèle servant au partage des connaissances pour le développement, le déploiement et l'évolution des applications Web en utilisant une approche basée service. Nous avons suivi l'approche orientée service pour la construction des applications Web. Cette approche sert à construire une application Web à partir d'une composition de services existants. Notre système a en entrée une demande d'un service globale de l'utilisateur et comme résultat une application Web basé service qui assure le service globale demandé. En effet, l'application Web finale est une composition des services existants.

Les aspects importants que notre modèle tente à les prendre en compte sont la réutilisabilité par l'utilisation des services existants, l'extensibilité par la possibilité d'ajout de

nouveaux services. Ce qui a comme avantage principal la minimisation de temps de développement.

Ce modèle intègre l'approche services Web sémantique, l'approche agent et les ontologies pour construire un environnement de développent des applications Web. Nous avons utilisé les services Web sémantique afin d'avoir un outil puissant garantissant la côté sémantique des applications. Les services Web sémantique permettent une spécification sémantique du domaine tel que les termes et les concepts d'intérêts, les relations entre ceux-ci et les caractéristiques de domaine à travers les ontologies. Les agents sont les acteurs dans notre modèle et ils ont des tâches spécifiques. Ces agents utilisent les services Web et les ontologies d'une manière complémentaire et dans un chemin de coopération pour réaliser l'objectif global.

3. LE WEB ET LES ONTOLOGIES

L'émergence d'une tentative appelée Web Sémantique dans le Web a venu pour résoudre certaine limitation de Web traditionnel. A l'inverse du Web traditionnel, qui est d'abord conçu pour l'interprétation et l'utilisation humaine, le Web Sémantique est une vision d'un Web non-ambigüe et compréhensible par des utilisateurs logiciel. Le Web sémantique reste entièrement fondé sur le Web traditionnel. Il reste toujours un moyen pour publier et consulter des pages Web, mais avec le Web sémantique, cela se fait via des documents contenant non seulement des textes en langage naturel, mais aussi des informations pouvant être traitées automatiquement par des agents logiciels. Ceci est réalisé par l'annotation du contenu du Web par la description de propriétés et de relations. Cette annotation est réalisée avec un langage raisonnablement expressif avec une sémantique bien définie permettant le partage et la réutilisation des données à travers différentes applications.

Le consortium W3C a défini un cadre général pour ce type de descriptions qui se base sur RDF (Ressource Description Framework) pour la représentation de connaissance et récemment sur OWL (Ontology Web Language) pour la modélisation sémantique de connaissances [59].

3.1. ONTOLOGIE POUR LE WEB SEMANTIQUE

Nous retiendrons donc qu'une ontologie traduit un consensus explicite sur la formalisation des connaissances d'un domaine afin de faciliter le partage et la réutilisation de ces connaissances par les membres d'une communauté ou par des agents logiciels. C'est dans cette optique que les ontologies se présentent comme un pilier du Web sémantique, car elles permettent de faire communiquer les hommes et les machines en utilisant la sémantique partagée par les différents acteurs du Web et en décrivant ses ressources [52]. Le Web sémantique et l'utilisation des ontologies permettent :

- des possibilités de recherche améliorées
- une automatisation des tâches possibles bien plus importante
- une interopérabilité des systèmes.

En effet, les ontologies ont des capacités de recherche puissantes vu leur architecture dédiée. Les recherches et la navigation prennent en compte les liens unissant des éléments d'informations et les contraintes définies sur ces éléments. De plus, chaque élément des documents étant représenté à l'aide d'une ontologie, il devient très facile d'automatiser des tâches définies à partir des données de cet élément. Enfin, les ontologies proposent une modélisation des informations totalement indépendante de l'application dans laquelle se trouvent ces informations [51].

Pour le Web une ontologie décrit et définit de façon formelle les relations entre les termes [72]. Ce type d'ontologie possède une taxonomie et un ensemble de règles d'inférence. La taxonomie définit des classes d'objets et les relations entre eux. On peut exprimer ces relations en attribuant des propriétés aux classes et en permettant à des sous classes d'hériter de leurs propriétés [63].

3.2. LANGAGE D'ONTOLOGIES WEB (OWL)

Bien que les premiers langages utilisés pour le développement d'outils et d'ontologies n'aient pas été définis pour être compatibles avec le Web, OWL (Web Ontonlogy Language) est un langage d'ontologies dédié pour Web. En effet, OWL a un vocabulaire XML basé sur RDF et permettant de définir des ontologies Web structurées. La production des ontologies par des standards augmente l'intégration et facilite l'interopérabilité entre les applications qui

utilisent ces ontologies. De plus, OWL facilite l'interprétation des informations du Web par des applications destinées à traiter ces informations et non pas seulement à leur représentation [51].

OWL est un langage qui étend RDFSchema en utilisant des notions qui y ont été définies (resource, class, relation, type, subClassOf, range,domain, ...) et en permettant, en plus, la création de relations complexes entre différentes classes et la définition de contraintes plus précises sur des classes et des propriétés. [51]

OWL est considéré par W3C comme une langue d'ontologie standard. Il a non seulement la capacité de décrire les concepts dans un domaine mais aussi d'un ensemble plus riche d'opérateurs, donc ces concepts bien définis et bien décrits. On peut construire des concepts complexes en basant sur des concepts simples. En outre, on peut vérifier si tous les rapports et les définitions dans l'ontologie sont conformés et identifier quels concepts s'adaptent et sous quelles définitions. Donc, on peut maintenir la hiérarchie correctement entre les classes.

OWL peut faire un compromis entre son pouvoir expressif et son pouvoir de raisonnement parce qu'il fournit des sous langages de plus en plus conçu expressifs. Donc, OWL permet d'augmenter le sens du vocabulaire prédéfini dans une ontologie. On peut définir chaque sous langage grâce à sa expression [63]. OWL est défini en trois sous langages :

- OWL Lite : destiné aux utilisateurs voulant définir une hiérarchie et des contraintes simples
- OWL DL : destiné aux utilisateurs voulant définir des restrictions plus générales tout en respectant les logiques de description
- OWL Full : destiné aux utilisateurs voulant définir des restrictions générales sans aucune contrainte à respecter pour le faire.

Les spécifications RDF et OWL constituent une partie du Web Sémantique qui est basée sur les concepts suivants :

- un schéma de nommage global grâce aux URIs
- une syntaxe standard pour décrire une ressource : RDF
- une façon standard pour décrire les propriétés de cette ressource : RDF Schema
- un moyen standard pour décrire les relations entre ressources : OWL [51].

Parmi les Objectifs de OWL, on peut citer :

- Offrir un langage standard pour définir des ontologies Web basé sur RDFSchéma.
- Etendre les constructions de base pour améliorer : l'interopérabilité, le raisonnement et les évolutions.
- Inspirer de DAML (Darpa) + OIL (EEC) XML ➔ RDF ➔ RDFShema ➔ OWL.

4. LES SERVICES WEB SEMANTIQUE

4.1. LES LIMITES DES SERVICES WEB

Les perspectives d'automatisation des processus liés aux web services (découverte, composition et invocation) sont pour l'instant limitées par les formalismes de description employés. Les informations qui y sont décrites sont insuffisantes pour qu'un agent logiciel puisse 'raisonner' et ainsi comprendre si le service correspond à ce qu'il cherche et comment l'utiliser. Par exemple les documents WSDL servent essentiellement aux développeurs. L'exploitation logicielle de tels documents se limite souvent à la génération de squelettes de code source pour employer le service. Ceci n'est pas suffisant.

De même, les mécanismes disponibles pour rechercher des services *via* des annuaires UDDI sont relativement primitifs. En effet, ils sont basés sur des paires *(clé, valeur)* qui limitent fortement la qualité des recherches.

Nous voyons donc que les limites actuelles sont essentiellement d'ordre sémantique. WSDL offre un excellent formalisme pour la partie statique des services. En revanche, la description de la partie dynamique des services est plus problématique : plusieurs standards sont émergent et ne prennent pas en compte tous les besoins pour qu'un agent logiciel puisse raisonner. [49] [21]

La commute de Web sémantique a amélioré les standard de technologie des services Web par l'ajout d'une couche sémantique à la description de service afin d'automatiser la découverte (recherche des services sur le Web), la composition, le contrôle et l'exécution. L'automatisation des tâches est très désirable et, ainsi un très grand nombre de projets de recherche adoptent la technologie de Web services sémantique dans les différents domaines d'applications (les grilles bioinformatiques, les méthodes de résolution des problèmes).

4.2. DEFINITION DES SERVICES WEB SEMANTIQUES

On sait qu'une ontologie est un ensemble de concepts, de leurs propriétés, et des relations entre eux. Les ontologies fournissent des techniques pour exprimer la sémantique d'une façon bien définie [71]. On a besoin d'ontologies pour ajouter la sémantique aux services Web. Mais on sait que la modélisation avec des ontologies n'est pas tout à fait différente de celle dans le contexte de la technologie de programmation. Il y a plusieurs analogies avec le modèle logiciel utilisant des classes, de la transmission, des propriétés, ... etc. [63].

Les « Web Services Sémantiques » visent à enrichir la représentation des Web Services avec des connaissances relatives à l'objectif du service, au processus mis en œuvre par le service et à ses contraintes d'utilisation. Ces connaissances peuvent alors être exploitées par des agents pour automatiser la recherche**,** l'invocation, la composition et l'exécution des Web Services. Les langages de description des « Web Services Sémantiques » assurent le marquage sémantique des services Web et rendent explicite la connaissance nécessaire à la sélection et à la composition [50].

Les services Web Sémantiques seront capables d'interagir avec n'importe quel autre service, autant que ces derniers décrivent leurs fonctions en des termes compatibles et partagés du Web sémantique (Interopérabilité). De plus, la découverte et l'utilisation des services sémantiques résistent aux différents changements de l'interface puisque l'agent réinterprète les données à chaque utilisation. Ainsi, l'agent serait en mesure de proposer le (ou les) meilleur(s) choix à utiliser qui peuvent accepter ou non le changer d'état de l'environnement.

Mais tout n'est pas toujours rose dans le monde sémantique. En effet, le Web sémantique est souvent critiqué lorsqu'il s'agit de la confiance. Comment faire confiance à une machine. Un service transactionnel présentement interagit avec un humain. Comment la machine pourrait-elle prendre le rôle de l'humain ? Il y a aussi la question de confiance lorsqu'il s'agit du service lui-même. Comment comparer deux services sémantiquement identiques ? etc. Il reste beaucoup à faire.

4.3. LANGUAGE WEB D'ONTOLOGIES SERVICE (OWL-S)

OWL-S est une ontologie et un langage pour les services Web développés dans le cadre du projet DAML1. L'OWL-S est basé sur OWL qui permet de décrire des ontologies. Ainsi OWL-S est une ontologie OWL particulière. OWL étant le successeur de DAML+OIL, OWL-S succède aux travaux antérieurs de DAML-S qui étaient basés sur DAML+OIL. Cette ontologie a pour objectif de décrire de façon non-ambigüe les services Web de façon qu'un agent logiciel puisse exploiter automatiquement ces informations. Les bénéfices de l'emploi d'une ontologie sont multiples, mais le premier qui peut venir à l'esprit concerne l'amélioration de la recherche de services par rapport aux solutions existantes telles UDDI [48].

L'approche OWL-S propose une ontologie de services dans laquelle chaque service est décrit selon trois dimensions : le « **Profil** » du service qui précise les fonctionnalités du service, le « **Modèle** » du service qui décrit le processus mis en œuvre par le service, et le « **Grounding** » du service qui spécifie en termes de messages les règles d'interaction avec le service. Cette ontologie de services enrichit WSDL et UDDI pour permettre un accès et une utilisation du service à partir de son contenu (objectif, messages….) plutôt que par des mots clés. [50, 56]. OWL-S s'applique à renseigner les Web Services de façon à pouvoir exécuter les tâches suivantes :

1. La découverte automatique des services Web
2. L'invocation automatique des services Web.
3. La composition et l'interopérabilité des services Web.
4. La surveillance automatisée de l'exécution des services Web.

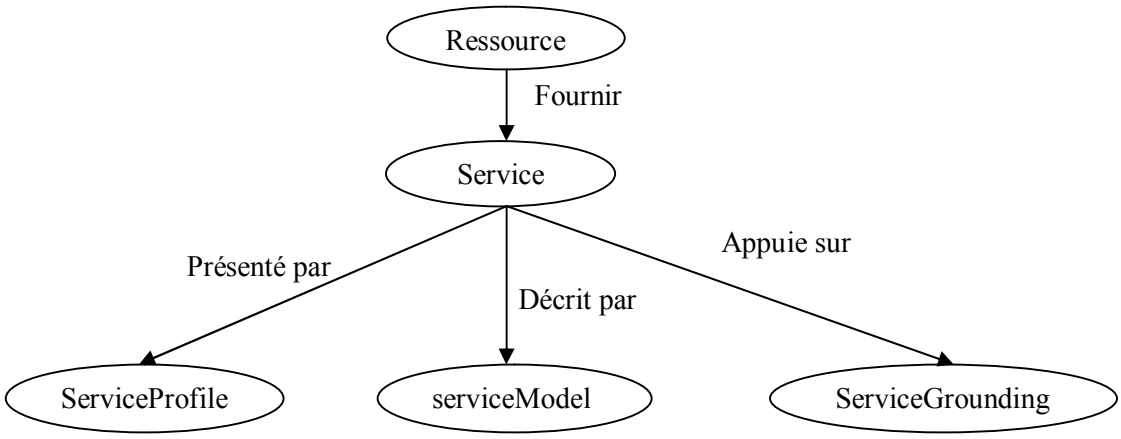


Figure 1. Aperçu général de l'ontologie OWL-S [21].

5. MODELE CONCEPTUEL

On rappel que notre modèle qu'on va présenter a pour but global de créer des applications Web à partir des requêtes de l'utilisateur tout en utilisant l'Internet comme un moyen collaborateur dans la réalisation de ce système.

Ce modèle se présente sous forme d'un environnement de développement des applications Web en suivant une démarche basée sur la composition des services. Cet environnement est dédié à la création d'applications Web à partir de requêtes des utilisateurs. La requête de l'utilisateur subit des traitements d'analyse de recherche s'il y a un service qui peut répondre à cette requête. Un autre type d'utilisation de notre environnement, l'ajout de nouveaux services. Une vue générale de notre système peut être schématisé comme suit :

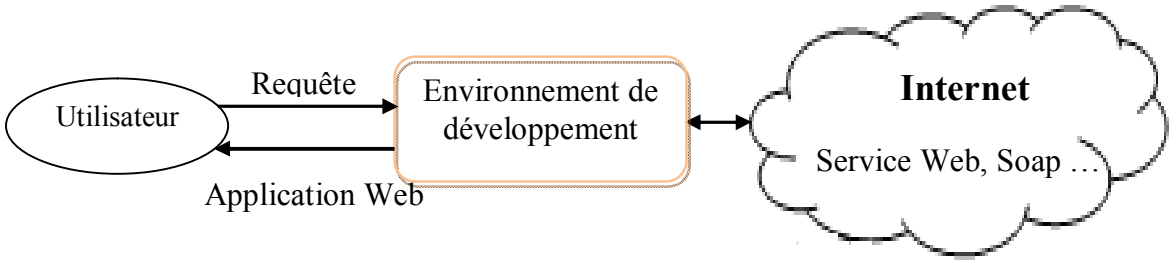


Figure 2. Vue générale de l'environnement de développement des Applications Web

.

Le modèle détaillé qui montre les grandes fonctions assurées par notre système se présente comme suit :

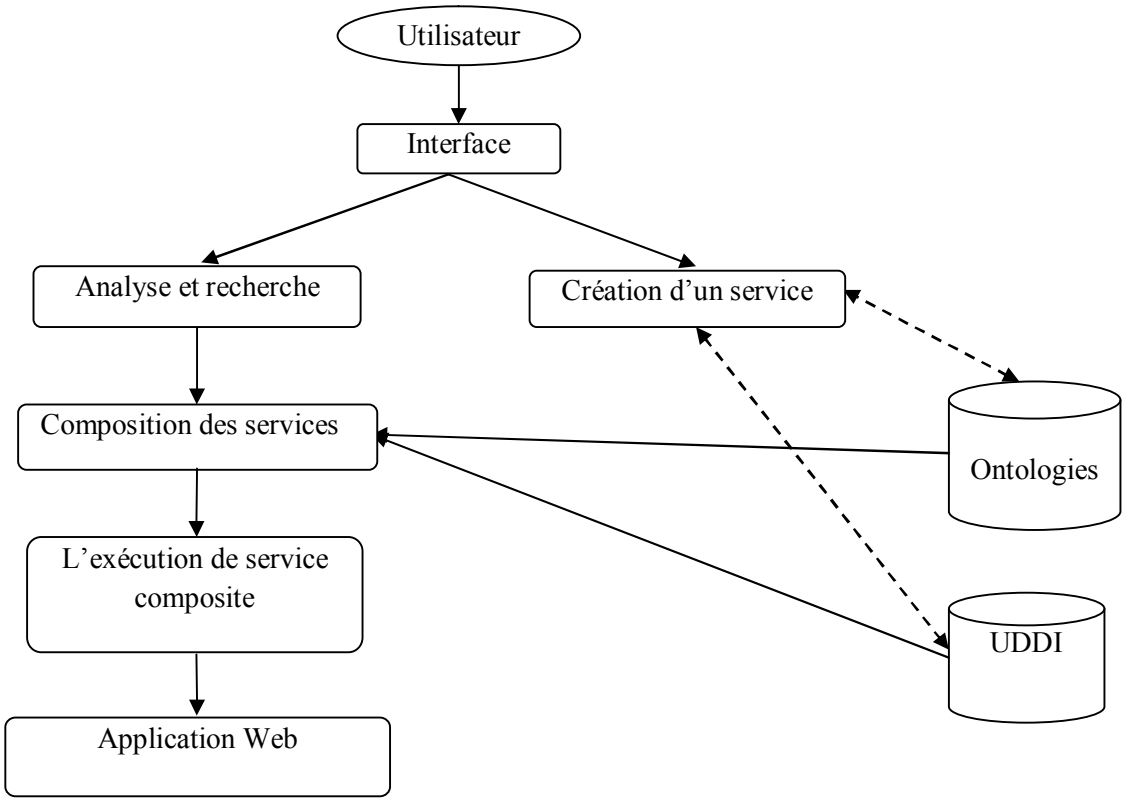


Figure 3. Modèle conceptuel du système**.**

Notre modèle doit vérifier certains critères à savoir :

- Elaborer un environnement convivial qui lui convient ; simple et facile à gérer.
- Présenter une interface simple et claire lui permet d'éditer ses requêtes.
- Disposer d'un outil graphique lui permet de construire ses requêtes
- Présenter l'application Web finale.
- Présenter une interface lui permet de créer et insérer des nouveaux services dans la base des services.

6. ARCHITECTURE DU SYSTEME

Pour que notre modèle soit complet et claire on doit lui associer une architecture. Les composants de cette architecture interagissent ensemble afin de réaliser les fonctions globales mentionnées précédemment. La figure 4 présente cette architecture tout en illustrant les relations entre ses composants. Les composants sont :

- les ontologies.
- les services Web sémantique.

- les agents : on a quatre agents :
  - ✓ Agent créateur
  - ✓ Agent analyseur
  - ✓ Agent compositeur
  - ✓ Agent chercheur.

L'ontologie est une sorte d'approche pour la représentation des connaissances, qui est utilisé pour décrire l'information dans la requête et annoter les services Web. Dans notre modèle on va utiliser trois types d'ontologies, l'ontologie des services Web, l'ontologie de domaine et l'ontologie de processus. Les requêtes des utilisateurs peuvent être des requêtes d'un service global ou une requête d'ajout d'un nouveau service. Notre système multi-agents est constitué d'un ensemble d'agents. L'agent analyseur reçoit la requête d'un service global de l'utilisateur puis la décompose en sous-requêtes. Le groupe d'agents chercheurs où chaque agent chercheur prend une seule sous-requête à la fois et trouve le service qui répond à celle-ci. L'agent compositeur rassemble les services trouvés par le groupe d'agents chercheurs sous forme d'un service global qui représente l'application Web demandée et qui sera retourné à l'utilisateur. L'agent créateur est le responsable de traitement de la requête d'ajout des nouveaux services.

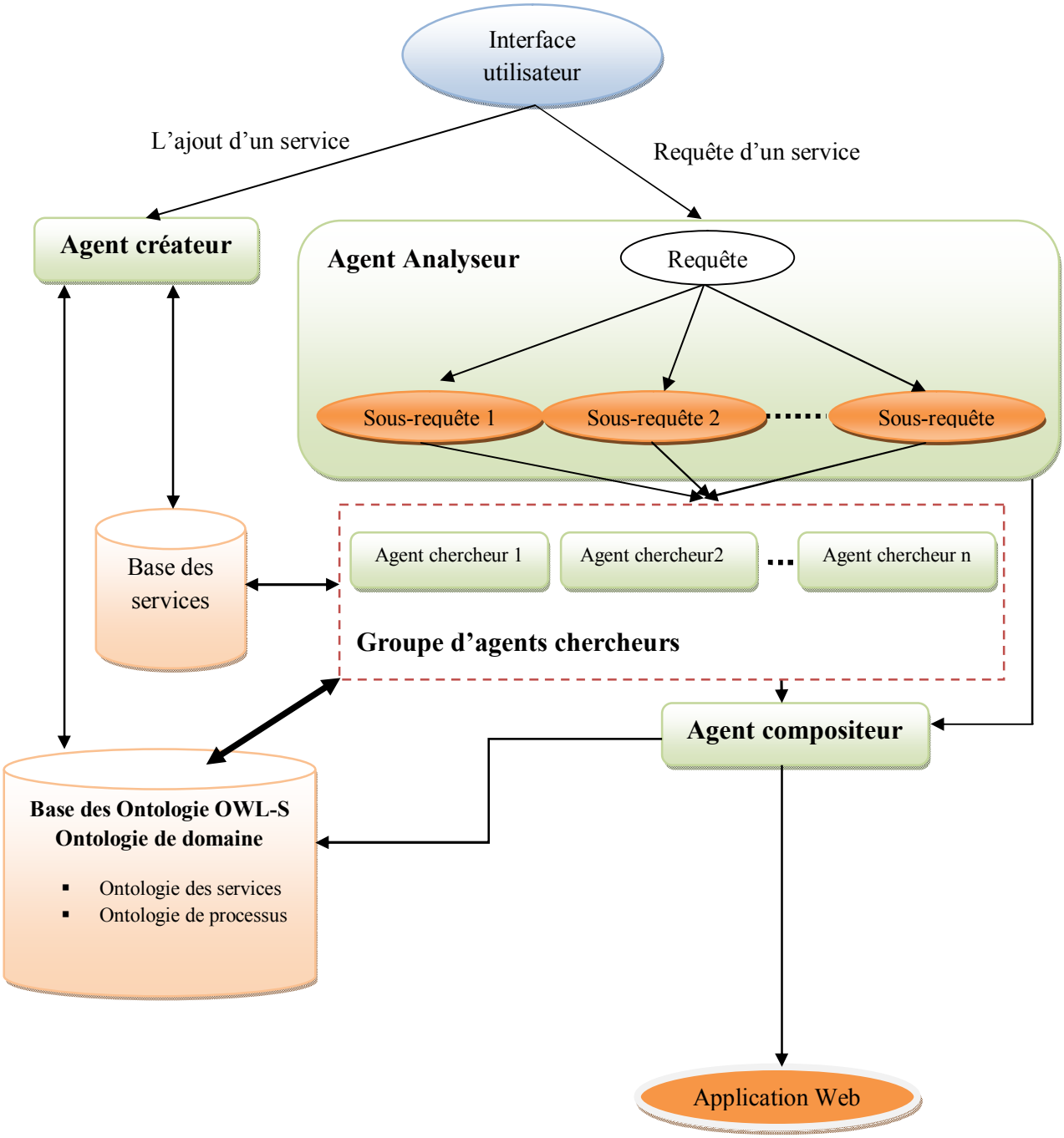


Figure 4. Architecture globale du système.

7.1. PROTOCOLE DE COMMUNICATION ET FONCTIONNEMENT

Dans cette section on va présenter le protocole de communication et fonctionnement de notre système de développement des applications Web. Nous allons décrire ce protocole dans une séquence d'étapes suivies :

- Le système va présenter à l'utilisateur une interface graphique pour qu'il puisse décrire son application demandée suivant une spécification graphique qui sera interprétée par l'agent analyseur à une requête d'entrée.
- Celle-ci doit être analysée et décomposée également par l'agent analyseur en un ensemble des sous-requêtes organisées selon la logique issu la spécification de l'utilisateur dans un plan descriptif de l'application à générer et qui sera utilisé pendant l'opération de composition.
- Pour chaque sous-requête, l'agent analyseur va activer un des agents chercheurs.
- Apres que l'agent chercheur reçoit sa sous-requête, il va chercher s'il y a un service dans la base des services Web du système répondant à cette sous-requête.
- Après que les agents chercheurs trouvent les services correspondant aux sous-requêtes, chaque agent retourne la description de service trouvé (URL, document WSDL, conditions d'utilisation, etc.) à l'agent compositeur qui sera utilisée dans la génération de l'application finale.
- Si l'un des agents chercheur ne trouvent pas un service qui correspond à sa requête, il cherche dans l'ontologie de services de Web. S'il ne trouve pas, un message sera envoyé à l'utilisateur pour lui informer de la nécessité de développement de ce service élémentaire ou son achat.
- L'agent compositeur compose le programme à partir des descriptions des services reçus des agents chercheurs pour générer l'application finale. L'application générée sera retournée à l'utilisateur comme une réponse à sa requête.
- Dans le cas où l'utilisateur veut ajouter un nouveau service, l'agent créateur va lui fournir de l'aide pour le créer et faire les mises à jour nécessaires pour l'insérer dans la base des services.

7.2. ONTOLOGIES DE DOMAINE

Dans notre système nous utilisons OWL-S pour présenter les ontologies. Les ontologies du domaine qui sont spécialisées pour la description des paramètres des services. L'ontologie

de domaine toujours inclut des classes reliées, des propriétés, des relations entre les entités et des règles d'inférence, etc.

Pour développer une ontologie d'un domaine d'application particulier, il y a différentes méthodes et outils. Une de ces méthodes emprunte l'idée de développement des classes utilisées dans la méthode orienté objet. Le développement des classes suivant les méthodes orienté objet utilise les noms pour identifier les classes dans le domaine d'application, les mots descriptifs peuvent être utilisés pour identifier les attributs des classes et les verbes peuvent être utilisés pour représenter le comportement des classes (les méthodes). Ainsi, dans la définition de l'ontologie du domaine, cette méthode s'intéresse à la structure des classes et les relations entre ces classes.

Comme les ontologies du domaine, l'ontologie du service Web décrit les hiérarchies des classes de services et leurs sous-classes. Ces sous-classees de service héritent les propriétés et les fonctionnalités de leurs supers classes [63]. L'ontologie de processus nécessite une collection des services connectés l'un avec l'autre si leurs interfaces sont sémantiquement assortis [64].

7.3. LE SYSTEME MULTI-AGENTS

Dans cette partie on va présenter l'architecture des agents formant notre modèle et leurs composants internes tout en illustrant les messages échangés entre eux.

7.3.1. AGENT ANALYSEUR

La requête exprime les besoins de l'utilisateur. Afin de réaliser le but effectivement, la requête est toujours décomposée en sous-requêtes puisque ces sous-requêtes sont simple à l'implémentions et requièrent moins de ressources. Cette décomposition d'une requête peut être exprimée et organisée comme un graphe de ET-OU. Dans ce graphe, le ET signifie que la satisfaction de toutes les sous-requêtes d'une requête est une condition pour la satisfaction de la cet requête. Alors que le OU indique que la satisfaction d'une des sous-requêtes est une condition suffisante pour la satisfaction de requête.

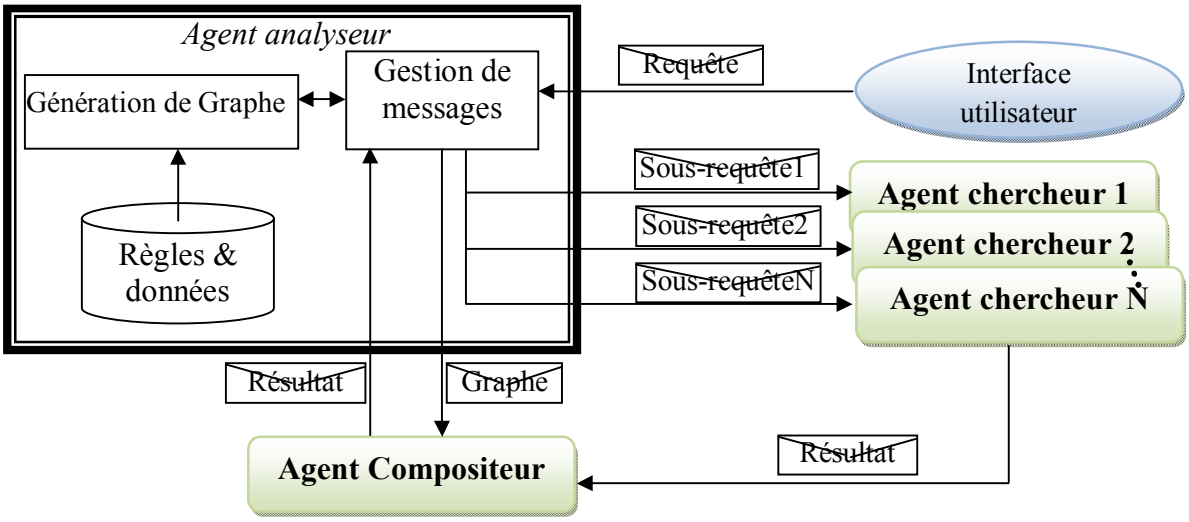


Figure 5. Architecture de l'agent analyseur.

Les informations dans la requête utilisateur représentent les informations fonctionnelles et les informations non-fonctionnelles qu'offre un domaine d'application particulier. L'agent analyseur reçoit la requête contenant la spécification de l'application de l'utilisateur et en génère un graphe, organisé sémantiquement, qui sera utilisé comme un plan d'exécution et qui décrit les sous-requêtes, leurs positions et les relations entre elles en les considérant comme des nœuds de ce graphe. Nous pouvons dire que ce graphe représente le squelette de l'application à générer. L'agent analyseur doit invoquer l'agent compositeur en le munissant de ce graphe après l'invocation de groupe des agents chercheurs pour collaborer et satisfaire les sous-requêtes. Chaque sous-requête contient, à son tour, les informations qui décrivent le service élémentaire désiré. Pour exécuter cette sous-requête, les agents chercheurs ont besoin d'utiliser ce type d'informations pour lier cette requête au service correspondant. Pour cette raison que l'information dans la requête doit être décrite sémantiquement. Dans ce modèle, nous utilisons l'ontologie pour décrire les informations et les services web. L'information dans la requête est sémantiquement décrite par les propriétés suivantes :

- La capacité fonctionnelle concerne la fonction fournie par l'information dans la requête.
- Les propriétés non-fonctionnelles concernent la qualité de service, emplacement, etc.
- Les effets concernant quelques états et conditions doivent être satisfaits pour satisfaire la requête.

Ainsi, une partie de la sous-requête, qui est le profil, inclut la description du service et l'interface du service attendu, auquel on définit les paramètres de sorties, les contraintes de sorties, les paramètres d'entrées et les contraintes d'entrées. Les contraintes d'entrée spécifient les exigences de l'utilisateur sur les entrées, tel que les propriétés des paramètres de sortie [64].

7.3.2. LE GROUPE D'AGENTS CHERCHEURS

Dans notre modèle il y a un groupe d'agents chercheurs. Le nombre d'agents dans ce groupe dépend du nombre probable de sous-requêtes. Il peut être fixé par l'expérience de telle sorte qui garantit la performance du système. Chaque agent est associe à une seule sous-requête.

Le rôle de l'agent chercheur est de trouver le service convenable à la sous-requête. Cette recherche se fait premièrement dans la base des services par la comparaison des paramètres d'entrées, des contraintes d'entrées, des paramètres de sorties et des contraintes de sorties définies dans la sous-requête avec celles des services. En plus, il y a un autre test qui doit être effectué. C'est le test des contraintes non fonctionnelles. Dans le cas où le service demandé n'existe pas dans la base des services on va parcourir l'ontologie de service de Web pour trouver un service convenable par le test des paramètres comme on a fait dans le cas de base des services locale.

Dans l'étape suivante, si un agent chercheur trouve plusieurs services, il sélectionne un service qui vérifie les qualités des services demandés par l'utilisateur. La sélection du service doit être conforme aux standards de qualité de service (QoS).

Les fournisseurs des services ordonnent les contraintes de QoS, appelées des contraintes objectives (visées), pour la sélection de service. L'utilisateur aussi donne ses contraintes considérées comme des contraintes subjectives. Ainsi, le processus de sélection des services nécessite deux étapes :

- La correspondance des contraintes subjectives des services.
- L'évaluation des contraintes objectives des services.

En fonction des contraintes *subjective*, l'agent peut déterminer approximativement si un service répond à ses besoins, ensuite il va vérifier les contraintes *objectives* du service.

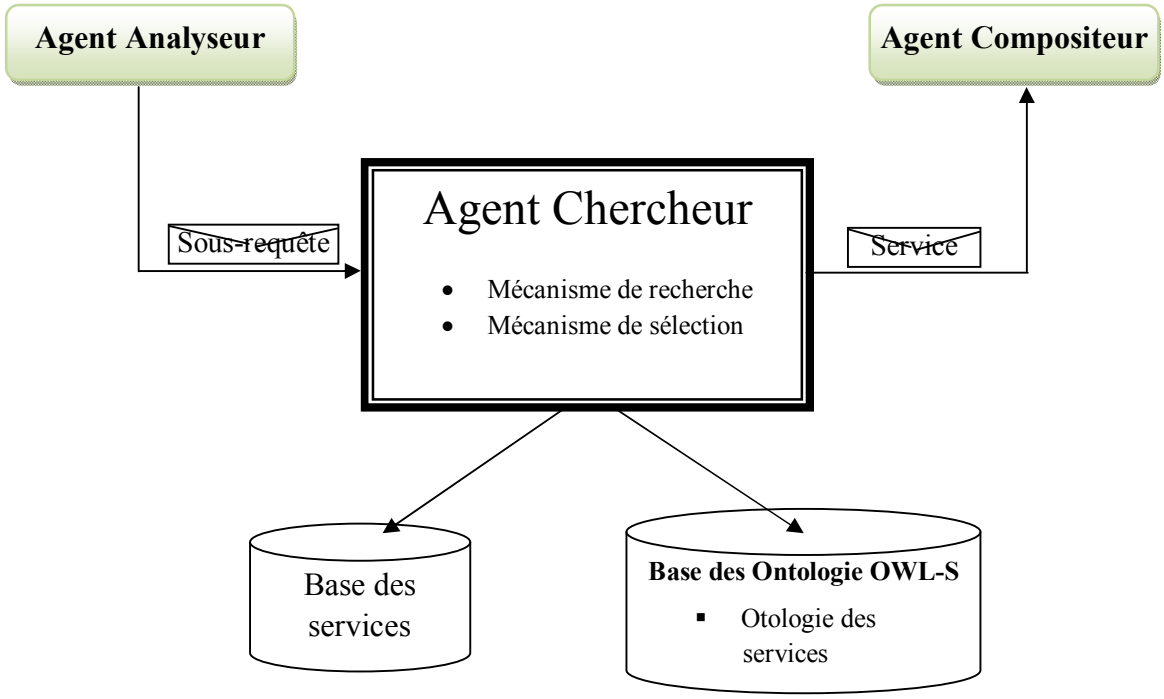


Figure 6. Architecture d'un agent chercheur.

7.3.3. AGENT COMPOSITEUR

Le but de l'agent compositeur est de produire un plan d'exécution d'actions basé sur les buts de l'utilisateur et l'ensemble de services disponibles. Dans notre modèle, la requête de l'utilisateur est exprimée comme un but de composition des services Web sémantique. On va utiliser le modèle du processus de OWL-S pour faire cette composition. Le OWL-S offre la possibilité de la composition automatique des services Web mais cette composition manque de sémantique. Il y a plusieurs travaux sur la sémantique de OWL-S [78, 81, 62, 85]. Parmi ces travaux, on adopte l'approche de composition des services Web sémantique au niveau de processus [53].

La plus part des travaux sur la composition de services Web sémantique étaient focalisés jusqu'à maintenant sur le problème de la composition des services au niveau fonctionnel, c.-à-d., la composition des services, qui sont considérés comme des composants "atomique" (ceux-ci sont décrit en termes de leurs entrées, leurs sorties, leurs prés-conditions, et leurs effets), peut être exécutée dans une simple forme de requête-réponse. On a besoin de prendre en compte les cas réels où les services composants ne peuvent être exécutés dans une seule étape de requête-réponse. Une des solutions de ce problème est la composition des services Web sémantique au niveau du processus.

Le problème de la génération automatique de services Web composés peut être résolu à l'aide de l'interaction directe avec les services composants. Les services composants sont des processus d'un ensemble d'états, et ils requièrent un protocole d'interaction qui peut nécessiter différentes étapes, séquences, condition, et itérations pour suivre l'exécution de ces processus.

On traite problème de la composition au niveau des modèles de processus OWL-S. Etant donné un ensemble de services disponibles sur le Web et leurs modèles de processus sont décrits en OWL-S, et étant donné un but de composition qui décrit un ensemble des requêtes désirées sur le comportement du service Web composé qui doit être généré. On génère automatiquement un service Web composé exécutable, il est aussi décrit en modèle de processus OWL-S. Le service Web composé, pendant l'exécution, interagit avec les services disponibles afin de satisfaire le but.

Théorie de la composition [53]

Notre but est de générer automatiquement un nouveau service *W* (appelé le *service composé*) qui interagisse avec un ensemble de services Web W_1,…, W_n (appelés les *services composants*) publiés dans le Web et satisfait un but d'une composition donnée. Plus spécifiquement, on commence de *n* descriptions de processus des services Web, c.-à-d., leurs modèles de processus décrit en OWL-S, et on transforme automatiquement chacun de ceux-ci en un *système de transition d'état*: $\sum_{w1}$,…, $\sum_{wn}$. Intuitivement, chaque $\sum_{wi}$ est une représentation compacte de tous les comportements, les évolutions possibles du service composant W_i.

Ensuite, on construit le STS (STS : *State Transition System*) parallèle $\sum_{\|}$ qui combine W_1, …,W_n et représente toutes les évolutions possibles de ces services composants, sans aucun contrôle, et l'interaction avec le service composé qui doit être généré. On formalise aussi les requêtes des services composés comme un but de composition, appelé ρ. Intuitivement, ρ décrit la fonctionnalité que le service composé doit avoir. Étant donné $\sum_{\|}$ et ρ, on génère automatiquement le STS $\sum_c$ qui contient le code du nouveau service. $\sum_c$ représente le service qui dynamiquement reçoit et envoie des messages depuis/vers les services composant W_1, …, W_n et se comporte selon la réponse reçu depuis W_1, …, W_n. $\sum_c$ tel que $\sum_c \blacktriangleright \sum_{\|}$ satisfait le but de la composition ρ, où $\sum_c \blacktriangleright \sum_{\|}$ représente touts les évolutions des services composants $\sum_{\|}$ alors qu'ils sont contrôlés par le service composé $\sum_c$. Puis le STS

$\sum_c$ est automatiquement transformé en un service web exécutable, c.-à-d., décrit en OWL-S comme un modèle de processus.

Des définitions formelles

La définition formelle du problème de la composition au niveau de processus des services Web est basée sur la notion de système de transition d'état STS. Les STSs sont des modèles généraux pour la définition des systèmes dynamique qui peuvent être dans différents *états* (quelques états sont marqués comme des *états initiales*) et peut se développer en un nouvel état comme un résultat de l'interprétation de quelques *actions*.

Les actions sont distinguées en *actions d'entrée*, qui représentent la réception des messages, *actions de sortie*, qui représentent les messages mis aux services externes, et une action spéciale τ, appelé *action interne*. L'action τ est utilisée pour représenter les évolutions internes qui sont invisible pour les services externes, c.-à-d., le fait que l'état du système peut se développer sans une production de sorties, et indépendamment de la réception des entrées. La *relation de transition* décrit la façon dont l'état peut se développer en se basant sur les entrées, les soties ou sur l'action interne τ. Enfin, une fonction étiquettage associe à chaque état un ensemble de propriétés *prop* qui maintienne dans l'état. Ces propriétés doivent être utilisées pour définir le but de la composition ρ.

Définition 1 : système de transition d'état (STS)

Un système de transition d'états $\sum$ est un tuple (S, S^0, I, O, R, L) ou :

- S l'ensemble des états.
- S^0 appartient à S. C'est l'ensemble des états initiaux.
- I est l'ensemble des actions d'entrées.
- O est l'ensemble des actions de sorties.
- $R \subset S \times (I \cup O \cup \{\}) \times S)$ est la relation de transition.
- $L : S \rightarrow 2^{prop}$ est la fonction d'étiquetage.

On assume que la boucle infinie de τ-transitions ne peut paraître dans le système (c.à.d. le service devait interagir avec l'environnement après un nombre fini des étapes). On assume aussi qu'il n'y a pas un état de l'origine d'une transition d'entrée et de sortie simultanément.

D'après le modèle formel, on distingue trois différents types d'actions. Les actions d'entrées I modélisent toutes les requêtes arrivants au processus et les informations qu'elles amènent. Les actions de sorties O représente les messages sortants. L'action τ est utilisée pour modeler les évolutions internes du processus, par exemple l'attribution et la prise des décisions. Finalement, l'ensemble des propretés *Prop* de STS sont des expressions de la forme <variable> = <valeur> ou <tableau> [idx_1,...,idx_n]=<valeur>, et la fonction d'étiquetage estévidente.

Définition 2 : le produit parallèle de deux STSs

On va définie formellement le produit parallèle $\sum_1 \| \sum_2$ de deux STSs $\sum_1$ et $\sum_2$. Il modélise le fait que les systèmes peuvent être développés indépendamment (aucune communication direct), il est utiliser pour générer $\sum_{\|}$ à partir des services Web composants $W_1, \ldots, W_n$. Formellement $\sum_{\|} = \sum_{W1} \| \ldots \| \sum_{W n}$.

Soit $\sum_1 = (S_1, S_1^0, I_1, O_1, R_1, L_1)$ et $\sum_2 = (S_2, S_2^0, I_2, O_2, R_2, L_2)$ deux STSs avec $(I_1 \cup O_1) \cap (I_2 \cup O_2) = \emptyset$. Le produit parallèle $\sum_1 \| \sum_2$ est définie comme suit :

$$\sum_1 \| \sum_2 = (S_1 \times S_2, S_1^0 \times S_2^0, I_1 \cup I_2, O_1 \cup O_2, R_1 \| R_2, L_1 \| L_2)$$

Où :

- $((S_1, S_2), a, (S_1`, S_2)) \in (R_1 \| R_2)$ si $(S_1, a, S_1`) \in R_1$;
- $((S_1, S_2), a, (S_1, S_2`)) \in (R_1 \| R_2)$ si $(S_2, a, S_2`) \in R_2$;
- $(L_1 \| L_2) = L_1(S_1) \cup L_2(S_2)$

Le problème de la composition automatique consiste en génération d'un STS $\sum_c$, qui dirige le STS $\sum_{\|}$ en satisfaisant des besoins de la composition ρ. On défini maintenant formellement le STS décrivant le comportement de STS $\sum$ quant il est contrôlé par $\sum_c$.

Définition 3 : le système contrôlé

Soit $\sum = (S, S^0, I, O, R, L)$ et $\sum_c = (S_c, S_c^0, I_c, O_c, R_c, L_c)$ deux STS ou $L_\emptyset(S_c) = \emptyset$ pour toutes $S_c \in S_c$. Le STS $\sum_c \blacktriangleright \sum$ décrivant le comportement de système $\sum$, quant il est contrôlé par $\sum_c$. Il est défini comme suit :

$$\sum_c \blacktriangleright \sum = (S_c \times S, S_c^0 \times S^0, I, O, R_c \blacktriangleright R, L).$$

Où :

- $((S_c , S), \tau,(S_c`, S)) \in (R_c \blacktriangleright R)$ si $(S_c ,\tau, S_c`) \in R_c$
- $((S_c ,S), \tau,(S_c, S)) \in (R_c \blacktriangleright R)$ si $(S_c ,\tau, S_c`) \in R_c$
- $((S_c, S), a, (S_c`,S`)) \in (R_c \blacktriangleright R)$, avec $a \neq \tau$, si $(S_c , a , S_c`) \in R_c$ et $(S, a , S`) \in R$

On remarque qu'on n'a besoin que les entrées de $\sum_c$ qui coïncident avec les sorties de $\sum$ et vice-versa. On remarque aussi que, bien que les systèmes sont connectés pour que les sorties d'un est associée aux entrées des autres, les transitions résultant de $R_c \blacktriangleright R$ sont étiqueté par les actions entrées/sortis. Ceci nous permet de distinguer entre les transitions correspond à τ-actions de $\sum_c$ ou $\sum$ et les transitions dérivant à partir de la communication entre $\sum_c$ et $\sum$. Finalement, On remarque que nous admettons que $\sum_c$ n'a pas des étiquettes associées aux états.

Le STS $\sum_c$ peut ne pas être s'adéquate pas pour contrôler le système $\sum$. Par conséquent, il faut garantir qu'à chaque fois que $\sum_c$ exécute une transition de sortie, $\sum$ peut l'accepter, et vice versa. On défini une condition de ce qu'un état s de $\sum$ est capable d'accepter un message. On assume que s peut accepter un message `a ` s'il y a au moins successeur s` de s dans $\sum$, accessible à partir de s à travers une chaîne de $\tau$-transitions, tel qu'on peut exécuter une transition d'entrée étiquetée par `a`. Inversement, si l'état n'a pas un tel successeur s`, et le message `a`a été envoyer à $\sum$, une situation impasse sera atteint. Dans la définition suivante, on dénote par τ-fermeture(s) l'ensemble des états accessible depuis s à travers une séquence de τ-transitions, et par τ-fermeture(s), avec $s \subseteq S$, l'union τ-fermeture(s) sur tous $s \in S$.

***Définition 4 : Le contrôleur de dégagement des impasses (*deadlock-free controller)**

Soit $\sum = (S, S^0, I, O, R, L)$ et $\sum_c = (S_c, S_c^0, I_c, O_c, R_c, L_\emptyset)$ deux STS. Le STS $\sum_c$ est le contrôleur de $\sum$. $\sum_c$ est ***un contrôleur de dégagement des impasses*** si chaque état de $(s_c, s) \in (S_c \times S)$ qui est accessible à partir de l'état initiale de $\sum_c \blacktriangleright \sum$ satisfaisait les conditions suivantes :

- Si $(s, a, s') \in R$ avec $a \in I$, il existe $s'_c \in \tau$-fermeture (s_c) tel que $(s_c', a, s_c'') \in R$ pour un certains $s_c'' \in S_c$.
- Si $(s_c, a, s_c') \in R_c$ avec $a \in O$, il existe a quelque $s' \in \tau$-fermeture(s) tel que $(s', a, s'') \in R$ un certains $s'' \in S$.

La tâche de la composition automatique besoin de générer ***le contrôleur de dégagement des impasses*** $\sum_c$ qui garantié la satisfaction du but de la composition ρ. Ceci est formalisé par le besoin que le système contrôleur $\sum_c \blacktriangleright \sum_{\|}$ doit satisfaire ρ écrit $\sum_c \blacktriangleright \sum_{\|} \vDash \rho$ La définition de ce besoin est technique: Il dépend de la définition de l'exécution de $\sum_c \blacktriangleright \sum_{\|}$, et ceux-ci doit être définie en prend en compte la règle spéciale de τ-actions, qui décrit l'évolution interne de système.

Définition 5 : le problème de la composition

Soit $\sum_1, \sum_2, \ldots \sum_n$ un ensemble des STSs et soit ρ le but de la composition. Le problème de la composition des $\sum_1, \sum_2, \ldots \sum_n$ et ρ devient le problème de trouver le contrôleur $\sum_c$ ***le contrôleur de dégagement des impasses*** tel que $\sum_c \blacktriangleright (\sum_1 \| \sum_2 \ldots \| \sum_n) \vDash \rho$.

La composition au niveau de processus des services Web est obtenue en quatre étapes :

1ème étape : Des services Web au système de transition d'état : La description au niveau de processus des services Web composants disponibles sont transformées en STSs.

2ème étape : Expression du but de la composition : Cette étape consiste à formalisation du but ρ qui définisse la fonctionnalité que le service Web composé doit fournir.

3ème étape : Synthèse de la composition : Durant cette étape, le STS qui implémente le service Web composé est automatiquement généré en commençant par les STSs construits dans l'étape 1 et de but spécifié dans l'étape 2.

4ème étape : Le déploiement et l'exécution du service composé : Dans cette étape, le STS généré à l'étape 3 est transformé en un service web exécutable.

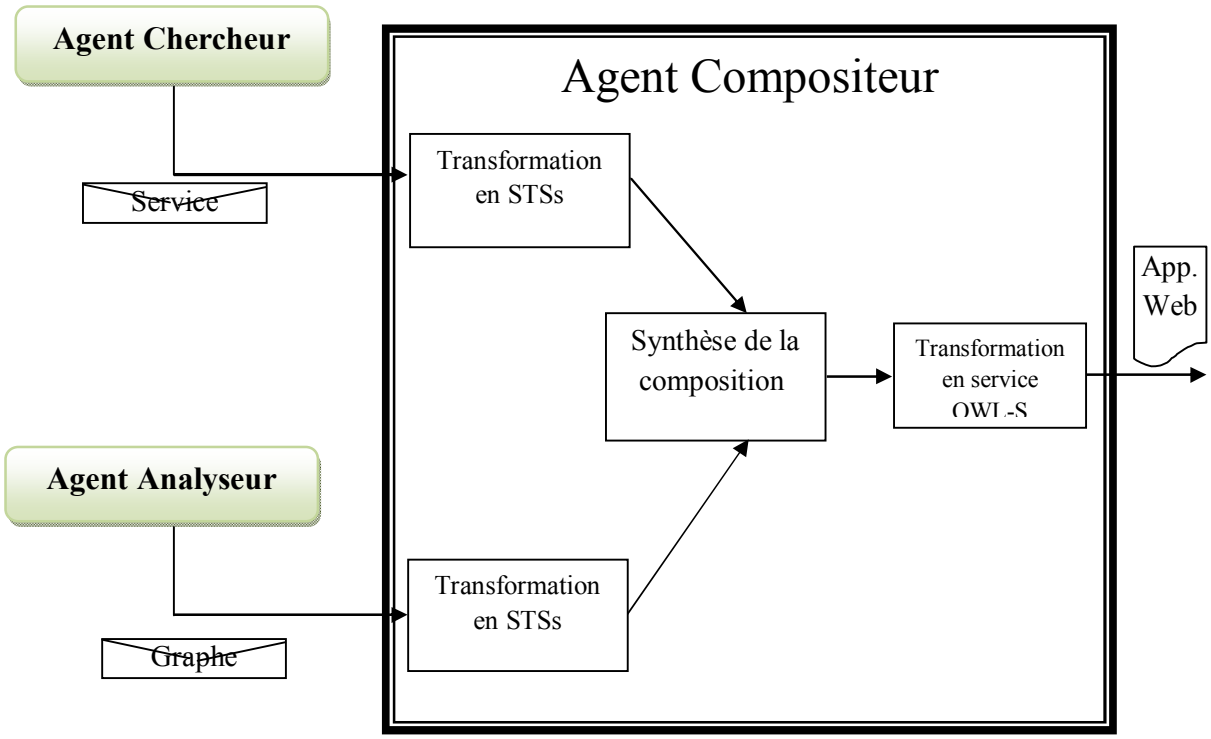


Figure 7. Architecture d'un agent Compositeur.

7.3.4. AGENT CREATEUR

Quand l'utilisateur veut ajouter un nouveau service, il envoi une requête d'insertion avec un ensemble d'informations (le nom de service, le texte de description, l'instance des relations, les entrées et les sorties, etc). L'agent créateur insère ce nouveau service dans l'ontologie des services avec ses informations. L'ontologie du processus doit aussi être changée par la connexion des paramètres d'entrés et de sorties de ce nouveau service aux autres paramètres compatibles. Ceci réduit la complexité de la recherche des services pour la composition automatique [64].

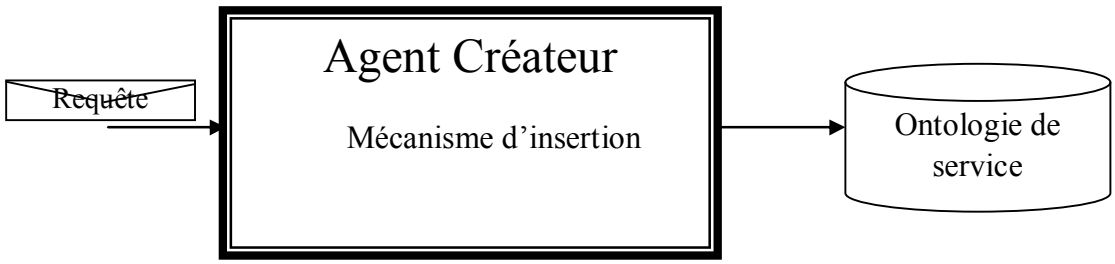


Figure 8. Architecture d'un agent créateur.

8. CONCLUSION

On a présenté un modèle basé service servant au partage des connaissances pour le développement, le déploiement, l'évolution et le soutient des applications Web. On a proposé de composer des services existant pour construire l'application Web. L'architecture de notre modèle se construit à partir des agents, des ontologies et des services Web sémantique. On a donné l'architecture globale du système puis on a présenté l'architecture locale de chaque agent.

Chapitre 5

1. INTRODCTION

Dans le chapitre précédant nous avons proposé un modèle de développement des applications Web à travers la composition des services existants. Afin de montrer la validité, la fiabilité et l'extensibilité du modèle, on a intérêt à faire une étude de cas. D'où nous allons appliquer notre modèle sur l'exemple d'organisation de voyage, un exemple fréquemment utilisé et il est concéderai comme un exemple typique pour la composition des services [53].

2. DESCRIPTION DU PROBLEME

On suppose le scénario suivant : Mohamed habite à Alger, il veut assister à une conférence au Kyoto (Japon). Il prépare les arrangements de voyage. Il doit consulter une compagnie aérienne pour le vol, il peut utiliser un aéroport près d'Alger et un aéroport près de Kyoto. Pour préparer les arrangements de logement, il cherche un hôtel proche au lieu de la conférence inclut une chambre libre à la date de son voyage.

Pour faciliter la tâche, il va utiliser l'Internet qui va lui offrir des services. Le premier service qu'il peut utiliser est de consulter le site d'une compagnie aérienne pour faire une réservation d'un vol. Ensuite, il va chercher dans le Web quelles sont les possibilités de réserver une chambre dans un hôtel. Nous remarquons que ce processus est complexe et nécessite plus d'attention et de temps. Notre but est d'automatiser ce processus en utilisant notre système.

A partir du scénario précédant, on peut dire que le processus d'organisation du voyage est composé de deux services majeurs : le service de réservation de vol et le service de réservation d'hôtel, de plus ces deux services sont eux même des services composés.

3. DESCRIPTIONS DES SERVICES

Un service est une entité qui présente une description de ce qu'il offre comme sortie, comment il fonctionne et enfin comment on peut l'employer. Dans la description de service on distingue trois éléments: le profil de service « *ServiceProfile »,* le modèle de service *« ServiceModel »* et les connaissances de service « ServiceGrounding ».

L'OWL-S propose des ontologies pour définir les services. Pour la réservation de voyage, qui forme le service global, nous allons modéliser les différents services Web participants avec l'ontologie OWL-S.

Le service RéservationHôtel accepte une requête pour réserver une chambre dans une période de temps et un emplacement, et si au moins un hôtel est disponible, il propose une offre d'hôtel à un certain prix. Cette offre peut être acceptée ou refusée par le service externe qui invoque le service RéservationHôtel. Le modèle de processus d'OWL-S pour ce service peut être comme suit :

```
<process:CompositeProcess rdf:ID="RéservationHôtel">
        <process:Sequence>
        <process:AtomicProcess rdf:about="#RequêteHôtel">
            <process:Input rdf:ID="Periode">
            <process:parameterType rdf:resource="#Periode"/></process:Input>
            <process:Input df:ID="Locatio">
            <process:parameterTyper df:resource="#Location"/></process:Input>
            <process:Conditional Output rdf:ID="Prix ">
            <process:coCondition rdf:resource="#RéservationHôtelPossible"/>
            <process:parameterType rdf:resource="#Prix "/>
            </process:ConditionalOutput>
            <process:ConditionalOutput rdf:ID="Hôtel">
            <process:coCondition rdf:resource="#RéservationHôtelPossible"/>
            <process:parameterType rdf:resource="#Hôtel"/>
            </process:ConditionalOutput>
            <process:ConditionalOutput rdf:ID="NA">
            <process:coCondition rdf:resource="#NotRéservationHôtelPossible"/>
            <process:parameterType rdf:resource="#NotDisponible"/>
             </process:ConditionalOutput>
        </process:AtomicProcess>
        <process:CompositeProcess>
            <process:IfThenElse>
            <process:ifCondition rdf:resource="#RéservationHôtelPossible"/>
            <process:then>
                  <process:CompositeProcess>
                  <process:Choice>
                  <process:AtomicProcess rdf:ID="#AccepteHôtelOffre "/>
                  <process:AtomicProcess rdf:ID="#RefuséHôtelOffre "/>
                  </process:Choice>
                  </process:CompositeProcess>
              </process:then>
              </process:IfThenElse>
         </process:CompositeProcess>
        </process:Sequence>
</process:CompositeProcess>
<process : condition rdf:ID="RéservationHôtelPossible">
```

```
<expr:expressionBody>
#DisponibleHôtel(#RequêteHôtel.Periode,#RequêteHôtel.Location) != UNDEF
</expr:expressionBody>
</process:condition>
```

Le service RéservationHôtel est composé du service comportant les processus atomiques suivants : RequêteHôtel, AccepteHôtelOffre, et RefuséHôtelOffre.

Le service RequêteHôtel reçoit en entrée une requête contient une période et un emplacement. Le service retourne une offre incluant le prix et le nom de l'hôtel s'il existe un hôtel avec une chambre disponible; sinon un message (NA) " non disponible " est retourné. Ceci représente la condition de sortie du service. La décision s'il existe un hôtel contient une chambre vide est prise suivant la condition RéservationHôtelPossible, qui est définie en terme de la fonction DisponibleHôtel (*d, e*) : cette fonction retourne le nom de l'hôtel si la réservation est possible dans la période *d* et l'emplacement *e* et *UNDEF* sinon. La fonction DisponibleHôtel (*d, e*) est utilisée aussi pour donner la valeur du paramètre de sortie Hôtel de service RequêteHôtel. Similairement, le paramètre de sortie Prix est défini en termes de la fonction PrixChambre (*p, h*), qui retourne le prix de la chambre dans la période *d* dans l'hôtel *h*.

Le service RéservationVol est conceptuellement semblable au service de RéservationHôtel: il accepte une requête de réservation d'un vol pour une période et un lieux donné. Si le vol est disponible, il retourne une offre avec le prix et le numéro d'avion. Cette offre peut être acceptée ou refusée par le service externe qui invoque le service RéservationVol. Son modèle de processus en OWL-S a aussi une structure similaire au modèle de processus de RequêteHôtel. Le service RéservationVol est une composition des processus atomiques suivants : RequêteVol, AccepteVolOffre et RefuseVolOffre. Le modèle de processus d'OWL-S pour le service RéservationVol est comme suit :

```
<process:CompositeProcess rdf:ID="RéservationVol">
        <process:Sequence>
           <process:AtomicProcess rdf:about="#RequêteVol">
           <process:Input rdf:ID="Periode">
           <process:parameterType rdf:resource="#Periode"/></process:Input>
           <process:Input df:ID="Location">
           <process:parameterTyperdf:resource="#Location"/></process:Input>
           <process:ConditionalOutput rdf:ID="Prix ">
              <process:coCondition rdf:resource="#RéservationVolPossible"/>
              <process:parameterType rdf:resource="#Prix "/>
              </process:ConditionalOutput>
```

```
                <process:ConditionalOutput rdf:ID="Vol">
                <process:coCondition rdf:resource="#RéservationVolPossible"/>
                <process:parameterType rdf:resource="# Vol "/>
                </process:ConditionalOutput>
                <process:ConditionalOutput rdf:ID="NA">
                <process:coCondition rdf:resource="#NotRéservationVolPossible"/>
                <process:parameterType rdf:resource="#NotDisponible"/>
                </process:ConditionalOutput>
        </process:AtomicProcess>
        <process:CompositeProcess>
                <process:IfThenElse>
                <process:ifCondition rdf:resource="#RéservationVolPossible"/>
                <process:then>
                <process:CompositeProcess>
                <process:Choice>
                <process:AtomicProcess rdf:ID="#AccepteVolOffre "/>
                <process:AtomicProcess rdf:ID="#RefuseVolOffre "/>
                </process:Choice>
                </process:CompositeProcess>
                </process:then>
                </process:IfThenElse>
        </process:CompositeProcess>
        </process:Sequence>
</process:CompositeProcess>
<process:condition rdf:ID="VolPossible">
<expr:expressionBody>
#DisponibleVol(#RequêteVol.Periode,# RequêteVol.Location) != UNDEF
</expr:expressionBody>
</process:condition>
```

4. SOLUTION PROPOSEE

Dans l'exemple précédant. Mohamed veut organiser son voyage en utilisant des services disponibles sur l'Internet. Notre modèle va lui offrir la possibilité de composer un ensemble de services tout en facilitant la tâche globale pour avoir le résultat désiré. Dans ce cas, Mohamed exprime sa requête et attend son application. Il va exprimer une requête qui contient la demande d'un service d'organiser le voyage. Le service Voyage contient deux services qui sont le service RéservationHôtel et le service RéservationVol. On peut remarquer que les deux services sont eux même des services composés. On peut exprimer la requête en OWL-S comme suit :

```
<process:CompositeProcess rdf:ID="Voyage">
        <process:Sequence>
        <process: CompositeProcess rdf:about="# RéservationVol"/>
          ……
        <process: CompositeProcess rdf:about="# RéservationHôtel"/>
          ……
        </process:Sequence>
 </process:CompositeProcess>
```

Comme nous avons dit dans le chapitre 4 (paragraphe 7.3), la requête peut être exprimée graphiquement sous fourme d'un graphe ET/OU. Les nœuds de ce graphe représentent les sous-requêtes (service élémentaire). Dans le cas de Voyage le graphe peut être présenté abstraitement comme suit :

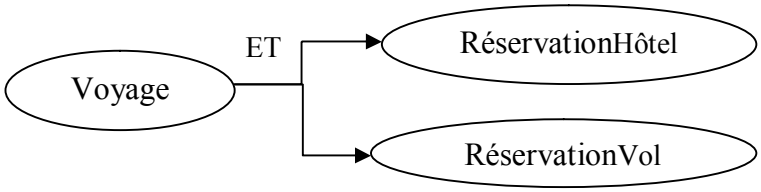


Figure 1. Graphe de service Voyage.

Dans notre modèle l'agent analyseur génère ce graphe puis le décompose en un ensemble de sous services. D'après le graphe illustré sur la figure 1, il y a deux sous-requêtes et chaque sous- requête corespende à un service. A chaque service sont associée des attributs qui forment le profil du service (les entrées, les sorties, les prés-condition…etc.), le modèle du processus de service (atomique ou composé) et d'autres informations non fonctionnelles concernant la qualité du service. L'agent analyseur invoque pour chaque sous-requête un agent chercheur.

Le groupe des agents chercheurs se constitue de deux agents. Chaque agent est respensable de la recherche d'un service convenable à la sous-requête et l'envoie à l'agent compositeur. Cette recherche est sous forme d'un parcoure de l'ontologie des services avec le teste des profils de services. Après que l'agent trouve le ou les services, il doit choisir le service qui vérifie les qualitées du service demandé par l'utilisateur.

Notre objectif est de construire un service composé appelé Voyage (les entrées sont une période et un emplacement spécifiques) qui interagisse avec les services RéservationHôtel et

RéservationVol. Il réserve un vol et une chambre à l'hôtel, les deux ensembles, quand c'est possible.

Pour la composition on va suivre les étapes présentées dans le chapitre précédant :

Dans l'étape1 : nous avons considéré que les services Web sont décrit en OWL-S. Ensuite, ces descriptions sont transformées en des STS. Le STS de figure 2 est obtenu à partir du modèle de processus OWL-S de service RéservationHotel et le STS de figure 3 est obtenu à partir de modèle de processus OWL-S de service RéservationVol.

La figure 2 présente la description du STS correspondant au service Web RéservationHôtel. L'ensemble des états modélise les étapes de l'évolution du processus et les valeurs de ses variables. La variable spéciale pc implémente le "compteur de programme" qui indique l'étape courante de l'exécution du service (c.-à-d., pc à la valeur de RéservationHôtel quant le processus est en attente de recevoir la requête de réservation, et la valeur testRéservationHôtelPossible quant il est prêt pour tester si la réservation est possible). Les autres variables comme location ou prix correspondent aux utilisations dans les exchanges des messages.

Finalement, des tableaux comme DisponibleHôtel ou PrixChambre décrient les prédicats et les fonctions expriment les prospérités de service Web (c.-à-d., le fait qu'il y a un hôtel disponible dans une période et un emplacement donné, ou le prix de la chambre d'un hôtel pour une période donnée). Dans l'étape initiale S_0, pc est placé pour START, alors que toutes les autres variables basiques sont indéfinis. Les valeurs initiales des tableaux DisponibleHôtel et PrixChambre sont non spécifiés, mais ils peuvent prendre n'importe quels valeurs du domaine.

L'évolution du processus est modélisée à travers un ensemble de transitions possibles. Chaque transition définies ses conditions d'application dans le code source d'état, ses actions, et l'état de destination. Pour l'instance, on impose que l'action τ peut être exécutée dans l'état checkRéservationHôtelPossible, aboutissant à l'état isRéservationHôtelPossible, s'il y a quelque hôtels avec une chambre disponible pour une période et un emplacement spécifiques. On remarque que chaque clause TRANS de la figure 2 correspondent aux différents éléments dans la relation de transition R : c.-à-d., la clause (1) génère différent éléments de R dépendent aux valeurs des variables période et emplacement.

pc = checkRéservationHôtelPossible &
DisponibleHôtel[Periode,location] ≠ UNDEF− [TAU]→ (1)
pc = isRéservationHôtelPossible.

D'après le modèle formel, on distingue entre différents types d'actions. Les actions d'entrées I modélisent toutes les requêtes arrivant au processus et l'information qu'elles apporte (c.-à-d., RéservationHôtel est utilisée pour les requêtes reçue de réservation). Les actions de sorties O représentent les messages sortants (dans notre cas, les réponses des requêtes d'entrées). L'action τ est utilisée pour modéliser les évolutions internes de processus.

Finalement, l'ensemble des prospérités prop de le STS sont des expressions de forme <variable> = <value> ou <array> [idx 1,...,idx n] = <value>, et la fonction d'étiquetage est la précédente.

PROCESS RéservationHotel;
TYPE Period; Location; Hotel; Prix; NA;
STATE pc: {start, receiveRéservationHotel, testRéservationHotelPossible, siRéservationHotelPossible, siNonRéservationHotelPossible, réponceRéservationHotelPossible, réponceRéservationHotelNonPossible, choixAcceptHotelOffreRefuseHotelOffre, réponceAcceptHotelOffre, réponceRefuseHotelOffre, termineRéservationHotelNonPossible, termineRefuseHotelOffre, termineAcceptHotelOffre ;}
period: Period ∪{ UNDEF } ;
loc: Location ∪{UNDEF} ;
Hotel: Hotel ∪{ UNDEF } ;
Prix: Prix ∪{UNDEF } ;
na: NA∪{UNDEF} ;
AvailableHotel[Period,Location]: Hotel ∪{ UNDEF} ;
PrixHotelChambre [Period,Hotel]: Prix;
INIT pc = start;
req period = UNDEF;
req loc = UNDEF;
Offre Hotel = UNDEF;
Offre Prix = UNDEF;
INPUT RéservationHotel(Period, Location); AcceptHotelOffre(); RefuseHotelOffre();
OUTPUT RéservationHotelAnswer (Hotel UNDEF , Prix UNDEF , NA UNDEF);
AcceptHotelOffreAnswer(); RefuseHotelOffreAnswer();
TRANS
pc = siRéservationHotelPossible – [TAU] → pc = réponceRéservationHotelPossible,
pc = start –[TAU]→pc = receiveRéservationHotel;
pc = receiveRéservationHotel– [INPUT RéservationHotel(period,location)]→pc = testRéservationHotelPossible;
pc = testRéservationHotelPossible ∧ AvailableHotel[period,location] ≠ UNDEF–[TAU] → pc= siRéservationHotelPossible;
pc = testRéservationHotelPossible ∧ AvailableHotel[period,location] ≠UNDEF– [TAU] →pc = siNonRéservationHotelPossible;
pc = siRéservationHotelPossible – [TAU] →pc = réponceRéservationHotelPossible, Hotel = vailableHotel[period,location], Prix = PrixHotelChambre[period, Hotel];
pc = réponceRéservationHotelPossible – [OUTPUT RéservationHotelAnswer(Hotel,Prix,na)] → pc = choixAcceptHotelOffreRefuseHotelOffre;
pc = choixAcceptHotelOffreRefuseHotelOffre – [INPUT AcceptHotelOffre] → pc = réponceAcceptHotelOffre;
pc = réponceAcceptHotelOffre – [OUTPUT AcceptHotelOffreAnswer] → pc = termineAcceptHotelOffre;
pc = choixAcceptHotelOffreRefuseHotelOffre – [INPUT RefuseHotelOffre] →pc = réponceRefuseHotelOffre;
pc = réponceRefuseHotelOffre – [OUTPUT RefuseHotelOffreAnswer] →pc = termineRefuseHotelOffre;
pc = siNonRéservationHotelPossible – [TAU] →pc = réponceRéservationHotelNonPossible, na ∈ NA;
pc = réponceRéservationHotelNonPossible – [OUTPUT RéservationHotelAnswer(Hotel,Prix,na)] → pc = termineRéservationHotelNonPossible;

Figure 2. Le STS de processus RéservationHôtel.

PROCESS RéservationVol;
TYPE Period; Location; Vol; Prix; NA;
STATE pc: {start, receiveRéservationVol, testRéservationVolPossible, siRéservationVolPossible, siNonRéservationVolPossible, réponceRéservationVolPossible, réponceRéservationVolNonPossible, choixAcceptVolOffreRefuseVolOffre, réponceAcceptVolOffre, réponceRefuseVolOffre, termineRéservationVolNonPossible, termineRefuseVolOffre, termineAcceptVolOffre ;}
period: Period ∪{ UNDEF } ;
loc: Location ∪{UNDEF} ;
Vol: Vol ∪{ UNDEF } ;
Prix: Prix ∪{UNDEF } ;
na: NA∪{UNDEF} ;
AvailableVol[Period,Location]: Vol ∪{ UNDEF} ;
PrixOfVol[Period,Vol]: Prix;
INIT pc = start;
req period = UNDEF;
req loc = UNDEF;
Offre Vol = UNDEF;
Offre Prix = UNDEF;
INPUT RéservationVol(Period, Location); AcceptVolOffre(); RefuseVolOffre();
OUTPUT RéservationVolAnswer (Vol UNDEF , Prix UNDEF , NA UNDEF);
AcceptVolOffreAnswer(); RefuseVolOffreAnswer();
TRANS
pc = siRéservationVolPossible--[TAU] → pc = réponceRéservationVolPossible,
pc = start --[TAU]→pc = receiveRéservationVol;
pc = receiveRéservationVol-- [INPUT RéservationVol(period,location)]→pc = testRéservationVolPossible;
pc = testRéservationVolPossible ∧ AvailableVol[period,location] ≠ UNDEF--[TAU] → pc= siRéservationVolPossible;
pc = testRéservationVolPossible ∧ AvailableVol[period,location] ≠UNDEF-- [TAU] →pc = siNonRéservationVolPossible;
pc = siRéservationVolPossible -- [TAU] →pc = réponceRéservationVolPossible,
Vol = AvailableVol[period,location],
Prix = PrixVol[period, location];
pc = réponceRéservationVolPossible -- [OUTPUT RéservationVolAnswer(Vol,Prix,na)] → pc = choixAcceptVolOffreRefuseVolOffre;
pc = choixAcceptVolOffreRefuseVolOffre -- [INPUT AcceptVolOffre] → pc = réponceAcceptVolOffre;
pc = réponceAcceptVolOffre--[OUTPUT AcceptVolOffreAnswer] → pc = termineAcceptVolOffre;
pc = choixAcceptVolOffreRefuseVolOffre-- [INPUT RefuseVolOffre] →pc = réponceRefuseVolOffre;
pc = réponceRefuseVolOffre-- [OUTPUT RefuseVolOffreAnswer] →pc = termineRefuseVolOffre;
pc = siNonRéservationVolPossible-- [TAU] →pc = réponceRéservationVolNonPossible, na ∈ NA;
pc = réponceRéservationVolNonPossible-- [OUTPUT RéservationVolAnswer(Vol,Prix,na)] → pc = termineRéservationVolNonPossible;

Figure 3. Le STS de processus RéservationVol.

Dans l'étape 2 : c'est l'étape de définition du but de la composition. Dans notre exemple, la requête de l'utilisateur spécifie le lieu que l'utilisateur vent visiter et dans quel période. Afin de satisfaire cette requête, le service composé doit trouver un vol et une chambre à un hôtel compatible avec ceux de la requête. Le but de la composition peut être comme suit "si l'offre de voyage est disponible, alors l'alloué à utilisateur ". Dans notre cas, l'offre est possible s'il est possible de réserver un vol à un emplacement spécifique et une chambre d'hôtel pour une période donnée par l'utilisateur. Le fait que l'offre est allouée, il est décrit par le besoin que les processus HotelBooking et FlightBooking tiennent l'état final d'acceptation de l'offre de réservation. La condition du but peut être spécifiée par la formule :

HotelBooking.AvailableHotel[p,l] ≠ UNDEF ^ FlightBooking.AvailableFlight[p,l] ≠ UNDEF

→HotelBooking.pc = endAcceptHotelBooking
FlightBooking.pc = endAcceptFlightBoo_king ^
h = HotelBooking.AvailableHotel[p,l] ^
f = FlightBooking.AvailableFlight[p,l] ^
c = HotelBooking.CostOfRoom[p,h] + FlightBooking.CostOfFlight[f].

Où c, f, et h décrivent, respectivement, le prix d'offre, et les informations d'un vol et d'un hôtel spécifiques. Le but a été interprété comme une condition qui doit être atteint à la fin de l'exécution du service composé.

Dans l'étape 3 : La part essentielle de la tâche de la composition automatique est l'étape 3. La définition 5 de chapitre 4 paragraphes de l'agent compositeur est une définition formelle pour la résolution du problème par cette étape. Nous avons démarré à partir d'un ensemble de STSs des services Web existants et le but de la composition, un nouveau STS est généré celui-ci implémente le service composé. On a vu que le problème de la composition au niveau du processus est un problème de planification, où le STS parallèle $\sum\|$ est le domaine de planification, ρ est le but de planification, et $\sum_c$ est le plan de génération de but ρ.

La génération du plan $\sum_c$ n'est pas une séquence d'actions, il doit représenter des traitements des actions conditionnelles et des itérations. Pour générer le STS $\sum_c$, on propose l'utilisation de technique "planning as model checking" qui est détaille dans [53].

Dans l'étape 4, le STS $\sum_c$ généré automatiquement transformé en un service Web exécutable.

Un modèle du processus OWL-S pour le service Voyage est décrit en dessous. La spécification OWL-S commence par la déclaration des entrées (Période et Emplacement) et les sorties (Prix, Hôtel, Vol ou NA) sont les paramètres de la composition. Puis, le corps de la composition est défini comme une interaction souhaitable entre les services atomiques des services RéservationHôtel et RéservationVol.

Dans cet exemple, on a choisi que le service Voyage interagit premièrement avec le service de l'hôtel et puis avec le service de vol, par conséquent, le processus atomique RequêteHôtel est appelé. Si la réservation est possible, la condition RéservationHôtelPossible est vérifiée et le processus de service Voyage commence à interagir avec le service RequêteVol de RéservationVol. Si le vol est disponible (la condition RéservationVolPossible), le processus de service Voyage accepte à la fois l'offre de service de réservation d'hôtel (AccepteHôtelOffre) et celle du service réservation de vol (AccepteVolOffre). Si le vol n'est pas disponible, le service Voyage doit refuser seulement l'offre de l'hôtel (RefuséHôtelOffre) reçue précédemment. On remarque que la condition RéservationHôtelPossible est définie en termes de contenu de la réponse reçue à partir de service atomique RequêteHôtel. En effet, la réservation d'hôtel est possible si et seulement si NA n'est pas dans la réponse.

```
<process:CompositeProcess rdf:ID=" Voyage ">

        <process:hasInput rdf:resource="#Periode"/>
        <process:hasInput rdf:resource="#Location"/>
        <process:hasResult rdf:resource="#Prix "/>
        <process:hasResult rdf:resource="#Hôtel"/>
        <process:hasResult rdf:resource="#Vol"/>
        <process:hasResult rdf:resource="#NA"/>

        <process:Sequence>
        <process:AtomicProcess rdf:about="#RequêteHôtel"/>

        <process:CompositeProcess>
                <process:IfThenElse>
                <process:ifCondition rdf:resource="#RéservationHôtelPossible"/>
                <process:then>
                <process:CompositeProcess>
                <process:Sequence>
                <process:AtomicProcess rdf:about="#RequêteVol"/>
```

```
                <process:CompositeProcess>
                <process:IfThenElse>
                <process:ifCondition rdf:resource="#RéservationVolPossible"/>
                <process:then>
                <process:CompositeProcess>
                    <process:Sequence>
                     <process:AtomicProcess rdf:about="#AccepteeHôtelOffre"/>
                     <process:AtomicProcess rdf:about="#AccepteVolOffre "/>
                     </process:Sequence>
                     </process:CompositeProcess>
                     </process:then>
                     <process:else>
                     <process:AtomicProcess rdf:about="#RefuséHôtelOffre"/>
                     </process:else>
                     </process:IfThenElse>
                     </process:CompositeProcess>
                     </process:Sequence>
                  </process:CompositeProcess>
                  </process:then>
                  </process:IfThenElse>
                  </process:CompositeProcess>
                </process:Sequence>
   </process:CompositeProcess>

<process:condition rdf:ID="RéservationHôtelPossible">
<expr:expressionBody>
#RequêteHôtel.NA = UNDEF
</expr:expressionBody>
</process:condition>
...
<process:sameValues rdf:parseType="Collection">
<process:ValueOf>
<process:theParameter rdf:resource="#Periode"/>
<process:atProcess rdf:resource="#HôtelAndVol"/>
</process:ValueOf>
<process:ValueOf>
<process:theParameter rdf:resource="#Periode"/>
<process:atProcess rdf:resource="#RequêteHôtel"/>
</process:ValueOf>
</process:sameValues>
...
```

5. OUTIL DE PROGRAMMATION

Pour l'implémentation du système en utilisant la langage Java pour des raisons de portabilité.

5.1. LA PLATEFORME JADE

Jade est un logiciel libre distribué par TILab en Open Source avec une licence LGPL [18]. Jade a pour but de simplifier le développement des systèmes multi-agents tout en fournissant un ensemble complet de services et d'agents conformes aux spécifications FIPA. La plate-forme d'agent Jade inclut tous les composants obligatoires qui contrôlent un SMA. Ces composants sont l'ACC, l'AMS et le DF. Toute communication entre agents utilise des messages FIPA ACL.

Les agents peuvent être développés avec la technologie JADE (Java Agent DEvelopment Framework) qui est une plate-forme multi-agents développée en Java par CSELT pour le développement d'agents. Elle permet aussi le déploiement des services Web via son extension WSIG (Web Services Integration Gateway). C'est un environnement de développement de SMA compatible avec les spéc. FIPA2000 composé de :

- un système d'exécution pour SMA (compatible FIPA)

une plate-forme Java de développement de SMA en rendant accessible les fonctionnalités standards FIPA (FIPA standard assets).

5.2. L'ÉDITEUR DE OWL-S:

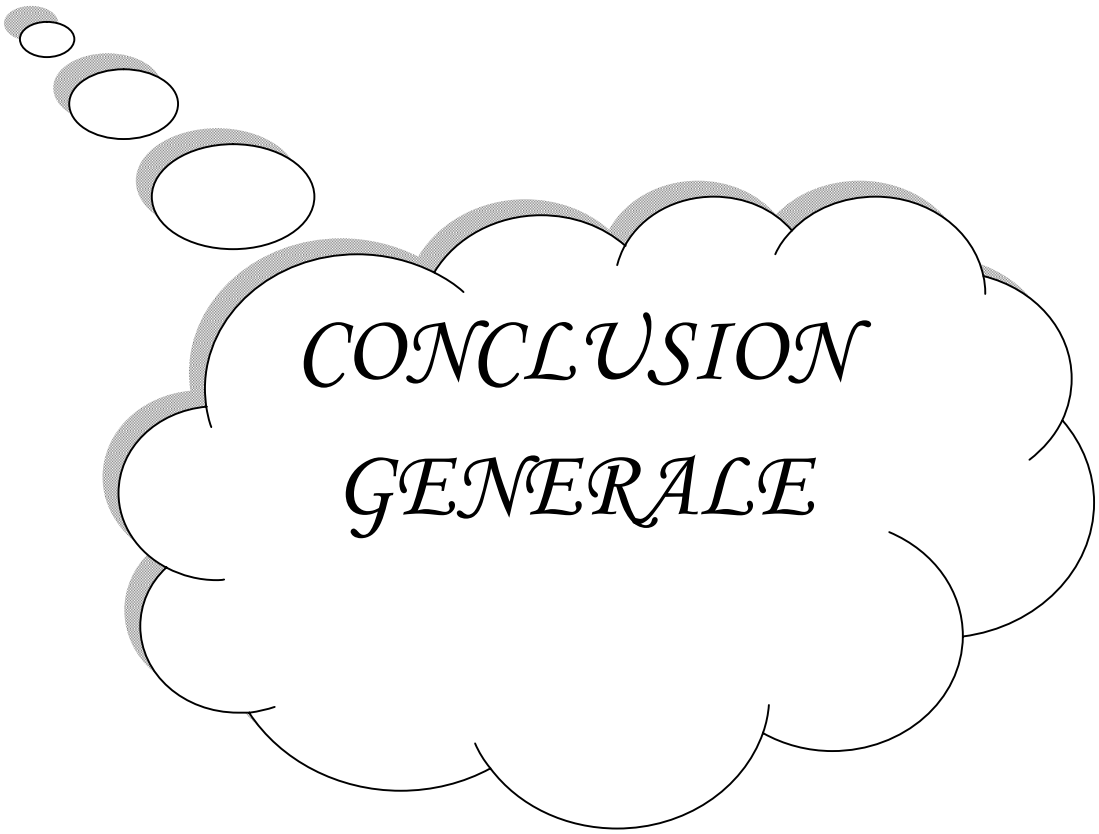
L'éditeur OWL-S est un outil qui a pour objectif de permettre le développement facile et intuitif des services Web en OWL-S pour des utilisateurs non experts dans ce langage. Il fournit des moyens pour faciliter la conception des services Web sémantiques.

6. CONCLUSION

Dans ce chapitre nous avons appliqué notre modèle sur l'exemple de l'organisation d'un voyage. Nous pouvons remarquer que notre système facilite la création des applications Web et décharge les concepteurs d'un grand travail. Cette facilité réside dans le fait que le concepteur donne une spécification de l'application Web et notre système génère l'application Web complète. Par contre, il faut avoir un langage de spécification formelle pour décrire formellement les spécificités des applications Web désirées prenant en compte l'aspect sémantique de ces applications. Ainsi on peut garantir la fiabilité de l'application générée

(elle répond à ses spécifications). Nous avons utilisé OWL-S pour décrire les services existants. Pour la composition, nous avons fait appel à la méthode de composition au niveau de processus qui garantit l'assurance de la logique réelle des applications Web.

Chapitre 6



1. CONCLUSION

Tout au long de ce mémoire, nous avons présenté les différentes technologies nécessaires pour proposer un modèle basé sur services pour le développement des applications Web. On a utilisé les technologies des services Web sémantiques et les ontologies.

Le but que nous avons abouti est un modèle de développement d'applications Web à partir de services existants. Notre modèle est un système multi-agents où les agents interagissent et coopèrent entre eux pour générer une application Web pour prendre en charge le processus de développement des applications Web basées services.

La composition des services Web sémantiques est réalisée en suivant une approche basée sur la composition des processus pour donner finalement le processus global de l'application Web.

Afin de démontrer notre travail, nous avons fait une étude de cas où nous avons appliqué notre modèle sur l'exemple de l'organisation d'un voyage.

2. PERSPECTIVES

Nous proposons certaines perspectives à savoir :

- Chaque agent offre ses propres services et les autres agents vont solliciter ses services. i.e. les services publiés seront gérés par des agents qui s'occupent de l'aspect de négociation des contraintes d'utilisation des ces services proposés. Dans ce cas l'application Web est le résultat de la collaboration des agents. Avec le temps, un mécanisme d'intégration des ontologies sera nécessaire pour garantir la fiabilité du système.
- Rendre notre système multi-agents ouvert qui peut communiquer avec les autres systèmes multi-agents. Pour cela, il faut avoir des mécanismes permettant l'intégration des systèmes hétérogènes pour coopérer et construire les applications Web.
- Utiliser d'autres méthodes formelles plus puissantes pour la composition et qui prennent en compte l'aspect de profils des services et l'aspect de processus, tout les deux ensemble pour aboutir à des meilleurs résultats.