Introduction générale

Les dernières décennies ont été marquées par le développement rapide des systèmes d'information distribués, et tout particulièrement par la diffusion de l'accès à Internet. Cette évolution du monde informatique a entraîné le développement de nouveaux paradigmes d'interaction entre applications tels que SOA (Service Oriented Architecture). Cette dernière a été mise en avant afin de permettre des interactions entre applications distantes.

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

La SOA offre un cadre adéquat pour le développement des Web services. Un Web service est une application simple qui implémente une fonctionnalité et qui la rend accessible sur internet en se basant sur des standards XML. Le développement des Web services est dû à leur simplicité et surtout à leur caractéristique principale : l'interopérabilité. En effet, cette propriété permet de résoudre un problème très ardu dans l'informatique répartie : celui de permettre à des applications hétérogènes de communiquer indépendamment des caractéristiques de la plate-forme matérielle et logicielle.

La dernière décennie a enregistré un développement intense des Web services. L'idée de composer des Web services a alors surgi : n'est-il pas possible de combiner des Web services existants pour obtenir de nouvelles fonctionnalités plus élaborées ? La réponse est le plus souvent affirmative. On parle alors de composition de Web services. Cette composition peut se faire selon plusieurs stratégies. Dans ce projet nous intéressons à la composition dynamique et plus particulièrement à la génération automatique d'un plan de composition.

Plusieurs travaux se sont intéressés à la génération (ou la construction) automatique d'un plan de composition. Notre travail s'inscrit dans le cadre de l'utilisation de la planification par intelligence artificielle. Des outils de composition ont été développés et mis à la disposition des travaux académique. Nous utilisons un outil qui s'appelle « OWLSXPlan » qui permet de construire un plan de composition de Web services.

Afin de fournir un Web service composite (ou issu d'une composition) exécutable nous nous sommes fixé l'objectif d'ajouter un module à l'outil « OWLXPlan ». Le rôle de ce module est de traduire le service composite généré (exprimé en OWL-S, donc sous forme non

exécutable) en un service composite exprimé dans le langage BPEL qui offre un cadre d'exécution.

La suite de ce mémoire s'articule sur 4 chapitres. Le premier chapitre présente le cadre du travail ; c'est-à-dire les Web services, leur définition, leur architecture, ainsi que les standards sur lesquels ils reposent.

Dans le second chapitre nous nous intéressons à la composition de Web services : ce qu'elle apporte et les différentes techniques et stratégies qui permettent de produire un Web service composite. Ce chapitre introduira également le besoin d'une annotation sémantique de Web services ce qui donne lieu aux Web services sémantiques.

Le chapitre suivant, quant à lui, s'occupera de l'utilisation de la planification par intelligence artificielle pour produire un plan de composition dynamique de Web services. Ce chapitre se chargera également de présenter le framework OWLS-XPlan utilisé pour la planification.

Le rôle du chapitre 4 est de proposer un Web service composite sous forme exécutable, et ceci en translatant le service composite produit par OWLS-XPlan et exprimé en OWL-S en un service composite exprimé en BPEL.

Enfin, une conclusion générale mettra le point sue ce que nous avons appris et réalisé pendant ce projet et les perspectives de ce modeste travail.

Introduction :

L'évolution d'Internet et la compétitivité entre les entreprises ont été les facteurs de l'explosion des Web services. En effet, les Web services peuvent constituer un apport de rapidité et d'efficacité pour l'e-business. La notion de Web service désigne essentiellement une application (un programme) mise à disposition sur Internet par un fournisseur de service, et accessible par des clients à travers des protocoles Internet standards. Leurs particularités par rapport aux autres technologies de l'informatique répartie résident dans le fait qu'ils offrent un modèle de composants à couplage faible en utilisant la technologie Internet comme infrastructure pour la communication. La composition des Web services constitue une évolution naturelle de cette technologie qui permet de mettre en place des composants Web services au profit de l'intégration d'applications sur le web afin d'atteindre de meilleures solutions **[10]**.

I.1. Les Web Services

I.1.1. Définition

Un Web service est un composant logiciel accessible à travers des intranets, des extranets et l'Internet moyennant des technologies web et un système standard de messagerie basé sur XML permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués **[2]**.

I.1.2. Architecture des Web services

I.1.2.1. Architecture de base

Trois rôles « Publish-Find-Bind » définissent l'architecture de base des Web services **[1]** (figure 1). Les acteurs de ces rôles sont :

a) **Le fournisseur de service** : crée un Web service et le publie dans l'annuaire de service (Publish) **[3]**.

b) **Le demandeur de services :**(client) effectue des recherches dans l'annuaire de services pour trouver (Find) la description du service qui répond à ses besoins.

c) **L'annuaire de services :** fournit au client la description d'un Web service, ce qui lui permet de faire une liaison (Bind) directe afin d'invoquer le service.

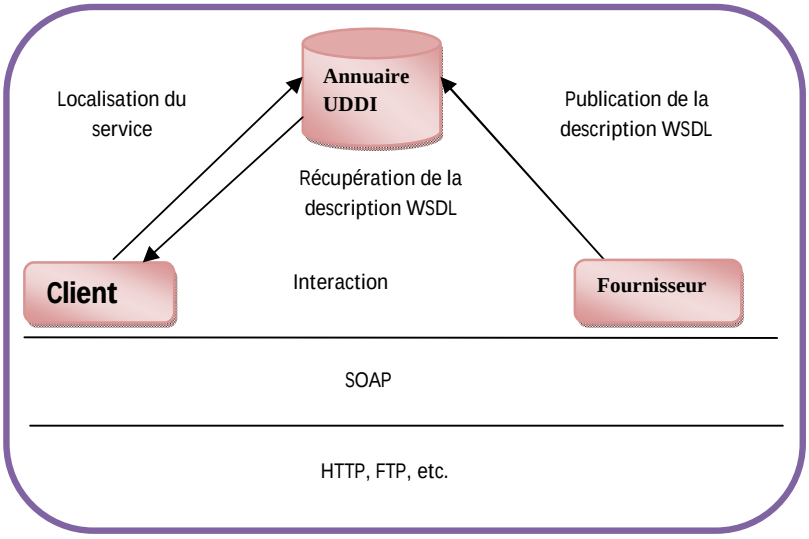

Fig.1. Architecture des Web services **[1].**

I.1.2.2. Architecture étendue

Une architecture étendue est constituée de plusieurs couches se superposant les unes sur les autres, d'où le nom de pile des Web services. La pile est constituée de plusieurs couches, chaque couche s'appuyant sur un standard particulier. On retrouve, au-dessus de la couche de transport, les trois couches formant l'infrastructure de base.

Nous apportons une explication de la mise en relief des trois types de couches **[2]**:

- **L'infrastructure de base (Discovery, Discription, Exchange) :** Ce sont les fondements techniques établis par l'architecture de référence. Nous distinguons les échanges des messages établis par SOAP, la description de service par WSDL et la recherche de Web services que les organisations souhaitent utiliser via le registre UDDI **[2]**.

- **Couches transversales (Security, Transactions, Administration, QoS) :** Ce sont ces couches qui rendent viable l'utilisation effective des Web services dans le monde industriel ;

- **La couche Business Processus (BusinessProcess) :** Cette couche supérieure permet l'intégration de Web service. Elle établit la représentation d'un **« BusinessProcess »** comme un ensemble de Web service. De plus, la description de l'utilisation de différents services composant ce service est disponible par l'intermédiaire de cette couche.

Fig.2. Architecture en Pile des services Web **[2].**

I.1.3. Le langage WSDL

Le langage WSDL est l'acronyme de « Web Service Description Languge »est un langage basé sur XML permettant de décrire et de publier les interfaces et protocoles des services Web d'une manière standard. L'interface d'un service Web décrit en fait tout le fonctionnement d'un service Web, et cache tout le détail de l'implémentation du service Web, donc elle est indispensable pour pouvoir invoquer un service Web par une application cliente, ou un autre service Web permettant une utilisation indépendante de la plateforme utilisée ainsi que du langage utilisé.

Le langage WSDL présente un format commun pour la description et la publication des interfaces et protocoles relatifs aux services Web. Une description WSDL d'un service Web est faite sur deux niveaux, niveau abstrait et niveau concret (voir la Figure suivante). Au niveau abstrait, la description du service Web consiste à définir les éléments de l'interface du service Web tel que : Types de données (DataTypes), les messages (Message), les opérations (Operation), les types de ports (PortType), et les liaisons (Bindings), ces parties décrivent des informations abstraites indépendantes au contexte de mise en œuvre. On y trouve : Les types de données envoyées et reçues ; les opérations utilisables et le protocole qui sera utilisé

_ Les types de données : « Data types »est l'élément qui définit les types de données utilisées dans les messages échangés par le service Web. Une fois définie, les « Data types », ou type peuvent être référencés dans n'importe quel message.

_ Les messages : L'élément « Message »spécifie les types d'opérations supportées par le service Web, il permet d'incorporer une séquence de messages corrélés sans avoir à spécifier

les caractéristiques du flux de données, par exemple, un message Input et un message Output corrélés sont mis en correspondance dans une seule opération de type « Request/Response ».

_ Les Opérations : l'élément « Operation »spécifie les types d'opérations supportées par le service Web, il permet d'incorporer une séquence de messages corrélés sans avoir à spécifier les caractéristiques du flux de données.

_ Les PortType : le « PortType »est un groupement logique ou une collection d'opérations supportées par un ou plusieurs protocoles de transport, il est analogique à une définition d'un objet contenant un ensemble de méthodes.

_ Les Liaisons : décrit la façon dont un type de port est mis en œuvre pour un protocole particulier (HTTP), et un mode d'invocation (SOAP). Cette description est faite par un ensemble donné d'opération abstraite. Pour un type de port, on peut avoir plusieurs liaisons, pour différencier les modes d'invocation ou de transport de différentes opérations.

Au niveau concret, le service Web est défini grâce aux deux éléments ; Port et Service. Ces deux dernières décrivent des informations liées à un usage contextuel du service Web. On y trouve : l'adresse du fournisseur implémentant le service, et le service qui est représenté par les adresses des fournisseurs.

_ L'élément Port : l'élément « Port », dans la partie concrète, spécifie une adresse URL qui correspond à l'implémentation du service Web par un fournisseur, et identifie un ou plusieurs « Bindings »(ou liaisons) aux protocoles de transports (HTTP, SMTP, FTP, . . .) pour un

« Port-Type »donné. La séparation du protocole de transport de la définition du « PortType» permet à un service Web d'être valable à travers plusieurs protocoles de transports, sans avoir à redéfinir l'ensemble du fichier WSDL.

_ L'élément Service : spécifie l'adresse complète du service Web, et permet à un point d'accès d'une application distante de choisir à exposer de multiples catégories d'opérations pour divers types d'interactions. **[3]**

```xml
<?xml version="1.0" encoding="UTF8" ?>
<definitions
   xmlns:tns="http://WS.pg/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   targetNamespace="http://WS.pg/" name="WSService">
   <types>
      <xsd:schema>
         <xsd:import namespace="http://WS.pg/"
         schemaLocation="http://localhost:15051/WebApp/WSService?xsd=1" />
      </xsd:schema>
   </types>
   <message name="direBonjour">
      <part name="parameters" element="tns:direBonjour" />
   </message>
   <message name="direBonjourResponse">
      <part name="parameters" element="tns:direBonjoursResponse" />
   </message>
   <portType name="WS">
      <operation name="direBonjour">
         <input message="tns:direBonjour" />
         <output message="tns:direBonjourResponse" />
      </operation>
   </portType>
   <binding name="WSPortBinding" type="tns:WS">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
         <operation name="direBonjour">
         <soap:operation soapAction="" />
            <input>
               <soap:body use="literal" />
             </input>
            <output>
               <soap:body use="literal" />

             </output>

         </operation>

   </binding>

   <service name="WSService">

      <port name="WSPort" binding="tns:WSPortBinding">

         <soap:address location="http://localhost:15051/WebApp/WSService" />

      </port>

   </service>

</definitions>
```

Fig.3 : Extrait de code WSDL décrivant un Web Service

I.1.4. Avantages des Web services

Les Web services ont plusieurs avantages par rapport aux technologies de systèmes répartis. Néanmoins nous citons les plus importants :

- **L'interopérabilité** qui garantit l'indépendance vis-à-vis des plateformes matérielles et logicielles **[2]**.
- **La simplicité** : pour utiliser les Web services il suffit de connaitre un langage tel que Java, C++, ou PHP ; un serveur Web et une connexion Internet.
- **Le couplage lâche** : contrairement aux autres technologies réparties, dans les applications SOA, le couplage entre services est lâche (très bas) ce qui rend leur maintenance très aisée.

I.2. La composition des Web services

I.2.1. Pourquoi la composition de Web services ?

Les composants sont destinés à être composés. La composition, et donc la réutilisation, est l'une des caractéristiques les plus importantes des systèmes à composants. Réellement, il n'est pas toujours facile de trouver des Web services qui s'apparient avec les requêtes des utilisateurs. Par conséquent, la composition des services satisfaisant la requête est un besoin grandissant de nos jours **[2]**.

La composition de services est une tâche complexe. Ces dernières années, la recherche dans le domaine des Web services a été très développée. Une grande partie de cette recherche a été consacrée à la composition de Web service qui autorise la composition des services simples, afin d'obtenir un service complexe qui répond aux exigences d'un utilisateur **[11]**.

I.2.2. Définition de la composition

La composition des Web services est le processus de construction de nouveaux Web services à valeur ajoutée, à partir de deux ou plusieurs Web services déjà présents et publiés sur le Web. Un Web service est dit composé ou composite lorsque son exécution implique des interactions et des changements des messages avec d'autres Web services, afin de faire appel à leurs fonctionnalités. La composition de Web services spécifie quels services ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'interaction **[7].**

I.2.3. Les stratégies de composition de Web services

Deux étapes sont nécessaires dans la production d'un Web services composite : la génération du plan de composition et la sélection des Web services qui vont réaliser les activités du plan.

I.2.3.1. Choix, sélection et liaison des Web services

Les Web services qui vont exécuter les activités du plan de composition doivent d'abord êtes trouvés. Pour cela, une recherche dans les annuaires de services est nécessaire. Si plusieurs services sont candidats pour réaliser une certaine activité, une sélection doit être faite. Cette sélection peut s'appuyer sur des paramètres tels que la disponibilité, le coût, le temps de réponse, etc. Une fois un service sélectionné il est alors lié à une activité ce qui permettra de l'invoquer. Suivant la stratégie de composition utilisée, ces opérations peuvent être réalisées pendant la conception du service composite ou pendant l'exécution du service **[3]**.

I.2.3.2. Génération du plan de composition

Le plan qui permet de répondre à une requête spécifique doit être généré. Suivant la stratégie de composition, ce plan peut être abstrait ou concret. Ce plan exprime le flot de données et de contrôle à travers des activités simples. Chacune de ces activités sera liée à un service donné. La génération du plan peut être manuelle : l'utilisateur spécifie lui-même le plan de composition. Cette génération peut être semi-automatique : l'utilisateur est assisté dans la génération de plan par des outils. Une génération automatique de plan signifie que le système propose un plan de composition sans intervention de l'utilisateur **[3]**.

I.2.3.3. Classification des stratégies de composition

En s'appuyant sur la méthode de génération de plan et sur le moment de sélection des Web services, une classification peut être faite. Nous pouvons distinguer trois stratégies majeures de composition **[3]**:

a) Composition statique : la génération du plan est manuelle et la sélection des Web services qui vont réaliser les activités du plan est faite pendant la conception du service composite. Ceci signifie que le service composite va invoquer les mêmes Web services chaque fois qu'il est exécuté**.**

b) Composition semi-dynamique : dans les méthodes de composition semi-dynamique, l'utilisateur est activement assisté dans la génération du plan de composition et/ou dans la sélection des Web services. Il est possible par exemple de présélectionner un ensemble de

services pour une activité donnée, et pendant l'exécution la sélection se fera à partir de cet ensemble.

c) Composition dynamique : dans une composition dynamique il n'y a pas d'intervention de l'utilisateur. Le plan est généré automatiquement suite à une requête. Les Web services qui vont être liés aux différentes activités du plan seront sélectionnés pendant la phase d'exécution.

Conclusion :

De plus en plus, avec l'essor d'Internet, le développement tend vers les technologies du Web. Les Web services sont des composants logiciels représentant une fonction applicative, ils représentent un mécanisme de communication entre applications distantes à travers le Web. Les Web services peuvent être utilisés par les humains ou programmes afin d'accomplir une tâche donnée.

Un des concepts intéressants qu'offre la technologie de service Web, et qui suscite beaucoup d'intérêt, est la possibilité de créer un nouveau service Web à valeur ajoutée par composition de plusieurs services Web existants. La composition entre donc dans cet ensemble de tâches, elle est intrinsèquement liée au problème de la description des Web services. La composition constitue un point essentiel dans notre travail. Ainsi, la composition des Web services définit un procédé utilisable, énoncé comme un nouveau service Web, dont les activités composantes sont d'autres Web services. Les Web services composites ainsi obtenus peuvent entrer à leur tour dans d'autres compositions pour fournir des nouveaux Web services plus complexes à forte valeur ajoutée.

Introduction :

L'utilisation intensive des standards, d'une part, et celle du langage XML, d'autre part, constituent les deux caractéristiques fondamentales de la technologie des Web services. Par conséquent, les différentes spécifications d'un Web service peuvent être facilement traitées par une machine. Par exemple, une description WSDL peut être automatiquement analysée par un parseur XML pour générer un proxy client facilitant ainsi la tâche du programmeur lors de l'implantation de son application. Cependant, le consensus sur la mécanique des interactions (format des messages, types des données et protocoles d'échanges) n'est pas suffisant pour permettre aux Web services d'interagir de manière claire, autonome et non ambigüe. Par exemple, deux descriptions XML identiques peuvent avoir des significations différentes selon le contexte. L'absence d'une sémantique explicite limite les possibilités d'automatisation des Web services. Pour pallier cette limitation, il y a clairement un besoin de langages de descriptions des services permettant de conférer une signification explicite et non ambigüe aux descriptions des Web services (figure 4). Par exemple, la nouvelle version du langage WSDL préconisée par leW3C permet de définir la sémantique des termes utilisés dans une description WSDL en référençant des ontologies RDF ou OWL.

Le Web sémantique a le potentiel de fournir l'infrastructure des Web services avec des informations sémantiques dont ils ont besoin. Il doit fournir des langages formels et des ontologies permettant de raisonner sur la description des services, le contenu des messages et les relations entre ces ontologies. Le résultat de l'utilisation de Web sémantique est une description non ambigüe de l'interface de Web services qui est compréhensible par une machine et fournit la base d'une interopérabilité transparente entre les différents services. Le langage de description de référence des Web services sémantiques est le langage OWL-S qui est une extension du langage OWL **[4]**. Le langage OWL-S fournit une ontologie de processus dans laquelle les processus sont vus de deux façons : un processus produit une transformation à partir d'un ensemble de données d'entrée vers un ensemble de données de sortie; et un processus produit une transition d'un état du monde vers un autre état du monde. Cette transition est décrite en termes de pré-conditions et d'effets. L'objectif est de trouver une solution intelligente pour l'automatisation de la composition dynamique des Web services sémantiques.

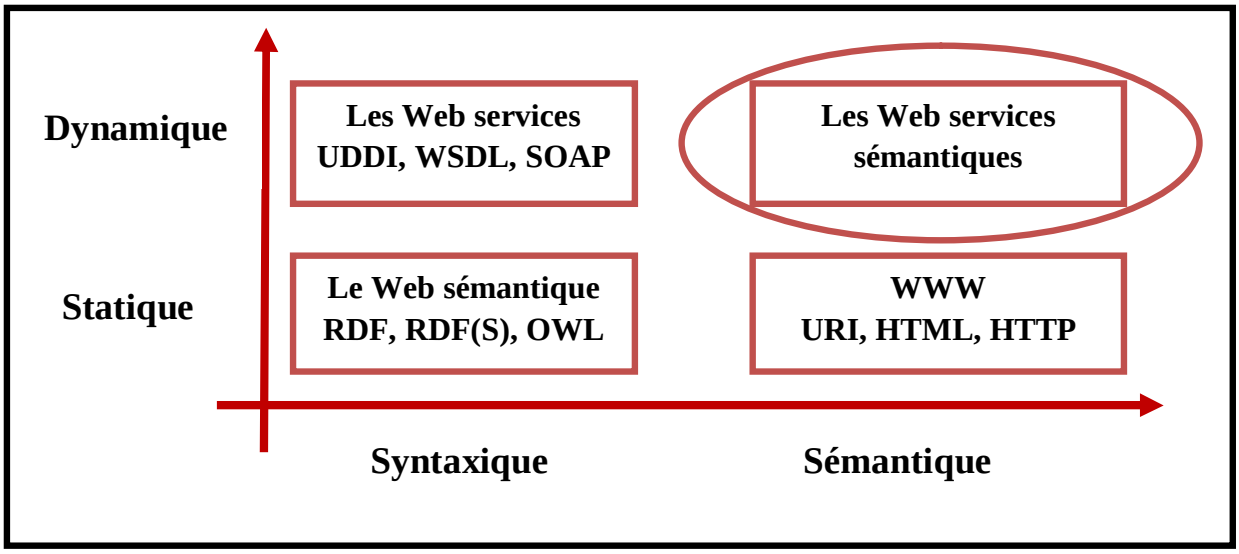

Fig.4. L'évolution du Web **[3].**

II.1. Les différents langages de définition et de manipulation d'ontologies

Evidemment, cette définition très générale laisse libre cours à l'imagination au sujet des applications potentielles : gestion de ressources, management, création de référentiels médicaux, etc. C'est sans doute la raison pour laquelle il existe de nombreux langages informatiques, plus ou moins récents, spécialisés dans la création et la manipulation d'ontologies. En voici quelques exemples :

- Open Knowledge Base Connectivity : OKBC 2.0 est une API permettant d'accéder à des bases de connaissance (Knowledge Representation System).
- Knowledge Interchange Format : est un langage destiné à faciliter des échanges de savoirs entre des systèmes informatiques disparates.
- LOOM est un langage de représentation des connaissances dont le but avoué est de «Permettre la construction d'applications intelligentes».
- DARPA Agent Markup Language : fondé sur XML et RDF, DAML-ONT est apparu afin d'autoriser l'expression de classes RDF plus poussées que ce que permettait à l'époque RDFS**.**

II.2. Le langage OWL (Ontology Web Langage)

II.2.1.Objectifs et motivation

OWL est, tout comme RDF, un langage XML profitant de l'universalité syntaxique de XML. Fondé sur la syntaxe de RDF/XML, OWL offre un moyen d'écrire des ontologies Web. OWL se différencie du couple RDF/RDFS en ceci que, contrairement à RDF, il est justement un

langage d'ontologies. Si RDF et RDFS apportent à l'utilisateur la capacité de décrire des classes et des propriétés, OWL intègre, en plus, des outils de comparaison des propriétés et des classes : identité, équivalence, contraire, cardinalité, symétrie, transitivité, disjonction, etc. Ainsi, OWL offre aux machines une plus grande capacité d'interprétation du contenu Web que RDF et RDFS, grâce à un vocabulaire plus large et à une vraie sémantique formelle.

II.2.2. Différentes déclinaisons de OWL

Plus un outil est complet, plus il est, en général, complexe.OWL a trois sous-langages offrant des capacités d'expression croissantes et, naturellement, destinés à des communautés différentes d'utilisateurs :

• **OWL Lite :** est le sous langage de OWL le plus simple. Il est destiné aux utilisateurs qui ont besoin d'une hiérarchie simple de concepts.

• **OWL DL :** est plus complexe que OWL Lite. Il permet une expressivité bien plus importante. OWL DL est fondé sur la logique descriptive (d'où son nom, OWL Description Logics), un domaine de recherche étudiant la logique, et conférant donc à OWL DL son adaptation au raisonnement automatisé. Malgré sa complexité relative face à OWL Lite, OWL-DL garantit la complétude des raisonnements et leur décidabilité.

• **OWL Full :** est la version la plus complexe d'OWL, mais également celle qui permet le plus haut niveau d'expressivité. OWL Full est destiné aux situations où il est plus important d'avoir un haut niveau de capacité de description, quitte à ne pas pouvoir garantir la complétude et la décidabilité des calculs liés à l'ontologie. OWL Full offre cependant des mécanismes intéressants, comme par exemple la possibilité d'étendre le vocabulaire par défaut de OWL.

Il existe entre ces trois sous-langages une dépendance de nature hiérarchique : toute ontologie OWL Lite valide est également une ontologie OWL DL valide, et toute ontologie OWL DL valide est également une ontologie OWL Full valide.

II.2.3. Document OWL

Les ontologies OWL se présentent sous forme de fichiers texte, de «documents OWL ». La création d'un document OWL fait l'objet de diverses recommandations :

II.2.3.1. Espaces de nommage de OWL

L'intégralité du vocabulaire de OWL provient de l'espace de nom de OWL, http://www.w3.org/2002/07/owl#.

II.2.3.1.1.Type MIME

Il n'existe aucun type MIME spécifique à OWL, un document OWL étant un document RDF. Il est donc recommandé d'employer le type MIME lié à RDF, à savoir application/rdf+xml ou, à la rigueur, le type MIME de XML, application/xml.

II.2.3.1.2. Extension

La recommandation du groupe de travail WebOnt indique qu'il est préférable d'employer, comme suffixe de nom de fichier, les extensions « .RDF » ou « .OWL ».

II2.3.2. Structure d'une ontologie OWL

La conception d'OWL a pris en compte la nature distribuée du Web sémantique et a intégré la possibilité d'étendre des ontologies existantes, ou d'employer diverses ontologies existantes pour compléter la définition d'une nouvelle ontologie.

II.2.3.2.1Espaces de nommage

Afin de pouvoir employer des termes dans une ontologie, il est nécessaire d'indiquer avec précision de quels vocabulaires ces termes proviennent. C'est la raison pour laquelle, comme tout autre document XML, une ontologie commence par une déclaration d'espace de nom contenue dans une balise rdf:RDF.

II.2.3.2.2. En-têtes d'une ontologie

Tout comme il existe une section d'en-tête <head>..</head> en haut de tout document XHTML bien formé, on peut écrire, à la suite de la déclaration d'espaces de nom, un en-tête décrivant le contenu de l'ontologie courante. C'est la balise owl:Ontology.

II.2.4.Eléments du langage

II.2.4.1. Les classes

Une classe définit un groupe d'individus qui sont réunis parce qu'ils ont des caractéristiques similaires. L'ensemble des individus d'une classe est désigné par le terme « extension de classe », chacun de ces individus étant alors une « instance » de la classe. Les trois versions de OWL comportent les mêmes mécanismes de classe, à ceci près que OWL Full est la seule version à permettre qu'une classe soit l'instance d'une autre classe.

La déclaration d'une classe se fait par le biais du mécanisme de « description de classe », qui se présente sous diverses formes. Une classe peut ainsi se déclarer de six manières différentes:

• **l'indicateur de classe :** La description de la classe se fait, dans ce cas, directement par le nommage de cette classe. Une classe « Humain » se déclare de la manière suivante :

<owl:Class rdf:ID="Humain" />

Il est à noter que ce type de description de classe est le seul qui permette de nommer une classe. Dans les cinq autres cas, la description représente une classe dite « anonyme », créée en plaçant des contraintes sur son extension.

• **l'énumération des individus composant la classe :** Ce type de description se fait en énumérant les instances de la classe, à l'aide de la propriété owl:oneOf.

• **La restriction de propriétés :** La description par restriction de propriété permet de définir une classe anonyme composée de toutes les instances de « owl:Thing » qui satisfont une ou plusieurs propriétés.

Ces contraintes peuvent être de deux types : contrainte de valeur ou contrainte de cardinalité. Une contrainte de valeur s'exerce sur la valeur d'une certaine propriété de l'individu (par exemple, pour un individu de la classe Humain, sexe = Homme), tandis qu'une contrainte de cardinalité porte sur le nombre de valeurs que peut prendre une propriété (par exemple, pour un individu de la classe Humain, aPourFrere est une propriété qui peut ne pas avoir de valeur, ou avoir plusieurs valeurs, suivant le nombre de frères de l'individu. La contrainte de cardinalité portant sur aPourFrere restreindra donc la classe décrite aux individus pour lesquels la propriété aPourFrere apparaît un certain nombre de fois).

• Enfin, les descriptions par intersection, union ou complémentaire permettent de décrire une classe par, comme leur nom l'indique, l'intersection, l'union ou le complémentaire d'autres classes déjà définies, ou dont la définition se fait au sein même de la définition de la classe courante :

```
<owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#etudiantsENST" />
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#aPourFrere" />
                        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
                        </owl:cardinality>
                </owl:Restriction >
        </owl:intersectionOf>
</owl:Class>
```

II.2.4.2. Héritage

Il existe dans toute ontologie OWL une superclasse, nommée Thing, dont toutes les autres classes sont des sous-classes. Ceci nous amène directement au concept d'héritage, disponible à l'aide de la propriété subClassOf. Egalement, il existe une classe nommée noThing, qui est sous-classe de toutes les classes OWL. Cette classe ne peut avoir aucune instance.

II.2.4.3.Les instances de classe

II.2.4.3.1. Définition d'un individu

La définition d'un individu consiste à énoncer un « fait », encore appelé « axiome d'individu». On peut distinguer deux types de faits :

- Les faits concernant l'appartenance à une classe

  La plupart des faits concerne généralement la déclaration de l'appartenance à une classe d'un individu et les valeurs de propriété de cet individu. Un fait s'exprime de la manière suivante :

```
<Humain rdf:ID="Pierre">
        <aPourPere rdf:resource="#Jacques" />
        <aPourFrere rdf:resource="#Paul" />
</Humain>
```

- Les faits concernant l'identité des individus

  Une difficulté qui peut éventuellement apparaître dans le nommage des individus concerne la non-unicité éventuelle des noms attribués aux individus. Par exemple, un même individu pourrait être désigné de plusieurs façons différentes.

  C'est la raison pour laquelle OWL propose un mécanisme permettant de lever cette ambiguïté, à l'aide des propriétés owl:sameAs, owl:diffrentFrom et owl:allDifferent.

II.2.4.3.2. Affecter une propriété à un individu

Une fois que l'on sait écrire une classe en OWL, la création d'une ontologie se fait par l'écriture d'instances de ces objets, et la description des relations qui lient ces instances.

II.2.4.4. Les propriétés

Maintenant que l'on sait écrire des classes OWL, il ne manque plus que la capacité à exprimer des faits au sujet de ces classes et de leurs instances. C'est ce que permettent de faire les propriétés OWL. OWL fait la distinction entre deux types de propriétés:

• les propriétés d'objet permettent de relier des instances à d'autres instances.

• les propriétés de type de donnée permettent de relier des individus à des valeurs de données.

II.2.4.4.1. Définition d'une propriété

Afin de spécifier une propriété, il existe différentes manières de restreindre la relation qu'elle symbolise. Par exemple, si on considère que l'existence d'une propriété pour un individu donné de l'ontologie constitue une fonction faisant correspondre à cet individu un autre individu ou une valeur de donnée, alors on peut préciser le domaine et l'image de la propriété. Une propriété peut également être définie comme la spécialisation d'une autre propriété.

II.2.4.4.2. Caractéristiques des propriétés

Il existe divers moyens d'attacher des caractéristiques aux propriétés, ce qui permet d'affiner grandement la qualité des raisonnements liés à cette propriété.
Parmi les caractéristiques de propriétés principales, on trouve la transitivité, la symétrie, la fonctionnalité, l'inverse, etc.…

II.3. OWL-S (Ontology Web Langage for Service)

OWL-S est un langage qui définit une ontologie de Web service basée sur le langage OWL. Les intérêts liés a l'utilisateur de OWL-S sont que ce langage inclus la sémantique et contient des fonctions indisponibles à la mise en œuvre de Web services : la description, la recherche et l'invocation. Il permet de réaliser les deux tâches suivantes :

*** La découverte automatique de Web services** :cette tâche est possible par ce que OWL-S permet d'exprimer et de résoudre des requête avec contenu sémantique par exemple des requêtes comme trouver des Web services qui vendent des ticket d'avion entre deux endroits spécifiques et qui permet de payer avec une carte de crédit particulière.

*** L'invocation automatique des Web services :** OWL-S fourni un ensemble d'APIs pour que l'invocation d'un Web service soit automatique.

II.3.1. Les structures de OWL-S

OWL-S se compose de trois parties :

II.3.1.1. Le profile de service

Pour faire la publicité et la découverte des services, les informations présentées dans le profile peuvent être séparé en 3 catégories :

- l'organisation qui fourni le service c'est à dire l'entreprise qui le publie et le commercialise sur internet.
- la fonction calculée par le service.
- les caractéristiques du service.

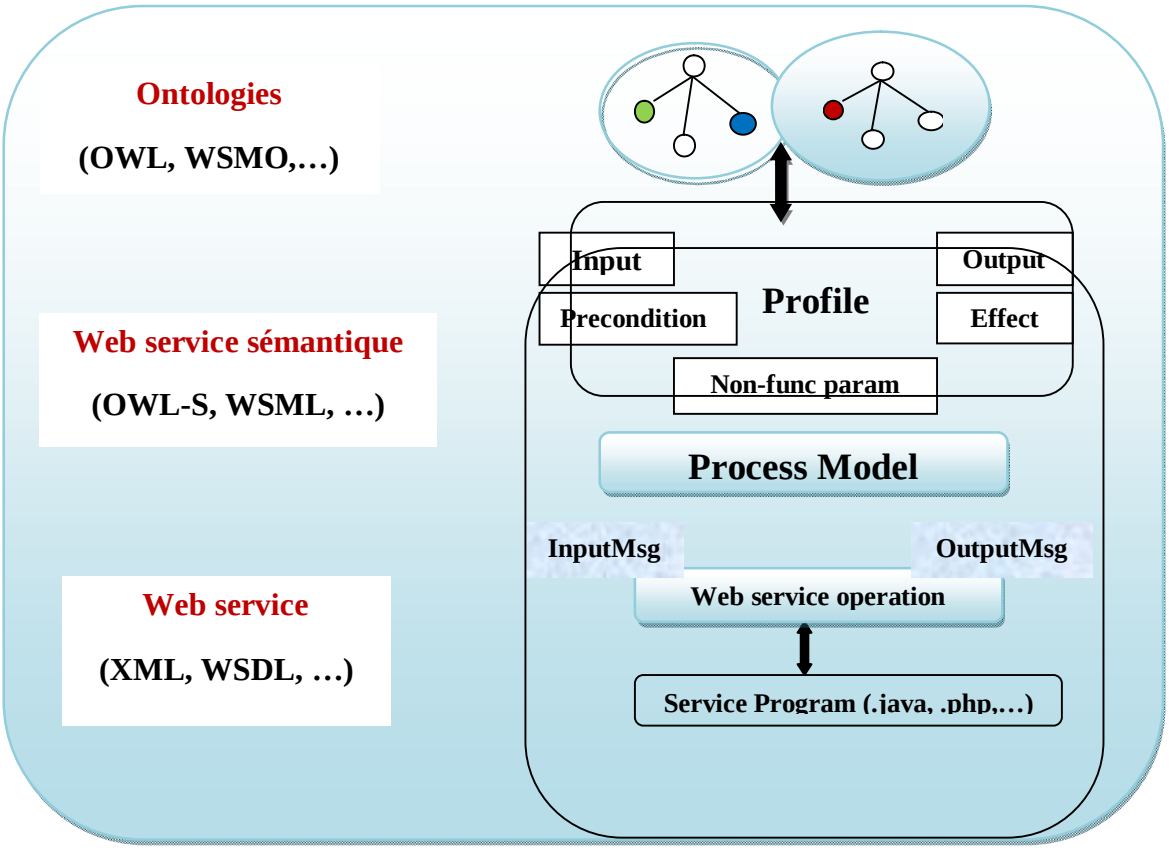

Fig.5. Web Service sémantique **[14].**

La description des fonctionnalités d'un service est indispensable au profile. Le profile représente deux aspects d'une fonctionnalité d'un service : la transformation d'une information (les inputs et les outputs), et les changements d'état après l'exécution de ce service. Aprés avoir déterminé les fonctionnalités d'un service, d'autres caractéristiques sont aussi importantes pour que le service soit découvert. Il y a en particulier le profile qui décrit les IOPEs (Input, Output, Preconditions, Effets) d'un service.

II.3.1.2. Le modèle du processus

Il donne une description détaillée d'une opération d'un service qui représente le comportement d'un service composé. Cette ontologie contient une classe processus qui détermine l'utilisation du service comme un processus. Elle permet de déterminer les entrées, les sorties, les pré-conditions et les effets .Concernant le type de processus il existe 3 catégories :

- **le processus atomique :** correspond à des services simples directement invocables et s'exécutant en une seule fois sans intervention de l'utilisateur.

- **le processus composite :** qui est décomposable en processus atomiques et composites. Il permet de spécifier l'enchainement de leurs différents composants à l'aide de structure de contrôle.

-Le processus simple : qui n'est pas invocable et qui correspond à des abstractions de processus atomiques ou composites. Les processus simples ne sont pas associés à un service grounding.

II.3.1.3. Le service « grounding »

Il fournit les caractéristiques techniques pour établir la communication avec le service au moyen des messages. Ceci permet de déterminer comment accéder au service publié. Il est nécessaire d'avoir une combinaison entre le grounding et WSDL, le fonctionnement du grounding est assuré si les deux entrées « WSDL et grounding » sont présentes et se correspondent. **[2]**

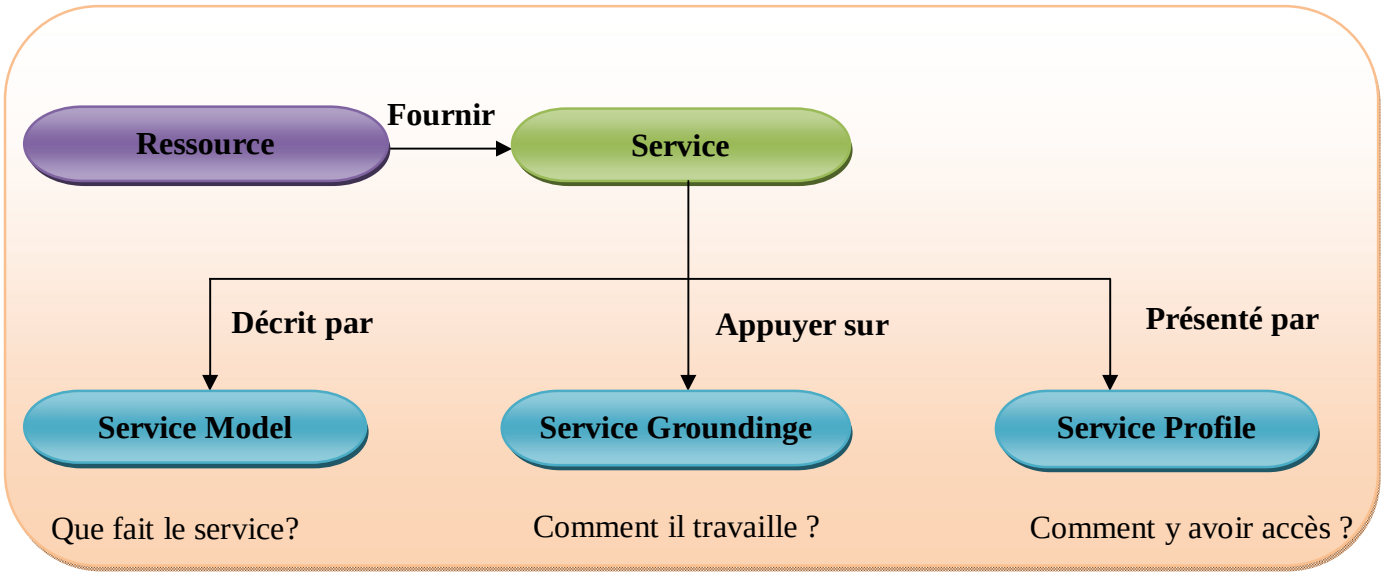

Fig.6. Niveau supérieur de l'ontologie OWL-S **[5].**

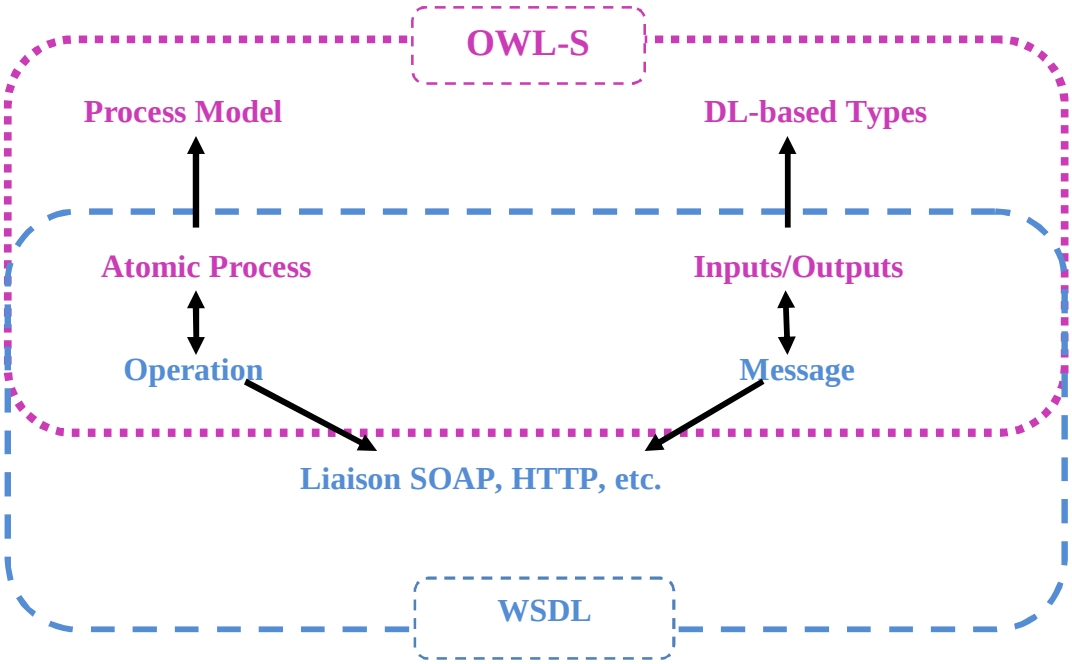

Fig.7. Relation entre OWL-S et WSDL **[13]**.

Conclusion :

Les avantages de l'utilisation du Web sémantique pour la description des Web services sont nombreux.

En plus, de rendre l'interface du Web service accessible automatiquement par des machines, la planification et l'utilisation des ontologies des Web services, et plus précisément OWL-S, permettent d'accroître l'efficacité dans la composition de Web services. Dans ce contexte, la notion d'ontologie en général peut jouer un rôle très important pour associer une sémantique à la description d'un Web service, et surtout pour la composition dynamique des Web services sémantiques.

Nous utilisons la planification et l'ontologie OWL-S pour proposer une solution parmi un ensemble, pour résoudre le problème de la composition dynamique des services Web sémantiques, et plus particulièrement qui fera l'intérêt du prochain chapitre

Introduction :

Ce chapitre s'occupera dans la première partie de l'utilisation de la planification par intelligence artificielle pour produire un plan de composition dynamique de Web services. Ce chapitre se chargera également de présenter le Framework OWLS-XPlan utilisé pour la planification

Partie 1 : La Planification par Intelligence Artificielle

Introduction :

La planification par intelligence artificielle (IA) est probablement le contexte qui offre le plus d'outils opérationnels pour la composition de Web services. La raison derrière cela est une analogie simple qu'on peut faire entre un problème de composition de Web services et un problème de planification.

III.1.1. Définition d'un problème de planification

Un planificateur IA produit un ensemble d'actions ordonnées ayant pour but la modification de l'état actuel du « monde » afin d'obtenir l'état désiré du « monde ». Le problème de planification est vu comme un problème de construction automatique ou semi-automatique d'ordre sur les actions disposées de sorte que l'exécution de ces actions a pour but de modifier l'état initial du monde en un autre état appelé état final du monde dans lequel certains buts ont été réalisés. Cet ordre est typiquement assimilé à un "ordre partiel". En effet, l'ordre retrouvé peut être assimilé à un ordre partiel où des séquences d'actions sont effectuées en parallèle.
Un problème de planification est définit par un 5-uplet $< S_0, S, G, A, \Gamma >$ tel que :

S : ensemble de tous les états possibles du monde.

$S_0 \subset S$: représente l'état initial du monde.

$G \subset S$: l'état but que le système tente d'atteindre.

A : l'ensemble des actions qu'un planificateur peut exécuter. Chaque action fait transiter le monde de l'état actuel vers un état futur.

$\Gamma \subset SxAxS$: la fonction de transition. Elle définit la pré-condition et les effets de l'exécution de chacune des actions.

Le principe d'un planificateur est de faire transiter le monde de l'état S_0 vers l'état G en utilisant la fonction de transition Γ.

III.1.2. Analogie avec un problème de composition de services

Une analogie simple peut être faite entre un problème de composition de Web services et un problème de planification. Cette analogie est comme suit :

Problème de composition de services	Problème de planification
Requête de l'utilisateur	S_0
Ensemble des Web services disponibles	Ensemble (A) des actions
Entrées (input) d'un Web service	Pré-condition de l'action correspondante
Sorties (output) d'un Web service	Effets de l'action correspondante

Tableau n°1: Analogie entre la composition de WS et la planification

On peut donc suggérer de transformer un problème de composition de Web services en un problème de planification et utiliser un planificateur existant pour générer un plan de composition.

Un problème de composition de Web services est généralement décrit en utilisant le langage OWL-S. Cette description représente la spécification externe du problème. D'un autre côté un problème de planification est souvent décrit en utilisant le langage PDDL. Cette description représente la spécification interne du problème. La figure 8 décrit le principe d'utilisation de la planification par intelligence artificielle pour proposer une solution à un problème de composition de Web services.

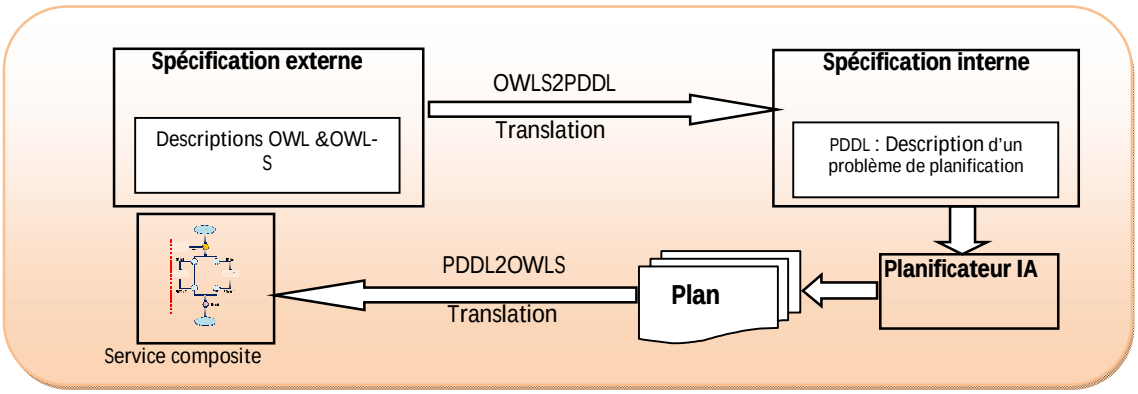

Fig.8. L'utilisation d'un planificateur par IA pour la composition de Web services **[19]**

III.1.3. Description du langage PDDL

Le langage de PDDL permet de décrire un domaine d'application et un problème à résoudre. Un domaine d'application se compose des actions paramétrées qui caractérisent des comportements d'un objet spécifique et de prédicats décrits par des propositions logiques.

Une action est caractérisée par un nom, une pré-condition et un effet. Une pré-condition et un effet sont décrits par des prédicats **[19]**

Un problème est caractérisé par des objets, un état initial et un but. Un état initial est décrit par une liste de prédicats qui ont la valeur vraie et un but est aussi décrit par une proposition logique formulée par des prédicats comme une pré-condition.

La planification est réalisée par un planificateur qui commence par l'état initial et applique des actions convenables pour atteindre le but.

PDDL (Planning Domain Definition Language) est un langage standard de codage pour les tâches de planification.

La description d'un problème en utilisant PDDL obéit à une syntaxe bien définie. La structure générale d'une description PDDL correspondant à un problème de planification est la suivante :

a) Nom du problème (problem<name>) : le nom attribué au problème à traiter.
b) Nom du domaine (:domain<name>) : ce nom fait référence à une description du domaine en relation avec le problème en question.
c) Besoins du problème (problemrequirements<require-def>) : tous les besoins requis dans le problème sont définis à ce niveau.
d) Objets (<objectdeclaration>) : il s'agit des objets concrets qui peuvent être substitués aux paramètres des prédicats et créer des faits dans les tâches de planification.
e) Etat initial (<init>) : description de l'état initial à partir duquel commencent les actions de planification.
f) Etat but (goal>+) : description de l'état que le planificateur tente d'atteindre en utilisant les actions permises.

II.1.3. Exemple sur un fragment du plan PDDL

```
<plan>
<step number="0" name="http://127.0.0.1/health-scallops/services/CreateMedicalTransportAccount.owl#CreateMedicalTransportAccountService" id="24" new="false" ps="1">
<param name="http://127.0.0.1/health-scallops/services/CreateMedicalTransportAccount.owl#CompanyData">http://www.owl-ontologies.com/Ontology1.owl#EMA</param>
<param name="http://127.0.0.1/health-scallops/services/CreateMedicalTransportAccount.owl#DesiredAccountData">http://www.owl-ontologies.com/Ontology1.owl#DesiredAccount</param>
<param name="http://127.0.0.1/health-scallops/services/CreateMedicalTransportAccount.owl#CreditcardInformation">http://www.owl-ontologies.com/Ontology1.owl#EMACC</param>
<effects>
<effect id="24">
```

<adds>
<fact name="**http://127.0.0.1/health-scallops/ontology/MedicalTransportCompany_Ontology.owl#ValidAccount**" id="**51**">
<param>**http://www.owl-ontologies.com/Ontology1.owl#DesiredAccount**</param>
</fact>
</adds>
<deletes />
<preconditions />
</effect>
</effects>
</step>
<step number="**1**" name="**http://127.0.0.1/health-scallops/services/CreateMedicalFlightAccount.owl#CreateMedicalFlightAccountService**" id="**22**" new="**false**" ps="**2**">
<param name="**http://127.0.0.1/health-scallops/services/CreateMedicalFlightAccount.owl#CompanyData**">**http://www.owl-ontologies.com/Ontology1.owl#EMA**</param>

Conclusion :

Plusieurs travaux de recherche ont utilisé le principe de la figure 8 pour proposer un plan de composition de Web services. Nous nous intéressons à l'outil « OWLS-XPlan » **[8, 19]** que nous avons utilisés dans le cadre de ce projet. La section suivante donne une description brève de l'outil en question.

Partie 2: La présentation de OWLS-XPlan.

Introduction

L'objectif de cette section est de présenter un outil de composition dynamique de Web services baptisé « OWLS-XPlan » **[8]**. Cet outil a été développé dans un cadre de recherche et mis à la disposition des chercheurs du domaine sous forme de boîte noire, mais également sous forme de boîte blanche. Le principe général de fonctionnement de OWL-XPlan est celui décrit par la figure 9 ci-dessous.

Les techniques de planification de l'intelligence artificielle peuvent contribuer à la résolution du problème de composition de Web services. Ce dernier peut être défini comme un problème de planification. En effet ; les services sont modélisées comme des actions et la composition comme un plan de connections des Web Service. Dans un premier temps, le planificateur construit le plan solution en explorant les Web services disponibles. Ensuite, il développe les différents états intermédiaires à partir de l'état initial, en appliquant les services, jusqu'à atteindre l´état but. Enfin, il retourne soit un plan qui est un séquence de services permettant d'atteindre le but, soit la situation d´échec.

La figure 9 décrit le principe de résolution d'un problème de composition de Web service en utilisant la plate-forme OWLS-XPlan.

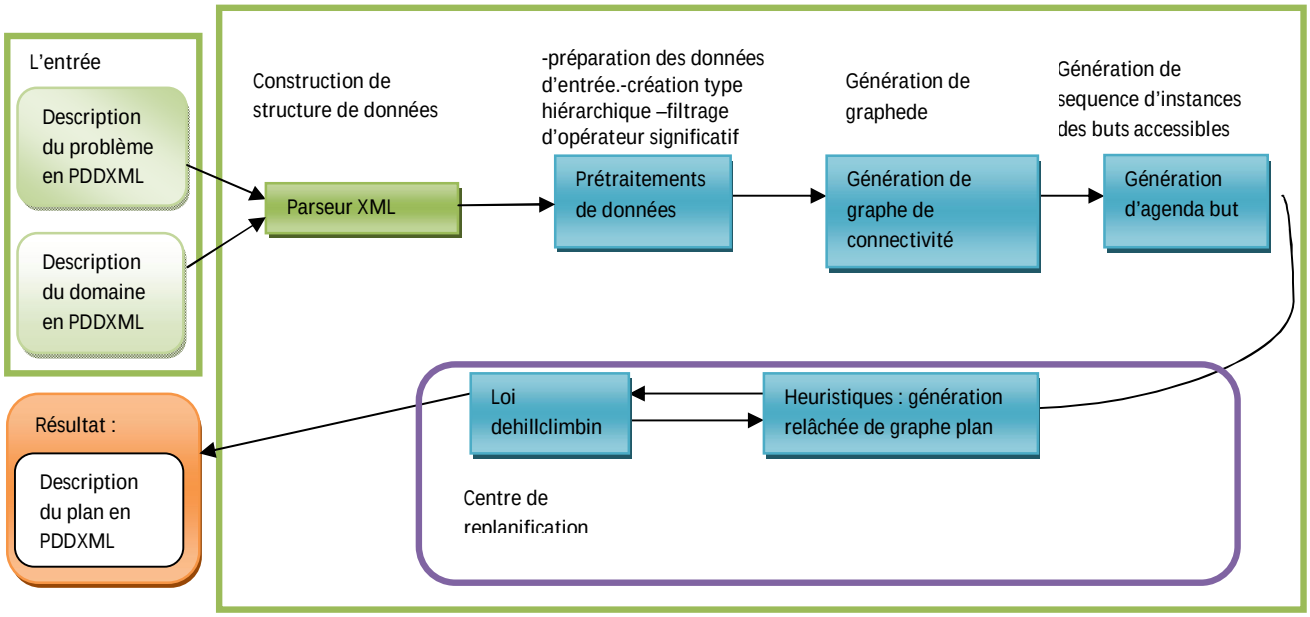

Fig.9. Résolution d'un problème de composition de Web service en utilisant OWLS-XPlan. **[8]**

III.2.1. Convertisseur OWLS2PDDL

La conversion d'OWL-S 1.1 des descriptions de service en PDDXML exige non seulement la transcription simple des types et des propriétés aux prédicats PDDL mais la cartographie de services à des actions. Pour cela, nous décrivons seulement le processus de traduction indispensable.

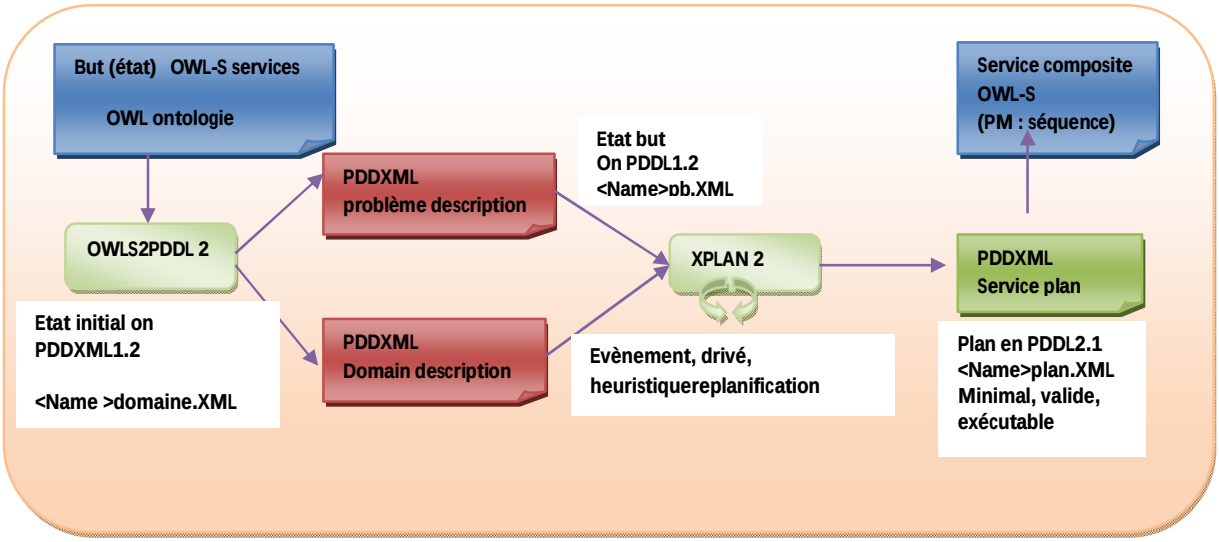

Fig.10. Architecture OWLS-XPlan. **[12]**

Tout paramètre OWL-S du profil de service d'entrée est corrélé avec une action équivalente en PDDL, et le paramètre de service « *hasPrecondition* » peut directement être transformé à la condition préalable de l'action par utilisation des prédicats. La même chose pour le paramètre condition « *hasEffect* ». Soit ce service existe déjà, donc sa traduction fait partie de la description du domaine de planification, ou, comme service demandé (requête) devient partie intégrante de la description du problème de planification.

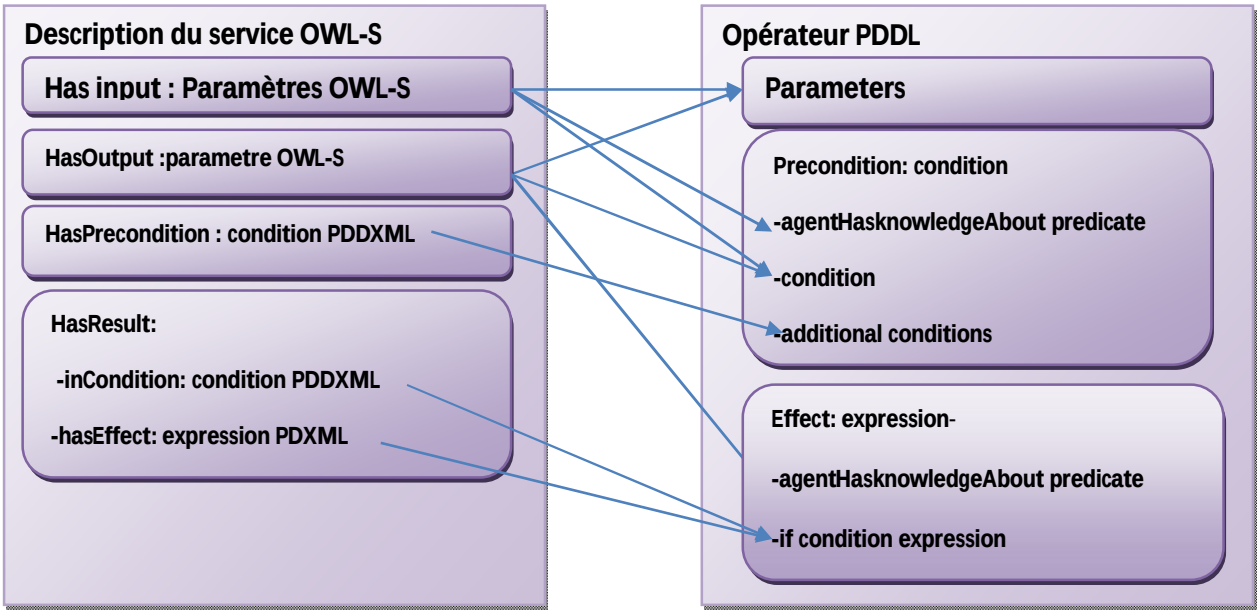

Fig.11 : Conversion de services OWL-S vers actions PDDL. **[8]**

III.2.2. Génération du plan

Les algorithmes de planification peuvent être groupés en deux grandes familles. D'un côté, les algorithmes de planification dans un espace d´états qui sont les algorithmes les plus simples ; de l'autre côté, les algorithmes de planification dans un espace de plans.

III.2.2.1. L'algorithme Graph-plan

Cet algorithme effectue une procédure proche de l'approfondissement itératif, en découvrant une nouvelle partie de l'espace de recherche à chaque itération. La taille du graphe de planification jusqu'à l'état k et le temps nécessaire pour le développer jusqu'à cet état sont polynomiaux en la taille du problème de planification. Cependant, la mémoire requise pour les données peut être un facteur limitant significatif. Ceci est dû à l'application des services qui créent des nouveaux objets dans chaque niveau. Enfin Graph-plan a une double complexité : horizontale et verticale, causée respectivement par le graphe de planification et les objets crées.

III.2.2.2. FF (Fast plan generation through Heuristic search)

FF est un planificateur basé sur la recherche par chainage avant dans l'espace d'états développé en appliquant les techniques de relaxation du Graph-plan. C'est-à -dire en ignorant les effets négatifs des actions. L'idée est de commencer l'exécution de Graph-plan sur le problème de planification, et d'extraire un plan relaxé. Ensuite, ce plan est utilisé pour évaluer

une nouvelle stratégie de recherche qui informe la recherche de la distance estimée vers l'état but. Une autre heuristique locale qui utilise la technique de « *hill-climbing* » est aussi définie. L'objectif de son utilisation est d'atteindre le but avec peu d'évaluation car les évaluations d'états sont couteuses. En effet, elle consiste à toujours sélectionner les meilleurs successeurs dans l'état courant. Les heuristiques ont montré leur efficacité à élaguer l'espace de recherche dans différents domaines d'applications. La génération du graphe de recherche est combinée avec une méthode de recherche « *hill-climbing* » pour élaguer l'arbre. L'information sur la qualité de l'action est utilisée par la recherche locale pour choisir parmi plusieurs choix qui sont également pondérés par les heuristiques. **[9]**

III.2.3. Composition dynamique de services en utilisant le réseau de tâches HTN

Un réseau hiérarchique de tâches (HTN) se compose des tâches primitives et non primitives. Une tâche primitive est exécutable tandis qu'une tâche non primitive ne peut pas être exécutée directement parce qu'elle est composée de tâches primitives et non primitives. Les outils de planification qui incorporent les algorithmes de planification se nomment planificateurs. STRIPS (STanford Research Institute Problem Solver) est un algorithme de planification classique conçu par Richard Fikes et Nils Nilsson en 1971. La planification de HTN consiste à réaliser partiellement ou entièrement une liste de tâches ordonnées (réseau de tâches). Un planificateur de HTN prend un problème qui est défini par un réseau de tâches comme une entrée et décompose les tâches non primitives en tâches primitives pour les exécuter.

Dans cette approche, les Web services sont modélisés comme des processus. On peut identifier trois types de processus : atomique, simple et composite. Le processus atomique est un modèle en une étape du point de vue client. Son exécution consiste en une simple invocation avec ses paramètres d'entrée. Le processus simple est aussi modélisé par une seule étape, mais contrairement à un processus atomique, il est possible d'accéder à l'intérieur de sa structure, ou de le remplacer par une expansion. Enfin, le processus composite représente une composition de Web services, c'est-à-dire qu'elle est décomposable en d'autres processus atomiques, simples ou composites. Dans l'ontologie des processus, chaque processus dispose de plusieurs propriétés; à savoir : inputs (optionnels), pré-conditions, outputs (conditionnels), effets (conditionnels). Les pré-conditions et les effets sont décrits en OWL-S. On peut constater que la décomposition de tâches dans la planification de HTN est très semblable à la décomposition de processus composites dans l'ontologie de processus OWL-S.

III.2.4. Principe de fonctionnement de OWLS-XPlan

L'outil de composition dynamique des Web services OWLS-XPlan compose de plusieurs modules de prétraitement et de planification. Il prend comme entrée un ensemble de services OWL-S, des ontologies, et d'une planification demande (but) ; il retourne une séquence de planification de services pertinents qui satisfait l'objectif. Pour ce faire, il convertit d'abord l'ontologie du domaine de descriptions des services OWL-S, aux descriptions de problème et de domaine PDDL utilisant le convertisseur intégré OWLS2PDDL.

La description de domaine contient la définition de tous types, les prédicats et les actions, alors que la description du problème comprend tous les objets, l'état initial, et l'état but. Ces deux descriptions sont ensuite utilisées par le planificateur XPlan pour créer un plan PDDL qui résout le problème donné dans le domaine réel. Pour des raisons de commodité, ils ont développé un dialecte XML de PDDL, appelé PDDXML qui simplifie l'analyse, la lecture et communiquant les descriptions PDDL à l'aide de SOAP.

Un opérateur de domaine de planification correspond au service profil dans le langage OWL-S. La clé de la traduction de OWL-S pour PDDXML est que tout service en OWL-S correspond à une action également nommé avec le même ensemble de paramètres d'entrée et les conditions logiques. Il combine recherche locale guidée à la planification graphique détendue, et une forme simple des réseaux de tâches hiérarchiques (HTN) pour produire une séquence d'actions que le plan résout un problème donné. S'il est équipé avec des méthodes, XPlan utilise uniquement les parties des méthodes de décomposition qui sont nécessaires pour atteindre l'état-but avec une séquence de services composés.

III.2.4.1. L'exécution de l'outil OWLS-XPlan

L'outil de composition dynamique des Web services OWLS-XPlan compose de plusieurs modules de prétraitement et de planification. Il prend comme entrée un ensemble de services OWL-S, des ontologies, et d'une planification demande (but) ; il retourne une séquence de planification de services pertinents qui satisfait l'objectif.

Pour ce faire La figure suivante montre brièvement chaque étape de fonctionnement de l'outil OWLS-XPlan :

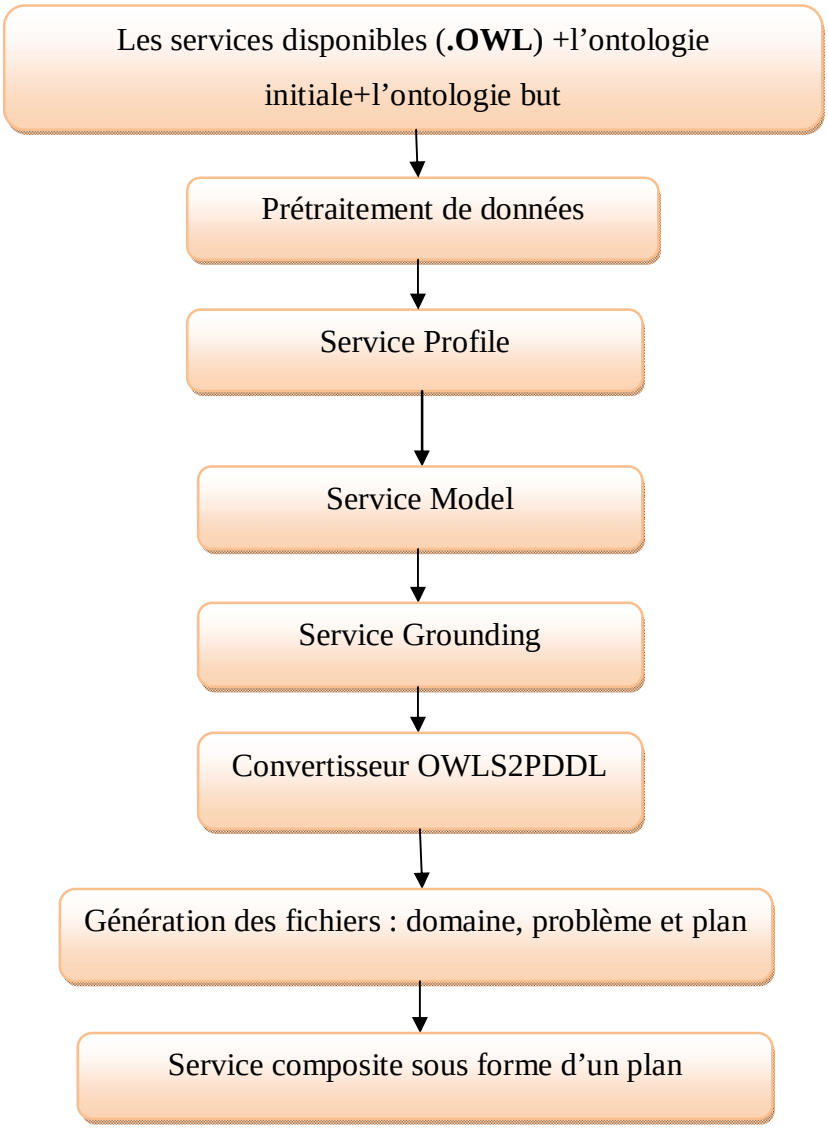

Fig.12 les étapes d'exécution de OWLS-XPlan

III.2.4.2. Explications des différentes étapes :

a)- Le planificateur OWLS-XPlan prend en entrée un ensemble de services décrits en OWL-S, une description de l'état initial et l'état but.

b)- Le prétraitement des données concernant l'importation des fichiers standards depuis le site.

c)- Service Profile qui spécifie des informations concernant le service tels que : nom de service, le texte de description, et le contact d'information.

d)- Service « *Model* » qui spécifie les entrées et sorties pour chaque processus (service). Les transitions d'un état à un autre sont décrites par les pré-conditions et les effets de chaque processus. Un model peut être décrit par le lien « *descriptedBy* », un model de service qui expose comment un service fonctionne**.**

e)- Service « *Grounding* » Définit les détails techniques permettant d'accéder au Web service, tels les protocoles, les URIs, les messages envoyés, . . . etc.

Les deux classes**« *ServiceProfile* »** et **« *ServiceModel* »** d'une description OWL-S s'attachent à abstraire la représentation d'un service Web. Par contre, la classe **«*ServiceGrounding* »** est la forme concrète d'une représentation abstraite.

f)- convertisseur OWLS2PDDLil convertit d'abord l'ontologie du domaine de descriptions des services OWL-S qui contient la définition de tous types, les prédicats et les actions, aux descriptions de problème et de domaine PDDL qui comprend tous les objets, l'état initial, et l'état but. Ces deux descriptions sont ensuite utilisées par le planificateur XPlan pour créer un plan PDDL qui résout le problème donné dans le domaine réel. Pour des raisons de commodité, ils ont développé un dialecte XML de PDDL, appelé PDDXML qui simplifie l'analyse, la lecture et communiquant les descriptions PDDL à l'aide de SOAP.

g)- pour la génération du plan Il combine recherche locale guidée à la planification graphique détendue, et une forme simple des réseaux de tâches hiérarchiques (HTN) pour produire une séquence d'actions que le plan résout un problème donné. S'il est équipé avec des méthodes, XPlan utilise uniquement les parties des méthodes de décomposition qui sont nécessaires pour atteindre l'état-but avec une séquence de services composés.

Conclusion :

Nous avons vu dans ce chapitre la planification par l'intelligence artificielle, le Convertisseur OWLS2PDDL et le principe de fonctionnement de l'outil OWL-S XPlan pour générer le plan de composition de façon dynamique .pour convertir ce plan de OWLS vers BPEL en utilisant des règles de transformation d'un métalangage vers un autre métalangage. Le chapitre suivant se charge d'expliciter cette transformation.

Introduction :

L'intégration de services a été proposée par des entreprises. Celles-ci ont développé plusieurs technologies telles que WSCI (Web Service Choreography Interface), WSFL (Web Service Flow Language) et plus récemment BPEL (Business Process Execution Language for Web Services). Cette dernière est un langage qui combine le langage XLANG proposé par Microsoft et le langage WSFL proposé par IBM. BPEL apparaît donc comme la résultante des approches précédentes. D'autre part, BPEL correspond à la combinaison des deux langages WSCI et BPML (Business Process Modelling Language).
Plusieurs plateformes (Netbeans, Eclipse, …) intègrent le moteur d'exécution BPEL ce qui permet d'avoir un environnement d'exécution pour un Web service composite. Rappelons qu'un service exprimé en OWL-S n'est pas sous forme exécutable. Dans cette section nous présenterons une approche qui traduit un service composite exprimé en OWL-S en un service équivalent exprimé en BPEL. La réalisation de cette translation permettra donc d'exécuter des services composites générés par OWLS-XPlan.

IV.1. Business Process Execution Language

Les Web Services Basé sur le langage XML pour la spécification formelle des processus d'affaires et les protocoles d'interaction. Un fichier BPEL permet l'utilisation du fichier WSDL de l'implication des services. Par conséquent, BPEL peut être vu comme une extension de WSDL.

Il s'agit d'un langage commun normalisé ayant pour objectif l'optimisation de la gestion des processus métiers de plus en plus complexes. Le langage BPEL a été développé pour fournir une spécification formelle des processus métiers et des interactions correspondantes. Ce langage s'appuie sur une notation très structurée pour définir un processus et ses interactions.
L'objectif est basé sur les techniques de modèles de transformation et de défis.
La figure montre les interactions entre les activités et les services

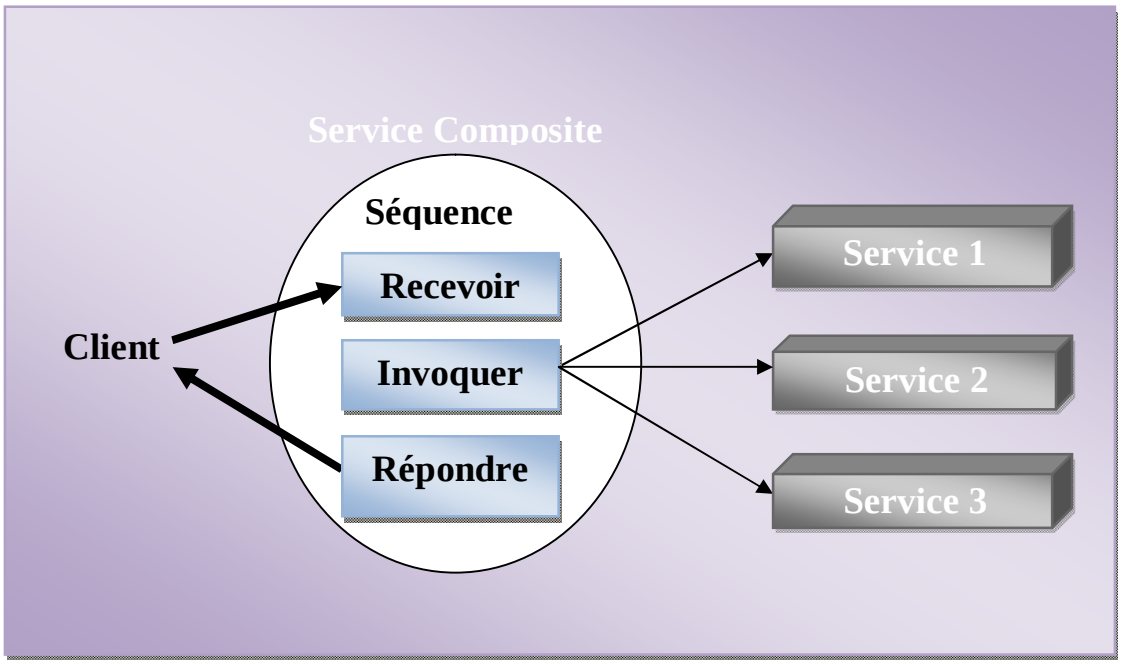

Fig.13. Vision interne des activités de BPEL4WS **[16].**

IV.2. Comparaison entre BPEL et OWL-S

Dans ce chapitre, nous avons délimité les descriptions de compositions de services en deux approches : OWL-S et BPEL. Quelques différences évidentes subsistent mais les concepts restent similaires. Ces deux langages de compositions ont tout d'abord besoin de créer les services pour ensuite les utiliser. Dans ce cas, le même service peut être utilisé par les deux langages sans aucun changement. La manière dont le service a été créé n'est pas importante et cela garantit le côté distribué du système, et plus précisément de transparence. Le processus d'exécution est similaire dans ces deux langages car ils utilisent une interface qui fait appel à un service abstrait, composé ou atomique. En traitant les services composés, la composition doit être vue par l'utilisateur comme un service unique.

Dans OWL-S, un des atouts les plus importants, concerne la capacité de découvrir les Web services sans ambigüité. Ceci est rendu possible par l'utilisation de la sémantique afin de décrire des services. En effet, BPEL demande plus d'efforts pour le traitement automatique de la découverte.

Il est important de mentionner que plusieurs auteurs ne font pas de comparaison entre BPEL et OWL-S, mais d'une manière générale effectue une comparaison entre BPEL et OWL-S Model. Ils considèrent en effet que le langage BPEL est responsable du processus de la

composition des services, alors que, dans OWL-S, la composition est assurée par OWL-S Model.

Les IOPE (Input, Output, Pré-condition, Effet) sont un atout réel présent dans OWL-S.

BPEL contient la description des entrées et des sorties ; par contre, les pré-conditions et les effets ne sont pas pris en charge. Les IOPE sont présents dans OWL-S, et plus particulièrement dans l'ontologie Profile du service.

La plupart des structures utilisées par BPEL et OWL-S sont similaires. Ce que BPEL appelle activités structurées, OWL-S appelle constructeur de contrôle, mais les deux sont des mécanismes utilisés pour coordonner les Web services puis les exécuter.

IV.3. Simple Transformer (SiTra)

Pour la translation de OWL-S vers BPEL nous nous sommes inspiré sur le framework « SiTra » **[17]** qui définit des règles abstraites pour la translation d'un langage A vers un langage B. Nous avons appliqué le principe de transformation pour notre cas particulier (OWL-S vers BPEL).

IV.3.1. Principe de la transformation

Le modèle de transformation se compose de deux grandes étapes :

a) La première étape consiste à définir et à préciser le modèle de transformation. Cela est une tâche complexe impliquant le domaine de connaissance significatif et la compréhension à la fois des domaines source et modèle cible. Par exemple, la définition du modèle de transformation de OWL-S pour BPEL exige non seulement une compréhension des deux langages, mais également une approche analytique pour découvrir la cartographie correcte entre les éléments du modèle.

b) La deuxième étape consiste à exécuter la transformation. Actuellement, l'exécution élégante du cahier des charges est encore un sujet de recherche dans de nombreux cas et peut nécessiter une intervention manuelle importante afin de fournir une mise en œuvre correcte.

L'architecture de SiTra est représentée dans la figure 14 Une transformation spécifie comment les éléments du méta-modèle de la source sont mappés dans les éléments du méta modèle de

la destination. Un cadre de transformation, crée un modèle de destination, qui est une instance de méta-modèle de destination, à partir d'un modèle de source, qui est une instance du méta-modèle source

La figure ci-dessous montre la transformation SiTra qui est fournie par un certain nombre de classes Java. La classe SiTra et l'interface correspondante « *Transformer* » sont utilisées pour exécuter la transformation définie sur un modèle source en particulier. Les deux interfaces simples de la transformation des règles en Java sont présentées ci-dessous **[17]**.

```
interface Rule<S,T> {
        boolean check(S source);
        T build(S source, Transformer t);
        void setProperties(T target, S source, Transformer t);
}
```
[17]

```
interface Transformer {
        Object transform(Object source);
        List<Object> transformAll(List<Object> sourceObjects);
        <S,T> T transform(Class<Rule<S,T>> ruleType, S source);
        <S,T> List<T> transformAll(Class<Rule<S,T>> ruleType,List<S> source);
}
```
[17]

IV.3.2. Règles de Transformation

Le problème de la transformation est subdivisé en plusieurs règles. Une classe qui implémente cette interface doit être écrite pour chacune des règles de la transformation. Les méthodes de cette interface sont décrites comme suit:

a) La mise en œuvre de la méthode de contrôle doit retourner la valeur « true » si la règle est applicable à l'objet source. Ceci est particulièrement important si plusieurs règles sont applicables. Cette méthode est utilisée pour distinguer laquelle des règles multiples doit être appliquée par le transformateur.

b) La méthode de compilation devrait construire un objet cible résultat de la transformation de l'objet source. Une chaîne récursive des règles ne doit pas être invoquée dans cette méthode.

c) La méthode « *setProperties* » est utilisé pour définir les propriétés de l'objet cible. La définition des propriétés est séparée de la construction de la cible de sorte que l'appel récursif des règles est possible lors de la définition des propriétés. S'il est impossible de distinguer entre les règles multiples en utilisant la méthode de contrôle, l'invocation

de règles explicites doit être utilisée pour transformer des objets pour appliquent les règles multiples.

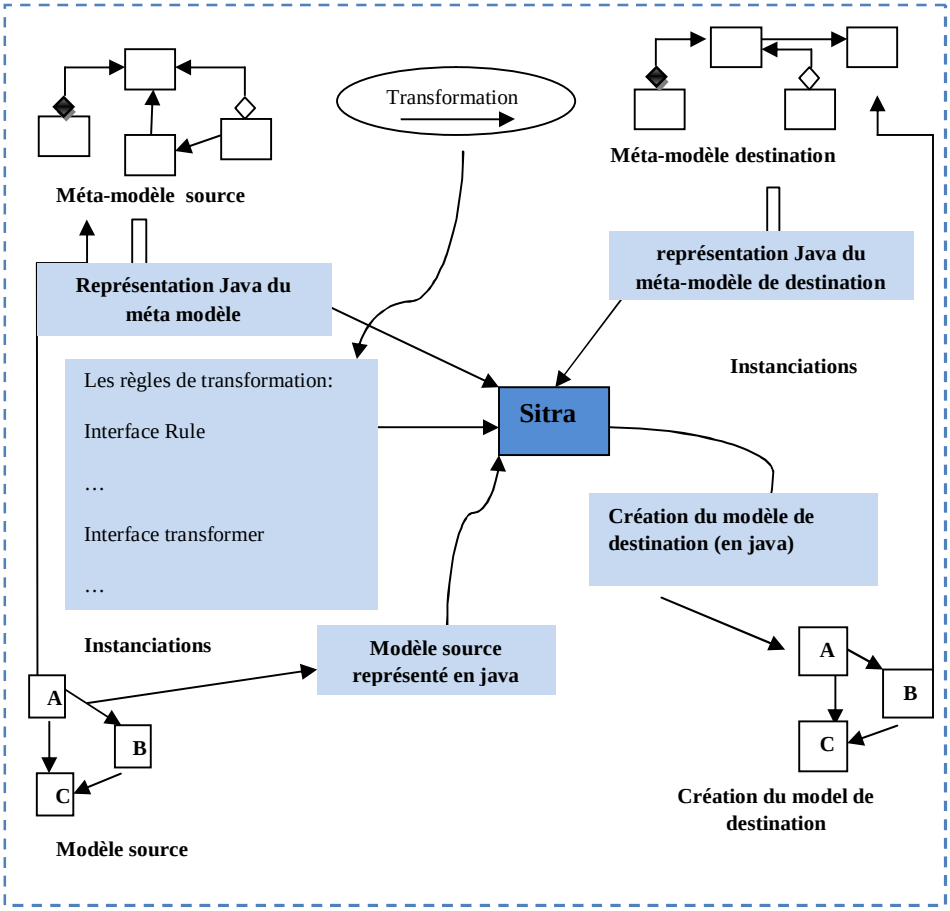

Fig.14. Vue générale du principe de transformation **[18]**

IV.3.3. Transformateur

Pour instancier une transformation, les classes de règles doiventt être ajoutées à une instance de la classe « *SimpleTransformer* ». La transformation peut alors être exécutée en appelant le procédé transformé avec l'objet de racine (s) du modèle de source. L'algorithme parcourt toutes les règles afin de vérifier quelle règle peut être appliquée à quelques objets de source.

```
FOR EACH rule
    IF rule.check(source) THEN
        IF notRecorded(source, rule) THEN
            target = rule.build(source, this)
            record(source, target, rule)
                rule.setProperties(source, this)
```

IV.3.4. Méta-modèle de OWL-S

Un processus OWL-S est une spécification de la manière dont un client peut interagir avec un service. Un processus donne une perspective détaillée sur la façon d'interagir avec un service. La figure suivante représente les divers attributs de processus OWL-S

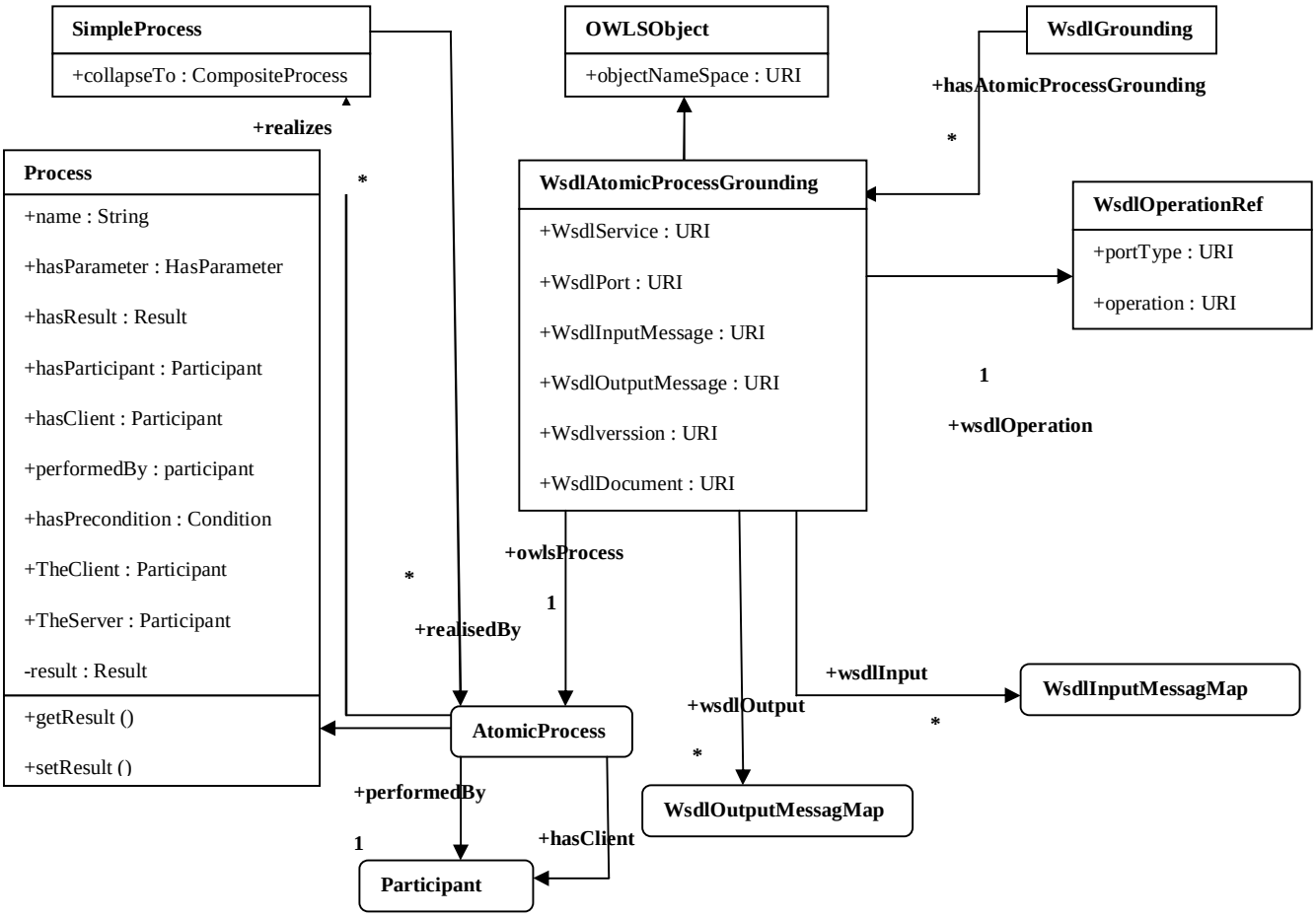

Fig.15. Partie du Méta-modèle OWL-S. **[18]**

IV.3.5. Méta-modèle de BPEL

La spécification BPEL peut être représentée par un méta-modèle compatible qui est représentée sur la figure 16. Le méta-modèle spécifie un certain nombre éléments du modèle qui sont équivalents à des constructions XML, définition de l'activité de divers types, qui permettent le traitement séquentiel, conditionnel, répétitif ou parallèle d'actions. Il définit un certain nombre de fonctionnalités, comme les variables, le contexte d'exécution et des exceptions permettant la création de processus complexes et réalistes, effectuer styles divers et d'invocation des manipulations de données d'une manière algorithmique.

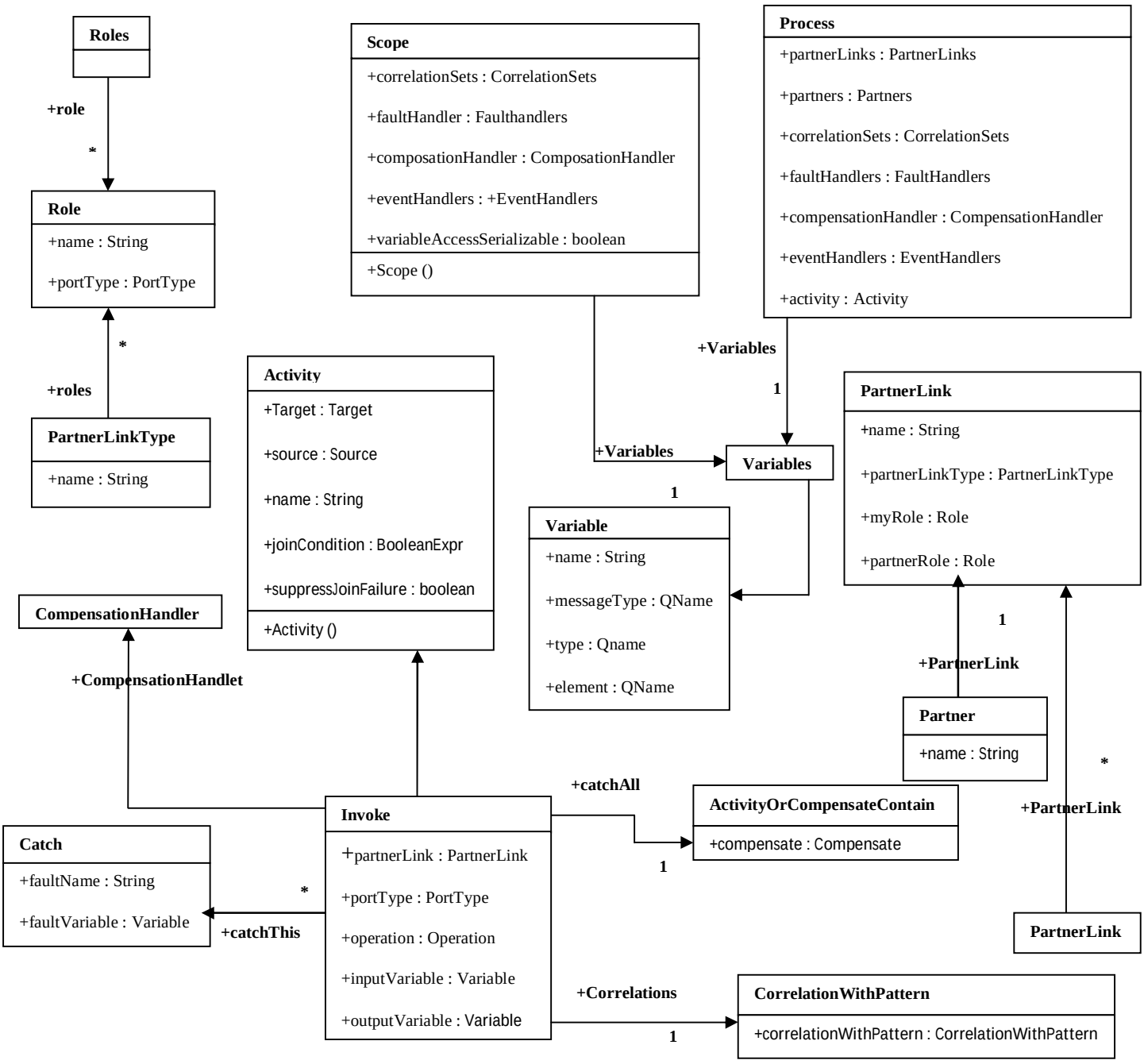

Fig.16. Partie du Méta-modèle BPEL. **[18]**

IV.4. Les règles de transition

Dans un processus atomique OWL-S, l'élément qui représente la classe la plus fondamentale des processus de Web services, est associé à un processus BPEL. La cartographie exige également la création d'un certain nombre de BPEL et d'autres éléments WSDL de sorte que le sens du modèle de sortie corresponde entièrement à celui de l'entrée.

IV.4.1 Translation du « Service Model » vers BPEL:

OWL-S Process	**BPEL**
<Process>	<Process>
<AtomicProcess>	<Process>+ <PortType> + <Operation> + <PartnerLinkType> + <Role> + <invoke> Operation + PortType names must be consistent with the created WSDL file. **Note:** require an <invoke> call within <Process> tags and Operation + PortType names must be consistent with the created WSDL file.
[<input>]* (of Atomic Process)	<input> + <message> + <variable>
<input>	<part> (of the <message> created for all inputs)
[<outputs>]* (of Atomic Process)	<output> + <message> + <variable>
<output>	<part> (of the <message> created for all outputs).
<precondition>	**Note:** This does not really map, due to complicated representation of Preconditions in OWL-S.
<result>	**Note:** This contains semantic information about the output of a process and does not map usefully to any BPEL class. The <with-Output> property will be covered by the mappings described above.
<participant>, <SimpleProcess>, <CompositeProcess>	<partner>
[<input>]* of Composite Process	<message> + <variable>
<input>	<part> (of the message created for all inputs)
[<outputs>]*of Composite Process	<message> + <variable>
<output>	<part> (of the message created for all outputs).
<ControlConstruct>	<Activity> (See below)
<sequence>	<sequence>
<Iterate>, <RepeatUntil>, <RepeatWhile>	<while>
<AnyOrder>	Model as <sequence>
<Split>	<flow>
<Choice>	Model as <switch>
<IfThenElse>	<switch>
<components> of a sequence, split... etc.	See below...
<Binding> (as in <hasDataFrom> property) toParam, valueSource, theVar, fromProcess	<variable> (or part thereof) corresponding to that generated for the given process and parameter.
<valueSpecifier>, <SplitJoin>, <Perform>, <Produce>	-

Tableau n°2 : Translation de modèle de processus OWL-S vers des éléments BPEL. **[18]**

IV4.2. Translation du « Service Profile » d'OWL-S vers BPEL:

OWL-S profile	**BPEL**
<input>, <output>	See Process Section
<profile>, serviceName, textDescription ContactInformation	Incorporate into process name(s). Include as plain text.
<precondition>, <result>, <parameter>, <ServiceParameter>, <ServiceCategory>, <ServiceClassification>, <ServiceProduct>,	-

Tableau n°3 : Translation de service Profile OWL-S vers des éléments BPEL. **[18]**

IV.4.3. Translation du « Service Grounding » d'OWL-S vers BPEL:

OWL-S (Grounding)	**WSDL/BPEL**
<OWLsProcess>	Maping to BPEL dealt with by Process Model.
<WSDLOperation>	<wsdl :Operation>
URI : wsdlPortType	<wsdl :PortType>
URI : wsdlService	<wsdl :Service>
URI : wsdlPort	<wsdl :Port>
URI : wsdlInputMessage	<wsdl :Message>
<wsdlGrounding>, <wsdlInputMessageMap>, URI : owlsParameter, URI : xsltTransformation	-
URI : wsdlMessagePart	<wsdl :Part>
URI : wsdlVersion	(<wsdl verssion=''…''>)
URI : wsdlDocument	Name of wsdl doc/targetnamespace.

Tableau n°4 : Translation équivalente de service Grounding OWL-S vers des éléments WSDL/BPEL. **[18]**

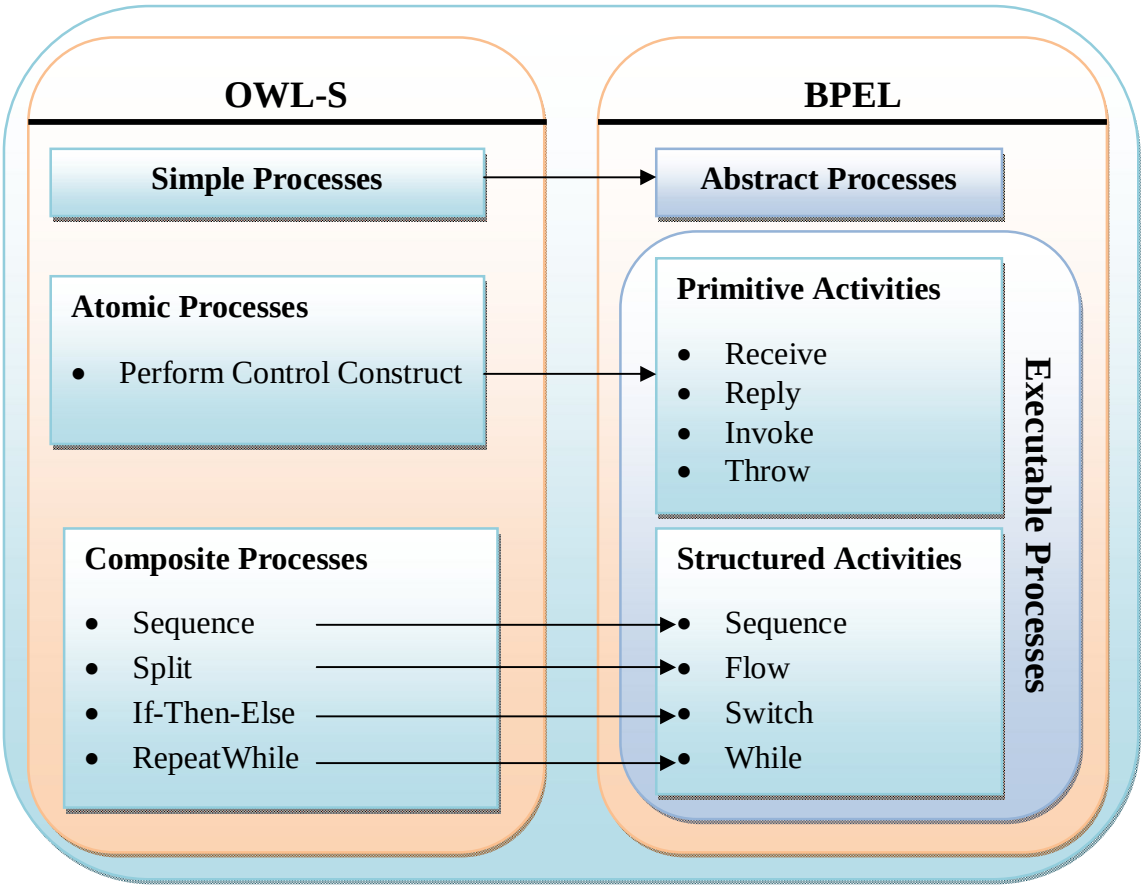

Fig.17. translation d'une méthode OWL-S vers BPEL. **[15]**

IV.5. Les outils d'implémentation

IV.5.1.Xampp : XAMPP est un ensemble de logiciels permettant de mettre en place facilement un serveur Web, un serveur FTP et un serveur de messagerie électronique. Il s'agit d'une distribution de logiciels libres (X Apache MySQL Perl PHP) offrant une bonne souplesse d'utilisation, réputée pour son installation simple et rapide. Ainsi, il est à la portée d'un grand nombre de personnes puisqu'il ne requiert pas de connaissances particulières et fonctionne, de plus, sur les systèmes d'exploitation les plus répandus.

IV.5.2. Eclipse : Eclipse est un projet de la Fondation Eclipse visant à développer tout un environnement de développement libre, extensible, universel et polyvalent. Son objectif est de produire et fournir divers outils gravitant autour de la réalisation de logiciel, englobant les activités de codage logiciel proprement dites mais aussi de modélisation, de conception, de test, etc. Son environnement de développement notamment vise à la généricité pour lui permettre de supporter n'importe quel langage de programmation.

Le projet Eclipse est pour cela organisé en un ensemble cohérent de projets logiciels distincts, sa spécificité tenant à son architecture totalement développée autour de la notion de plugin: toutes les fonctionnalités de l'atelier logiciel doivent être développées en tant que plug-in bâti autour de l'IDE Eclipse Platform.

IV.5.3. OWLS-XPlan :

IV.5.3.1. L'icône de préparation : La figure suivante montre la partie préparation qui concerne l'ontologie initiale, l'ontologie but et l'ensemble des services disponibles :

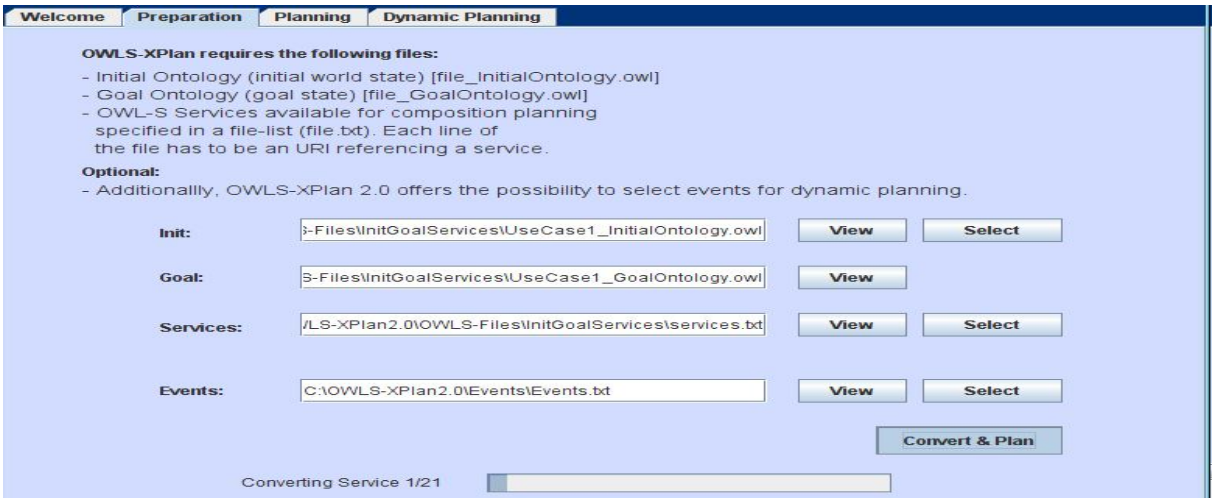

Fig.18. L'icône de préparation

IV.5.3.2. Génération du plan :

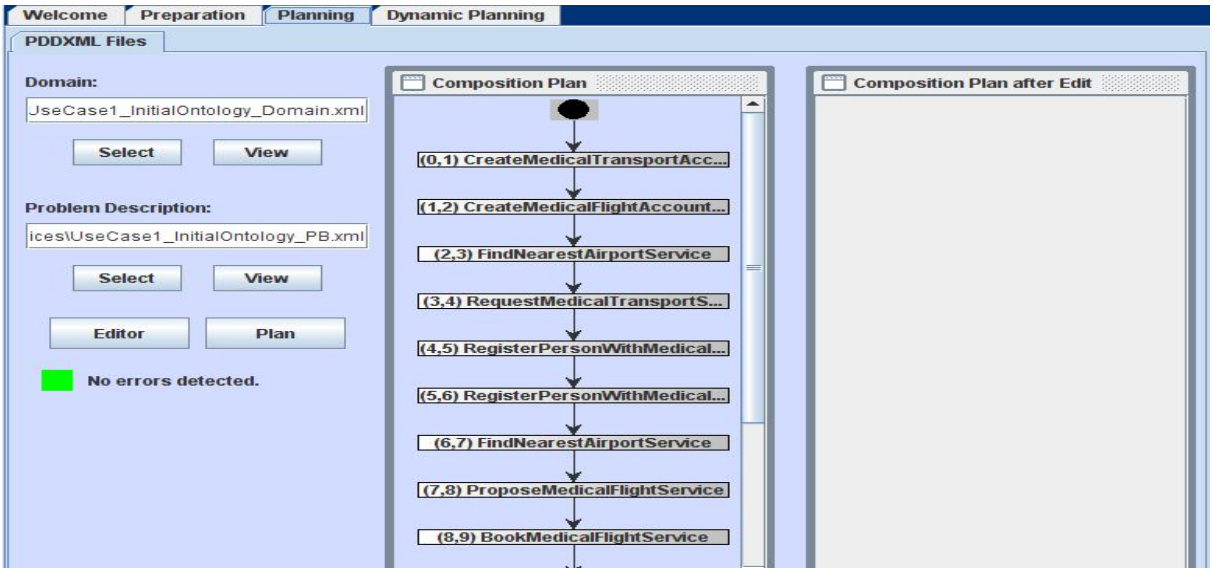

Fig. 19.L'icône génération du plan

Et cette figure correspond au plan généré sous cmd :

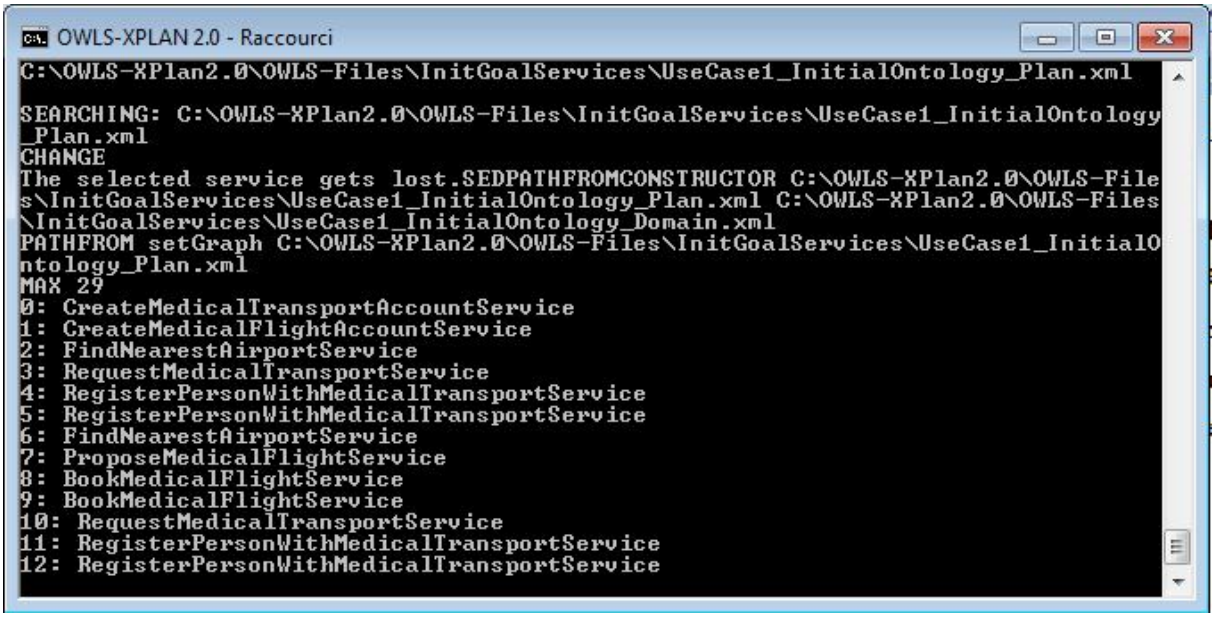

Fig.20. Service composite sous forme d'un plan.

IV.6. L'Architecture générale de l'application

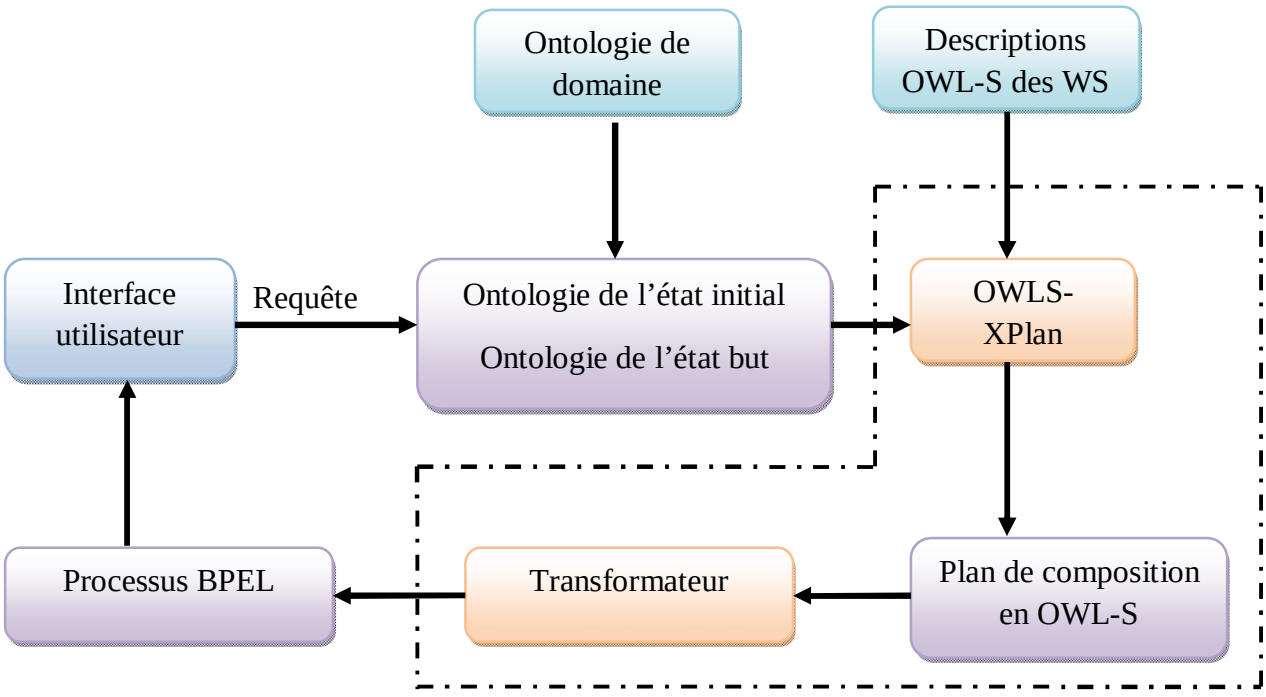

Fig.21. Architecture de l'application

Le principe de fonctionnement du système est le suivant :

a) Une requête de l'utilisateur est formulée à travers une interface. Cette requête est généralement introduite à travers un formulaire adapté au domaine.

b) En utilisant l'ontologie du domaine et les termes employés par l'utilisateur, la requête est traduite en un problème de planification. Ce problème est alors caractérisé par les descriptions de l'état initial et de l'état but.

c) Une fois le problème de planification formulé, l'outil OWLS-XPlan peut être lancé afin de construire un plan possible pour atteindre l'état but. Le plan généré est ensuite exprimé en OWL-S. Il faut savoir aussi que cette étape (génération du plan) peut ne pas aboutir et retourner un échec.

d) Le transformateur reçoit le service composite exprimé en OWL-S et utilise les règles de transformation pour générer un plan BPEL qui constitue le résultat du système.

L'étape suivante sera l'exécution du processus BPEL généré. IL appartient à l'utilisateur de choisir l'environnement d'exécution adéquat à cet effet.

Conclusion :

Le but de ce chapitre était de compléter le résultat obtenu par l'outil OWLS-XPlan en proposant une transformation en un processus BPEL. Cette approche permet de fournir un service composite sous forme directement exécutable ce qui est très important du point de vue de l'utilisateur. Nous avons implémenté une partie des règles de transformation pour montrer la faisabilité de l'approche. La suite logique serait de continuer avec toutes les règles de transformation afin de fournir un outil complet. Ceci nécessite davantage de temps et surtout d'expérimentation.

Conclusion générale :

L'objectif de ce projet était de s'attaquer au problème de composition de Web services, et en particulier la composition dynamique : c'est-à-dire la génération d'un plan de composition au moment où la requête de l'utilisateur est exprimée. Comme le domaine de composition dynamique est très vaste, nous nous sommes restreints à l'utilisation de la planification par intelligence artificielle pour la génération d'un service composite. Ce choix est largement justifié par le fait que la planification par intelligence artificielle est sans doute le domaine qui a produit le plus d'outils opérationnels dans le domaine de composition dynamique de services.

Afin de réaliser la composition dynamique nous nous sommes appuyés sur l'outil OWLS-Xplan réalisé dans un contexte identique au nôtre. Une grande partie des efforts fournis a été consacrée à la compréhension du principe de fonctionnement de l'outil en question afin de pouvoir l'intégrer dans notre travail. En effet, nous utilisons OWLS-Xplan comme une boîte blanche en manipulant directement le code mis à notre disposition.

Après avoir été en mesure de produire un plan de composition dynamique nous nous sommes intéressés à rendre ce plan exécutable ce qui permettra une utilisation effective de notre travail. Pour cela, nous nous sommes inspirés des travaux qui réalisent une traduction d'un langage source vers un langage cible. Dans notre cas le langage source est OWLS-Xplan et le langage cible est BPEL. En se basant sur les méta-modèles des deux langages, des règles de traduction peuvent être dégagées et utilisées pour réaliser la translation. Dans cette partie nous avons adapté l'outil baptisé « SiTra » afin de traiter le cas de translation de OWL-S vers BPEL.

La réalisation de ce projet n'a pas été sans difficultés. Nous pouvons citer en premier plan la quantité importante de connaissances qu'il était nécessaire d'aborder. En effet, nous nous sommes attaqués à un domaine nouveau et en pleine expansion. Les notions de composition de services, de Web services sémantiques, de planification par intelligence artificielle, parmi d'autres, sont nouvelles pour nous et nous ont demandé beaucoup de lectures et de synthèses.

Nous pouvons aussi citer des difficultés d'ordre technique qui ont entouré la réalisation de ce travail. En effet, nous avons utilisé des outils existants développés dans un cadre de recherche et il était d'bord difficile de comprendre le fonctionnement (en boîte blanche) et ensuite de réaliser une intégration afin d'atteindre notre objectif.

Ce projet a été une occasion d'apprentissage intense tant sur le plan des connaissances théoriques que sur le plan pratique. Nous avons en particulier appris comment utiliser des

travaux existants et les intégrer dans notre travail. Ce dernier point est très important car de nos jours il ne s'agit plus de réécrire des projets existants mais de comprendre ce qui a été fait par d'autres équipes, le critiquer et d'utiliser ce qui est utilisable ; tout cela dans un souci d'efficacité.

Nous avons pu expérimenter notre travail sur des exemples pratiques mais il faut diversifier les domaines d'application afin de valider ce travail.

Beaucoup de travail reste à faire en particulier dans la partie traduction de OWL-S vers BPEL car le nombre de règles de traductions est très important et la mise en œuvre de toutes ces règles demande davantage de temps et d'expérimentation.