UNIVERSITÉ DU QUÉBEC À MONTRÉAL

UN ENVIRONNEMENT DE COMPOSITION DE SERVICES WEB

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR

HACÈNE MECHEDOU

AOÛT 2007

TABLE DES MATIÈRES

LISTE DES FIGURES

RÉSUMÉ :

La composition des services web, en tant que moyen de fournir des applications à valeur ajoutée a engendré un grand intérêt aussi bien dans le milieu industriel que dans la communauté des chercheurs. Dans le premier cas, l'accent a été mis sur la définition des standards basés sur le langage XML. Elle est de nature exclusivement syntaxique. Dans le deuxième cas, on aborde la composition sous une approche basée sur le web sémantique et la description de services est basée sur des informations sémantiques. Les deux approches ayant été développées séparément bien que l'objectif que l'on veut atteindre est le même à savoir la fourniture d'outils de composition automatique de services.

Dans ce mémoire, nous proposons une approche « mixte » qui essaye de rassembler les avantages des deux méthodes précédentes. L'originalité de la solution consiste à utiliser les standards industriels de la composition tout en utilisant un système à base de connaissances modélisant les contraintes de la composition de services. Les mécanismes utilisés sont les règles exprimées par la logique d'ordre 1 avec les concepts orientés objet. Nous avons réalisé un prototype qui a montré la concrétisation des idées avancées afin de mettre en place un système de composition pouvant être utilisé pour des applications réelles. Le prototype est illustré par un exemple provenant de l'industrie de la distribution et du transport électrique.

MOTS-CLÉS : services web, composition, système à base de connaissances, Jess, wsdl, bpel4ws

CHAPITRE I

INTRODUCTION GÉNÉRALE

1.1 Introduction

L'un des soucis majeurs des entreprises, est le partage et l'échange de données ainsi que des informations entre les différents acteurs, qui interviennent dans la réalisation des différentes activités. La collaboration de partenaires (d'autres compagnies ou organisations) est devenue une nécessité pour satisfaire et fidéliser la clientèle. L'environnement de communication et de collaboration prend une importance particulière en ce sens qu'il soulève des défis technologiques de plus en plus importants.

Plusieurs innovations technologiques ont eu lieu pour répondre à ce besoin de partage et de communication. Ce qui a conduit à remettre en cause constamment l'architecture des applications et la recherche de la meilleure façon de développer du logiciel.

Il y a eu trois vagues technologiques ayant apporté des améliorations importantes à l'architecture et au processus du développement de systèmes distribués (figure 1.2).

Architecture Client/Serveur : Une application est modélisée par un ensemble de services fournis par des serveurs et un ensemble de clients utilisant ces services. Les clients et les serveurs sont des processus différents. La répartition des processus est basée sur un modèle logique car les processus ne sont pas associés nécessairement à des processeurs différents [29]. Figure 1.1

En général une application client/serveur est structurée en trois couches : présentation, traitement et données.

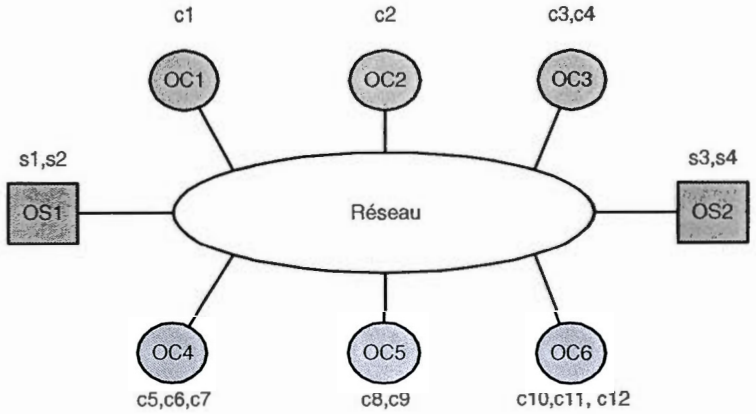

Figure 1.1 Architecture client / serveur.

Légende :

OS : Ordinateur Serveur

OC : Ordinateur Client

c : processus client

s : processus serveur

Architecture N-tiers : Cette architecture élargit l'idée de « Client/Serveur » pour répartir les différents modules d'une application sur plusieurs couches, lesquelles communiquent par le modèle RPC (Remote Procedure Call). Les technologies supportant ce modèle d'architecture sont COM/DCOM pour les plates formes Windows et EJB/RMI pour des applications bâties en Java. Elles présentent toutes les deux l'inconvénient d'être des solutions propriétaires, i.e., le développement doit se faire uniquement par l'une où l'autre (COM/DCOM ou EJB/RMI). L'avantage (entre autres) est d'assurer le traitement des transactions qui permet de gérer la queue des requêtes et de leurs priorités. Ce modèle s'applique pour des applications distribuées sur un réseau local (LAN). CORBA avait pour objectif d'appliquer le principe de répartition à un réseau large(WAN) et de faire communiquer des modules développés dans différentes plate formes et différents langages.

Par ailleurs, CORBA opère sur des ports qui ne sont pas ouverts par les firewalls dus aux contraintes de sécurité; ce qui cause des problèmes de déploiement.

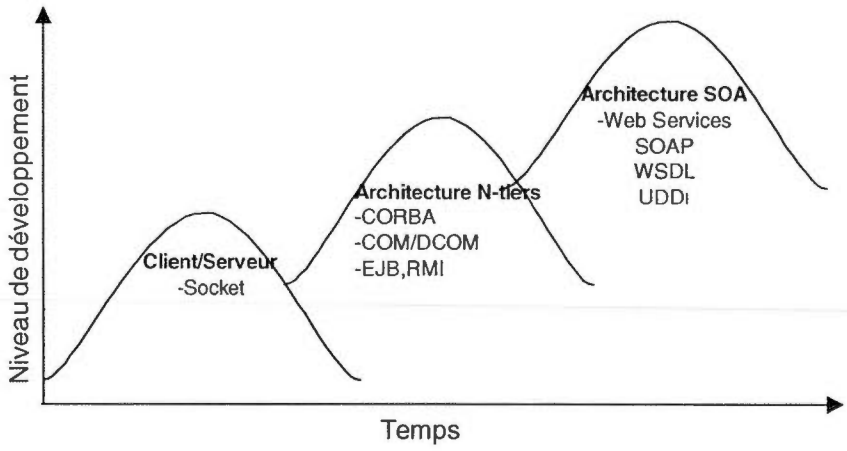

Figure 1.2 Évolution de l'architecture des applications

L'architecture SOA (Service Oriented Architecture) : Ce modèle utilise le concept de service comme base de développement d'applications. L'échange entre ces services se fait par le biais de mécanismes standards.

Une implémentation de cette architecture est basée sur la technologie des services web. L'apparition du XML (Extensible Markup Language), qui possède la propriété de décrire son contenu et qui est structuré sous-forme d'une arborescence a apporté un changement majeur dans les modèles de communication des applications. L'interprétation d'un document XML est donc facilitée par l'utilisation des schémas et des DTDs (Document Type Definition). Cette architecture est basée sur l'utilisation de services comme entité de construction d'un système. L'ensemble des services communiquent par des messages SOAP (Simple Object Access Protocol). Les services sont décrits par WSDL (Web Services Description Language).

XML constitue la base des protocoles utilisés par les services web (SOAP, WSDL, UDDI). Nous reviendrons sur les détails des services web au chapitre 2. SOA est donc basée sur les services web. La communication se fait à travers le web (HTTP) et du coup les

contraintes posées par les firewalls sont levées (contrairement à CORBA). Cette architecture ouvre des perspectives très prometteuses au développement du logiciel. C'est précisément dans ce contexte que s'inscrit notre travail à savoir la composition de services web.

1.2 La composition de services

L'idée et la motivation derrière la composition de services web est de fournir un environnement de développement d'applications de qualité avec de moindres efforts. Les applications doivent s'aligner aux processus d'affaires qui, à leur tour, doivent être flexibles pour s'adapter aux changements. Les processus d'affaires subissent des changements continus afin de répondre aux exigences des entreprises qui veulent satisfaire les besoins des clients en leur offrant un service personnalisé et de qualité. Ce qui permet de les fidéliser.

La nécessité d'automatiser les processus d'affaires et de les adapter continuellement et rapidement aux nouvelles exigences des clients, permettra aux entreprises d'atteindre leurs objectifs. Cette automatisation est rendue possible grâce aux nouveaux paradigmes et standards essentiellement basés sur XML.

L'intégration des applications nécessitent de plus en plus d'interopérabilité et d'échange de données, et ce indépendamment des plateformes utilisées et des langages avec lesquels elles sont programmées.

Le concept de réutilisation en génie logiciel, qui est amplement justifié dans le domaine d'intégration des applications et des applications B2B (business-to-business), est précisément supporté par les services web. En ce sens, ces derniers permettent d'envelopper des applications existantes et de les faire coopérer afin de répondre aux nouveaux besoins des entreprises moyennant le minimum de développement de logiciel possible. Mieux encore, l'interopérabilité des services web se fait à travers le web et ouvre de nouvelles perspectives dans le domaine de développement de logiciels.

Après avoir exploité l'architecture basée sur les composantes (COM/DCOM, EJB) qui se limite à une même plate-forme et dont le développement d'applications se fait de manière statique (manuellement par les développeurs), les services web rendent possible le

développement « dynamique » (automatiquement par la machine) grâce aux différents standards basés sur XML (UDDI, WSDL, SOAP).

Par ailleurs, ce développement « dynamique » relève encore de nouveaux défis car il nécessite de définir de nouveaux standards permettant de composer(ou d'assembler) les services web de manière automatique. Ces standards à leur tour nécessitent des outils pour les générer et les interpréter afin de pouvoir construire des applications à valeur ajoutée

La composition de services apporte un élément de réponse à l'intégration des applications et l'automatisation des processus.

1.3 Intérêts et motivations

1.3.1 Réduction du temps du développement

Le développement du logiciel a toujours été un souci majeur des entreprise en ce sens qu'il occasionne des coûts élevés et nécessitant des ressources humaines qualifiées non seulement en techniques de programmation mais aussi dans le domaine de compétence auquel elles veulent apporter des solutions informatiques en automatisant certaines tâches. L'expertise est donc un point clés pour la réussite de projets d'envergure. Cette expertise n'étant pas disponible et souvent chère à acquérir nécessite le recours à des services de tierces parties (partenaires). D'autre part, dans ce genre de projets, plusieurs domaines de compétences interagissent pour répondre aux besoins des clients. C'est précisément à ce niveau que l'utilisation des services web peut jouer un rôle prépondérant à partir du moment où ces derniers permettent de mettre à la disposition d'une entreprise des services spécialisés dont elle a besoin. Ainsi la construction d'applications basée sur ces services spécialisés permet d'atteindre les objectifs de l'entreprise rapidement et de répondre aux besoins de ses clients avec le moindre coût.

1.3.2 Fourniture de services à valeur ajoutée

Beaucoup de services sont disponible sur Internet et prêts à être utilisés par des entreprises à travers le web. Ces services parfois existent au sein d'une même entreprise dans différents départements et donc peuvent être utilisés sans limitations en termes de choix technologiques. Et ainsi chaque département préserve son indépendance et optimise l'utilisation de ses ressources pour se concentrer sur son domaine de compétence en offrant des services de haut niveau et donc à valeur ajoutée.

1.3.3 Disponibilité d'un nombre de plus en plus grand de services

Le nombre de services web ne cesse d'augmenter et leur nature (leur valeur ajoutée) est très variée; ce qui les rend de plus en plus utiles et nécessaires pour le développement de nouvelles applications. Le support des processus d'affaires par des services est rendu possible puisqu'il ne dépend plus directement des différents services utilisés. Si un service donné n'est plus disponible, il est facilement remplacé par un autre avec la même qualité sinon meilleure que celle du premier. La création d'applications à partir des services offre de nouvelles perspectives en termes d'innovation dans le domaine de développement du logiciel car il est maintenant possible de faire abstraction des détails spécifiques. Il existe toujours un service web que l'on peut utiliser. Ce qui permet de se concentrer sur l'objectif global de l'application.

1.3.4 Existence de standards XML pour la composition

Les standards XML liés aux services web en général et à la composition de services en particulier offrent des outils de planification, de définition et d'implémentation pour la composition de services. Nous allons voir, par la suite, que même l'existence des standards relève encore de nouveaux défis pour trouver des solutions au problème de la composition. Peu d'outils sont offerts pour supporter ces standards.

1.4 Problématique

Les processus d'affaires des entreprises évoluent constamment en subissant des changements liés à la conjoncture économique d'une part et aux exigences des clients d'autre part. Les clients veulent recevoir des services de plus en plus personnalisés et ce le plus rapidement possible.

Par conséquent les entreprises doivent disposer d'une grande flexibilité au niveau de leurs processus d'affaires. Ces derniers doivent donc être adaptés pour répondre aux objectifs que l'on veut atteindre. Le maximum d'activités nécessite l'automatisation des tâches qui les composent. L'interaction mutuelle des activités est également un facteur déterminant dans l'atteinte des objectifs. Habituellement, les activités correspondent à des applications logicielles qui communiquent et interagissent entre elles et en échangeant des données et des messages. Il s'agit alors d'interopérabilité et d'intégrations des applications. Vu sous cet angle le problème d'automatisation du processus d'affaires peut être ramené à celui de la composition des services web.

La composition de services web fait l'objet de plusieurs travaux aussi bien dans la communauté de recherche qu'au niveau industriel [1]. Il existe deux approches qui ont respectivement évolué de façon indépendante. En industrie, beaucoup de standards XML et de spécifications ont été définis. L'accent est mis sur la spécification du flux d'exécution des processus et de composition de services. En revanche, la communauté scientifique se préoccupe plus de la sémantique liée à la composition de services et utilise les systèmes d'inférences basés sur les connaissances de nature déclarative permettant de composer des services. Dans les deux cas il y a un manque d'outils pour la composition.

Dans ce travail, nous essayons de construire un outil permettant la composition de services en nous inspirant des techniques et des standards utilisés dans les deux approches précédentes. Nous pensons qu'il est possible de faire un outil d'aide à la génération d'applications en composant des services web.

1.5 Objectifs

L'objectif est de construire un outil de composition de services servant à générer des applications. L'outil doit être suffisamment simple pour être utilisé par des développeurs ne connaissant pas nécessairement les détails de services mis en contribution dans le processus de composition.

Cet outil doit être construit sur la base des standards industriels existants de telle sorte que les plans de composition générés peuvent être portables sur les engins d'interprétation des plans de composition.

L'outil de composition utilisera une base de connaissance dont la maintenance est assurée par un spécialiste de la composition. La base de connaissances évoluera dans le temps en enrichissant son contenu et sa qualité dépendra des services web modélisés. Les services dont la qualité n'est pas satisfaisante seront remplacés en fonction des besoins.

1.6 Méthodologie

Notre méthodologie comporte les étapes suivantes :

1. Étude des systèmes et des approches existants dans le domaine de la composition de services.

2. Choix d'un standard de description de services web.

3. Utilisation d'un système à base de connaissances basé sur JESS comme outil de composition.

Pour atteindre les objectifs cités dans la section précédente, nous avons étudié les deux approches qui existent dans la littérature :

- D'une part, l'approche industrielle se base uniquement sur des standards XML mais ne présente aucun moyen pour la composition automatique de service. Le principe est que la composition doit se faire manuellement, i.e., par une personne qui doit spécifier toutes les

conditions et contraintes liées au processus d'affaires. Cette façon de procéder est purement syntaxique.

- D'autre part, en web sémantique, on utilise les systèmes à base de connaissances. L'expertise liée à la composition est modélisée dans la base de connaissances sous forme de règles d'inférences. Chaque règle représente un service.

Dans le tableau suivant nous présentons la comparaison des deux approches :

Critères de comparaison	**Approche industrielle**	**Approche sémantique web**
Standards de spécification des services web	WSDL (Web Services Description Language)	RDF (Resource Description Framework)
Standards de composition des services web	BPEL4WS (Business Process Execution Language for Web Services) WSCI (Web Service Choreography Interface)	DAML-S (Darpa Agent Markup Language Services)
Standards d'exécution	SOAP (Simple Object Access Protocol)	
Nature de la description des services	Syntaxique	Sémantique
Génération de protocole d'interaction	Manuelle	Automatique par des systèmes orientés objectifs
Faiblesse	Description incomplète des fonctionnalités	Difficulté d'exprimer les SW sous forme d'objectifs à atteindre

Tableau 1.1 Comparaison des approches : industrielle et sémantique web

Dans le cadre de ce mémoire, nous avons utilisé une approche mixte. Elle consiste à utiliser les standards XML de l'industrie pour la description et la composition des services. Il s'agit donc de WSDL et du BPEL4WS respectivement. La génération du plan est faite par un système à base de connaissances comme dans les systèmes utilisés dans l'approche web sémantique.

Justification des choix

Nous pensons, que l'utilisation d'un système à base de connaissances est possible car nous pouvons limiter à un domaine donné l'ensemble des services web à utiliser pour la composition. La difficulté à décrire les services web en termes d'objectifs (voir tableau 1.1) est levée par cette hypothèse. D'autre part, la gestion de la base de connaissances par un expert de la composition est faite de manière transparente aux utilisateurs de l'outil.

Le choix des standards XML (WSDL, BPEL4WS) est fait pour les raisons suivantes :

- Nous pensons que de plus en plus de services web à valeur ajoutée seront disponibles sur le web. Et comme ces services utiliseront les standards industriels, qui sont préconisés par les grands operateurs, il est plus pertinent de s'y conformer que d'utiliser des formats de description et de composition spécifiques.

- Pour avoir une meilleure qualité de service, il est plus intéressant de disposer d'une certaine flexibilité au niveau des services modélisés dans la base de connaissances. Cette flexibilité concerne le remplacement d'un service par un autre qui est plus fiable et plus stable.

- L'utilisation d'un système à base de connaissances présente l'avantage de modéliser l'expertise liée à la composition sous forme de bribes de connaissances déclaratives. Ces dernières pourront donc être modifiées facilement. Le pouvoir déductif de ce type de système constitue une

caractéristique importante pour le cas de la composition de services car il permet d'obtenir des plans de composition variés.

1.7 Présentation du mémoire

Dans le chapitre 2, nous présenterons la littérature liée à la composition des services, nous analysons quelques systèmes et leurs caractéristiques.

Dans le chapitre 3 nous aborderons les systèmes à base de connaissances et les concepts de représentation des connaissances. Nous parlerons particulièrement des systèmes à base de règles et de leurs différentes composantes.

Dans le chapitre 4 nous présenterons notre outil de composition de services web. Nous décrirons les standards retenus pour la description des services. Nous allons également donner des exemples de modélisation de la composition de services afin d'illustrer les fondements de notre approches (approche mixte).

Dans le chapitre 5, nous décrirons une étude de cas où la composition de services est pertinente. Il s'agit d'une application de rétablissement d'un réseau de distribution d'énergie électrique.

En conclusion nous parlerons des résultats atteints dans le cadre de ce travail et des extensions qui peuvent être apportées à l'environnement de composition.

CHAPITRE II

LA COMPOSITION DE SERVICES WEB

2.3 Introduction

La composition de services est un sujet de plusieurs travaux de recherche aussi bien au niveau industriel que dans la communauté universitaire. Cette problématique est devenue intéressante car il relève le défit de réduire le coût et le temps de développement et de trouver de nouvelles façons d'intégrer les applications à travers le web. Il faut souligner que la composition de services est le résultat d'une évolution naturelle de l'architecture des applications et des technologies sous-jacentes. Plusieurs aspects architecturaux dans le cadre d'applications distribuées ont été améliorés pour une utilisation limitée à un réseau local (LAN). Par exemple la distribution des composantes sur des serveurs d'un réseau local permet d'assurer une interopérabilité des applications destinées pour une plate-forme donnée.

Dans la composition de services, on veut aussi disposer de cette flexibilité mais à travers le web. L'apparition des standards industriels pour les services web a permis d'apporter une nouvelle architecture basée sur les services. On parle de SOA (Services Oriented Architecture). Dans ce cas, il est devenu possible d'intégrer plusieurs composantes (services web) distribuées à travers le web. Cette interopérabilité utilise des standards XML. XML a la propriété de décrire la structure de son contenu et donc de permettre une interprétation automatique des données échangées entre les applications. Ce qui ouvre un horizon plein de possibilités dans le domaine du développement. La composition prend tout son sens dans ce contexte puisque on peut construire de nouveaux services à valeur ajoutée à partir de services de base existant sur l'environnement distribué « Internet ». Les différents niveaux d'abstraction offerts par les protocoles des services web, permettent de séparer le

mode fonctionnel du mode opérationnel d'un service web. Par contre, le processus de composition lui même exige la définition d'autres niveaux d'abstraction pour modéliser le flux de composition. Cette modélisation doit se faire par des outils permettant la génération automatique des plans de composition pour répondre aux exigences de modifications des processus d'affaires.

Dans ce chapitre, nous présenterons les deux approches de composition de services existant dans la littérature et nous montrerons les avantages et les inconvénients de chacune d'elles. Nous allons aussi définir un certains nombre de concepts liés à la composition de services pour comprendre le processus de génération des flux de composition.

Dans la section prochaine, nous parlerons du modèle d'implémentation de services web.

2.4 Modèle d'implémentation de services web

Un service web consiste à exposer un certain nombre de fonctionnalités utiles à travers un protocole web simple. Ce protocole est SOAP (Simple Object Access Protocol) et est basé sur XML. L'accès à ces fonctions exposées à travers le web se fait par des programmes clients écrits dans n'importe quel langage de programmation et tournant sur des plateformes variées (Windows, Linux...). Mieux encore, la génération de ces clients peut se faire automatiquement par des outils en se basant sur le descripteur du service. La description des services se fait par un deuxième protocole de services web qui est WSDL (Web Services Description Language).WSDL représente l'équivalent d'IDL pour les COM/DCOM. Il est également basé sur XML ce qui le rend manipulable par la machine.

Un service web peut être publié dans des registres afin qu'il soit retrouvé par des utilisateurs potentiels. La publication de services se fait par UDDI (Universal Discovery Description and Integration).

Dans la figure 2.1, nous présentons le modèle d'implémentation des services web.

- Chaque service Web est défini par un fournisseur. Le fournisseur de services déploie et publie la description de son service dans des registres en vue d'être localisé par des clients ;

- Les clients localisent leurs besoins en terme de services en effectuant des recherches sur les registres de services Web ;

- Une fois le service localisé, le client extrait sa description du registre ;

- Sur la base des informations définies dans la description du service, le client entreprend une interaction.

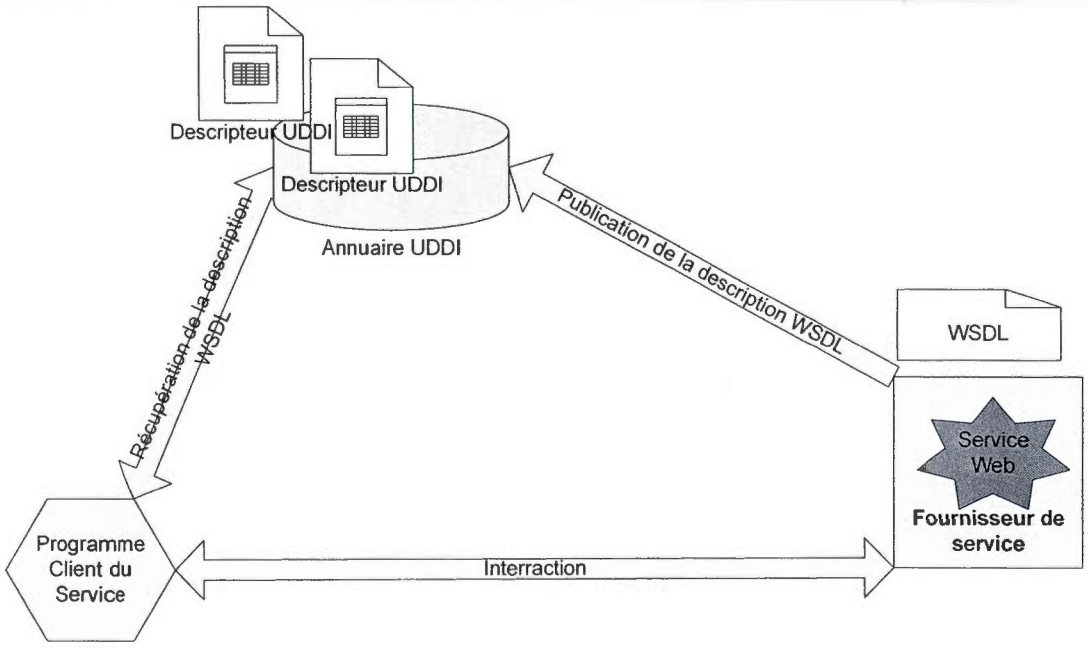

Figure 2.1 Le modèle d'implémentation de services web

2.5 Définitions

La composition de services est liée au cycle de vie des processus d'affaires. Les processus subissent des changements perpétuels pour répondre aux exigences d'un environnement dynamique. Les processus ne s'exécutent plus d'une façon prédictible et répétitive [14]. Un processus peut se définir de la façon suivante :

2.5.1 Processus

Un processus est un ensemble d'activités exécutées dans un certain ordre (workflow). L'exécution de ces activités implique des ressources humaines ou des systèmes, ou encore des applications [14]. Le cycle de vie d'un processus peut être couvert entièrement par des systèmes automatisés. Voir la figure 2.2

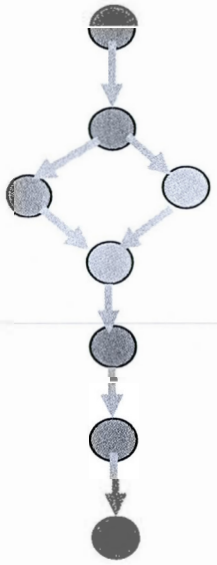

Figure 2.2 Structure d'un processus

2.5.2 Orchestration

C'est un processus exécutable pouvant interagir en même temps avec les services web internes ou externes et décrivant les règles d'affaires et l'ordre dans lequel les services sont exécutés [14]. Le processus est contrôlé par une des parties (services) intervenantes.

2.5.3 Chorégraphie

Elle décrit le rôle joué par chaque partie intervenant dans le processus et garde la trace des séquences de messages échangés entre les partenaires, clients, fournisseurs. La chorégraphie possède un caractère collaboratif en ce sens que chaque partie intervenant dans le processus décrit le rôle qu'elle joue dans l'interaction [14].

2.5.4 La composition de services

La composition de services regroupe les deux aspects « orchestration » et « chorégraphie » et s'intéresse au support des processus d'affaires en leur fournissant des standards et des outils permettant de générer les flux de composition.

On distingue deux types de composition :

- La composition statique consiste à générer manuellement les flux de composition. Il se fait soit par un langage classique de programmation ou par un standard comme BPEL4WS [16].

- La composition dynamique, en revanche, se base sur des outils automatisant la génération de flux. Ces outils disposent de certaines informations liées à la description sémantique de services. Les concepts sémantiques sont contenus dans des ontologies (web sémantique) qui sont des structures standardisées permettant leur interprétation [17].

2.6 Un exemple d'illustration de la composition

Nous reprenons l'exemple décrit dans [1] et [18]. Il s'agit d'une agence de voyage utilisant les services d'achat de billets d'avions et de réservation hôtelière. Le premier service appartient à deux compagnies aériennes et le deuxième à une chaîne hôtelière. Une troisième entreprise d'agence de voyages veut mettre en place un service Web d'organisation de formules de voyages selon une liste de critères ou de préférences (vol avec réservation d'hôtel). Au lieu de mettre en place un nouveau service, elle veut combiner et utiliser un mécanisme de gestion de préférences utilisant les services des deux compagnies précédentes. On parle d'agrégation de services.

La figure 2.2 montre les services utilisés par l'agence de voyage et les intervenants dans le processus de réservation

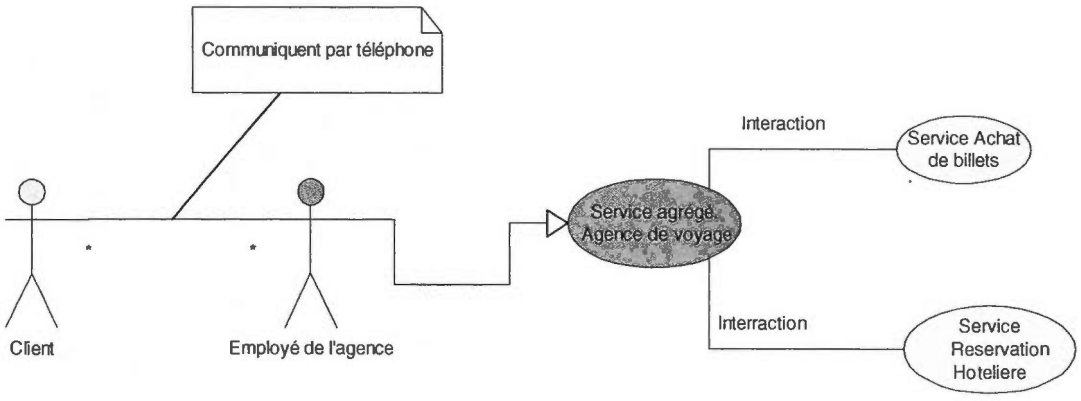

Figure 2.3 Agrégation de services

Dans cet exemple, le service d'agence de voyage utilise deux autres services dans un certain contexte relatif à la demande du client. Le comportement des services de base est dicté par la requête émanant du service agrégé.

Dans les prochaines sections (2.7 et 2.8) nous parlerons des deux approches de composition existantes dans la section 2.9, nous ferons une comparaison entre les deux en montrant les avantages et les inconvénients de chacune d'elle.

2.7 Approche industrielle

2.7.1 Description de services par WSDL

Dans cette approche, la description de services repose essentiellement sur l'aspect syntaxique [1] des messages en définissant les types de données échangés et les opérations que le service web fournit pour l'accès à ces données. La localisation des services est aussi incluse dans le document WSDL d'un service web. L'ensemble des opérations sont regroupées dans des « porttypes » qui représentent l'équivalent d'une classe dans un langage de programmation. La définition des messages consiste à spécifier ses différentes parties de données (types) que les opérations reçoivent lors d'une requête ou envoient lors d'une réponse. [6]. Les messages peuvent être comparés aux paramètres d'une fonction dans un langage classique. WSDL indique dans la partie « binding » le protocole de communication utilisé entre le client et le serveur. Dans la figure 2.4 nous indiquons les différentes parties d'un document WSDL.

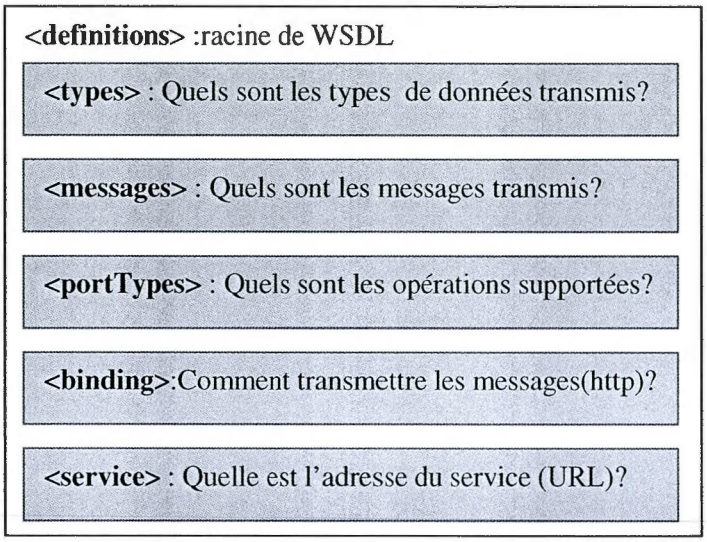

Figure 2.4 Structure d'un document WSDL

2.7.2 Le flux de composition par BPEL4WS

Le flux de composition permet de spécifier le comportement de l'ensemble des services appelés dans un contexte bien défini. Les messages échangés entre les services impliqués sont également indiqués. L'ordre d'invocation des opérations a son importance particulièrement pour les services composés

BPEL4WS permet justement de disposer d'un ensemble de directives et d'opérateurs permettant d'exprimer et de représenter les contraintes liées à la composition de services.

Un processus BPEL4WS utilise WSDL pour la description des messages et des données. La description des services est basée sur WSDL. Il est donc évident que BPEL est une abstraction décrivant le comportement d'un service composé (composite). La figure 2.4 montre la dépendance de BPEL des autres standards de services web.

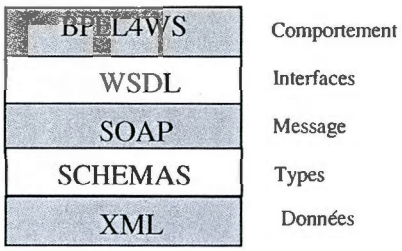

Figure 2.5 BPEL et les autres standards

Il est important de souligner que BPEL prend en charge le traitement des exceptions, i.e., permet de faire des recouvrements d'erreurs survenant lors de l'exécution. L'exécution en parallèle de services permet aussi de disposer d'un moyen d'améliorer les performances de services composés. BPEL sépare les règles de gestion des processus des services utilisés et par conséquent de remplacer facilement un service par un autre.

Nous reviendrons en détail sur les différents constructeurs de BPEL, dans le chapitre 4, car dans notre travail, BPEL a été choisi comme langage de composition.

Dans cette approche, la génération de flux de composition est faite manuellement. Il y a donc un besoin d'outils pour la composition automatique.

2.8 Approche web sémantique

2.8.1 Description de services

En web sémantique, les pages du web sont considérées liées par un ensemble d'informations décrivant la sémantique de ces liaisons dans une sorte de base de données globale [1]. En général cette description est un ensemble d'assertions sur des ressources web et de leurs propriétés. Le langage de description est RDF (Resource Description Format) [30]. Une description « RDF » est un ensemble de triplets ayant chacun un sujet, un verbe et un objet d'une phrase. Chaque élément de la phrase est représenté par une URI (Universal Resource Identifier). RDFS (Resource Description Format Schema) est utilisé pour définir les schémas RDF qui permettent de modéliser les classes et les propriétés des ressources. DAML-S (The DARPA Agent Markup Language Schema) est un autre standard utilisé pour

compléter la description de services. Il utilise les ontologies pour exprimer plusieurs types de relation ou de propriétés. DAML-S définit une classe pour la description de service appelée « Service » ayant les propriétés « presents, describedBy et supports ». Chaque propriété utilise les classes « ServiceProfile, ServiceModel et ServiceGrounding ».

Avec les standards du web sémantique, il est possible d'avoir une description assez complète des services web et des processus utilisant ces services.

2.8.2 Description de la composition

La description de services avec RDF/DAML-S est transposée sous-forme d'inférences logiques exprimées en Prolog par exemple. Golog [19] est un autre langage conçu spécialement pour la composition et il est une extension de Prolog. Il supporte la spécification et l'exécution d'actions complexes dans les systèmes dynamiques. Il permet de décrire la composition sous forme d'un ensemble d'objectifs à atteindre et de spécifier l'ordre dans lequel ces objectifs doivent être vérifiés.

On remarque donc que dans ce type de systèmes, on utilise les techniques empruntées aux systèmes à base de connaissances (Ce qui fait l'objet de chapitre 3 de ce travail). A chaque étape du processus de génération de plans de composition, il faut un choix parmi plusieurs possibilités (choix non-déterministe). Les étapes subséquentes dépendent des choix effectués lors des étapes précédentes et ainsi on arrive à construire un plan de composition de manière progressive.

Nous allons explorer en profondeur cette technique au chapitre 4 et nous donnerons des illustrations par des exemples mais en utilisant un autre langage (Le langage de JESS)

2.9 Comparaison des deux approches

La technologie des services web a offert un cadre de travail pour le développement très intéressant en termes de standards (SOAP, WSDL, UDDI). Ces standards ont eu des extensions incontestablement très riches pour justement supporter la composition de services complexes. Dans les sections précédentes, nous avons mentionné quelques uns : BPEL4WS, RDF/RDFS, DAML-S. Il en existe d'autres comme OWL_S, BPML WSCI.

Nous pensons que la définition de standards est une étape nécessaire mais non suffisante pour l'utilisation pratique de la composition de services dans le développement d'applications ou du logiciel en général.

Dans les deux approches développées au cours des dernières années, l'accent a été mis sur ces standards. Dans l'industrie, le problème de l'automatisation de la composition n'est pas ou peu abordé. L'approche web sémantique est beaucoup plus évoluée en ce sens qu'elle a pris en compte la sémantique relative à la construction de services web complexes. Elle a plus ou moins élaboré des standards plus robustes en matière de description de services. Certains outils de la composition ont vu le jour. Ces outils permettent de faire une pseudo-automatisation de la composition.

Nous constatons donc que malgré les tentatives d'élaboration de standards pour supporter le développement de services à valeur ajoutée, il reste beaucoup de travail à faire au niveau de la composition pour prendre en charge l'aspect « comportement » de service.

Dans les deux approches, les informations sur le processus peuvent s'attacher à des protocoles différents. Dans l'approche industrielle, le choix des actions à exécuter se fait de façon prédéterminée avec les constructeurs « pick et switch ». Les activités sont exécutées suite à des évènements qui surgissent. La nature des informations de description et de composition sont strictement syntaxiques

En revanche dans l'approche web sémantique, les choix des actions ne sont pas prédéterminés. Les transitions entre les états sont basées sur des pré-conditions. La nécessité de représenter des objectifs à atteindre lors de la composition de manière explicite permet d'avoir une flexibilité accrue. La nature des informations est sémantique.

2.10 Conclusion

Dans ce chapitre nous avons analysé les techniques utilisées pour la composition de services dans les deux approches et les caractéristiques de chacune d'elles. Nous avons vu qu'au niveau industriel, l'accent est mis sur la définition de standards et non sur l'automatisation de la composition. Cependant dans l'approche web sémantique, on s'intéresse à l'automatisation de la composition (complète ou partielle) et que les

informations de description de services ne sont pas complètement séparées de celles de la composition.

Beaucoup de travail reste à faire dans ce domaine; notamment au niveau du raffinement des standards et la construction d'environnement de composition pour automatiser la génération de flux de composition.

Par ailleurs, les travaux déjà effectués commencent à donner des résultats pour une semi-automatisation de la composition.

Dans le chapitre 4, nous présenterons notre approche appelée « approche mixte » et nous définirons les modèles de la composition que nous avons adoptés.

CHAPITRE III

JESS ET LES SYSTEMES À BASE DE CONNAISSANCES

3.1 Introduction

Dans ce chapitre, nous décrivons les systèmes à base de connaissances de façon générale et le système JESS (Java Expert System Shell) en particulier de façon plus détaillée. L'utilisation de JESS, dans notre approche pour la composition de services, constitue un des éléments clés de notre solution. La compréhension des concepts de base des « systèmes à base de connaissances » (SBC) a fait donc partie de la réflexion menée lors de l'élaboration de l'approche utilisée dans ce mémoire. Nous avons exploré l'approche des systèmes à base de connaissances de façon approfondie afin de comprendre ses principaux fondements et d'en tirer profit pour leurs application à la composition de services. L'idée d'utiliser un SBC pour la composition de services nous est venue en identifiant les caractéristiques suivantes de notre problématique :

- Les connaissances liées à la composition de services sont pragmatiques et sujettes constamment à des changements.
- Ces connaissances doivent être facilement modifiables et indépendantes les unes des autres
- Les mécanismes traitant ces connaissances sont aussi indépendants de ces dernières

Notre étude des systèmes à base de connaissances, nous a conduits à choisir JESS comme outil de représentation et de traitement de la composition de services. Le choix de

JESS facilite l'intégration des différents modules de notre système de composition. Cela s'explique du fait que nous disposons du code source (Java) de JESS qui nous a été fourni par « Sandia National Laboratories ». D'autre part, beaucoup d'outils traitant du XML sont également disponibles en Java.

Dans les prochaines sections, nous présentons dans un premier temps les principes de base d'un système à base de connaissances et nous abordons les mécanismes de représentation de connaissances offerts par JESS ainsi que ses modes de raisonnement.

3.2 Principes des systèmes à base de connaissances

On désigne généralement par le terme de SBC tout système qui permet de résoudre des problèmes dans un domaine délimité (diagnostic médical, VLSI) pour lequel il existe des experts, ayant une grande connaissance du problème, pouvant le résoudre efficacement. Les SBC tentent de simuler le raisonnement de ces experts humains dans des domaines d'application où des algorithmes ne sont pas toujours disponibles.

Il existe de nombreux domaines (médecine, géologie, chimie) où les connaissances ne sont pas suffisamment structurées pour qu'on puisse trouver des algorithmes qui permettent de résoudre un problème en utilisant une méthode connue et appropriée. Dans un SBC, on progresse essentiellement par essais/erreurs; à chaque étape du raisonnement plusieurs choix sont possibles et il est impossible de faire une examination exhaustive. Les SBC adoptent des méthodes de résolution de problèmes utilisées par les experts; ces méthodes proviennent de longues années d'expérience et sont basées sur des heuristiques. Ce sont ces techniques qui permettent d'orienter la recherche du système et d'éviter l'explosion combinatoire lorsque le système doit faire face à de trop nombreux choix.

En permettant d'automatiser des processus plus complexes que de simples routines de calcul, les SBC ont introduit l'informatique dans des domaines où son utilisation était réduite, voire impossible, ou encore ont permis de résoudre de façon plus simple des problèmes déjà traités.

La résolution d'un problème par des moyens algorithmiques peut se traduire par le schéma suivant : Données + Algorithme = Programme

Alors que l'approche « SBC » se résume par :

Connaissances + inférence = SBC

3.2.1 Caractéristiques des systèmes à base de connaissances

En plus de la dichotomie entre connaissances et moteur d'inférences, il existe un certain nombre de principes fondamentaux caractérisant les SBC:

- La base de connaissances est indépendante du moteur d'inférences qui utilise le contenu de cette base. C'est pourquoi on les appelle « systèmes basés sur la connaissance »;
- La connaissance est donnée sous forme déclarative;
- Les SBC doivent être capables de donner des explications concernant les raisonnements qu'ils ont effectués;
- Les connaissances manipulées sont essentiellement de nature symbolique, par opposition aux données numériques utilisées par des programmes classiques;
- Les SBC sont spécialisés dans un domaine d'application et non plus dans une tâche, comme les programmes classiques.

3.2.2 Anatomie d'un système à base de connaissances

Un SBC est principalement composé d'une base de connaissances (BC), représentant l'expertise collectée au sujet du domaine traité (par exemple la composition de services), d'une base de faits (BF), ensemble de faits décrivant le problème précis que l'on propose au système de traiter et d'un moteur d'inférences (MI) chargé d'appliquer les connaissances générales de la base de connaissances au cas particulier décrit dans la BF. Les interfaces sont indispensables à une bonne communication homme-machine, l'une facilite le dialogue avec l'utilisateur au cours d'une session et l'autre permet à l'expert du domaine de consulter ou d'enrichir la BC du système. La figure 3.1 montre la structure d'un SBC.

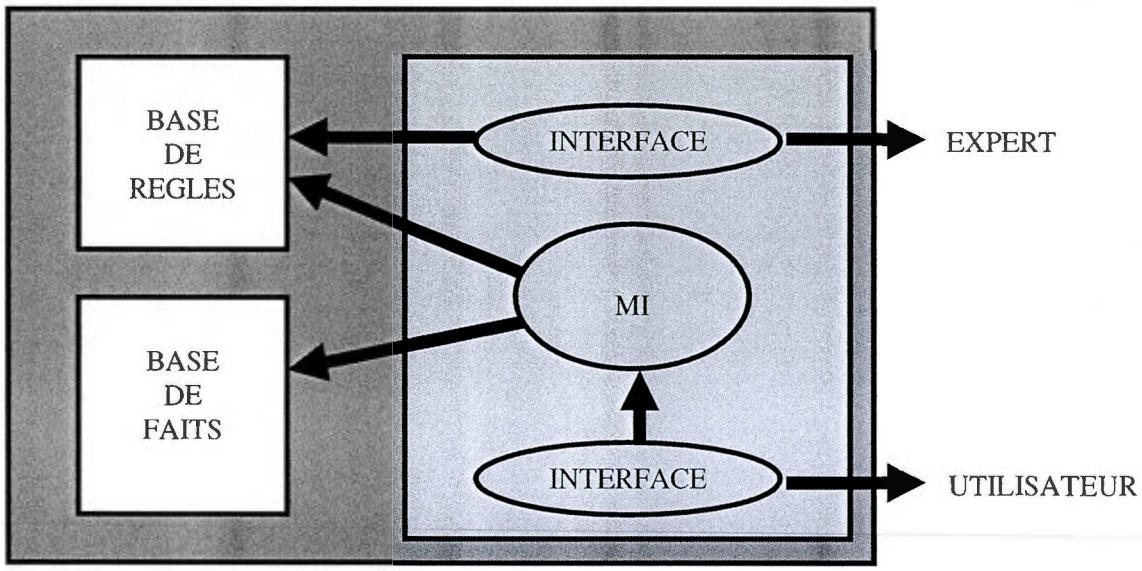

Figure 3.1 structure d'un système expert

3.2.3 Élaboration d'un système à base de connaissances

La conception et la réalisation d'un SBC peut suivre les étapes du schéma de la figure 3.2.

3.2.3.1 Acquisition

L'environnement d'acquisition des connaissances d'un système problème doit offrir un ensemble d'éditeurs (édition d'objets, éditeur de tableaux, éditeur de règles…), des fonctions intégrées et des fonctions graphiques. Les possibilités offertes par le module d'acquisition des connaissances doivent permettre d'utiliser les données sous leur forme initiale afin d'éviter les erreurs inhérentes à toute transcodification et de redondance.

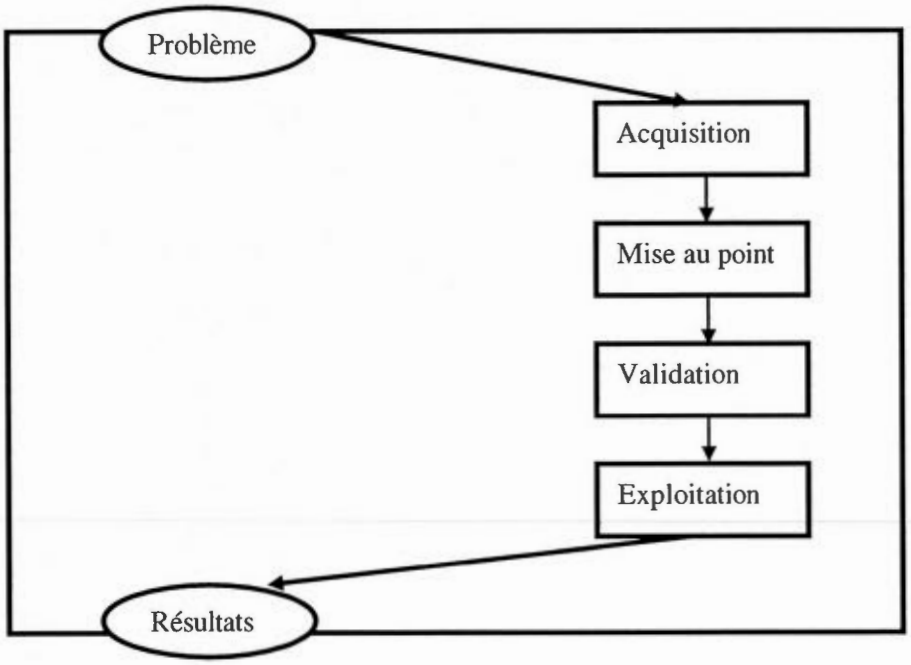

Figure 3.2 les étapes de développement d'un SBC

3.2.3.2 Mise au point

Cette phase met à la disposition du cogniticien un ensemble de fonctions lui permettant d'effectuer un contrôle syntaxique des règles et d'effectuer la recherche d'un élément (fait, règle, objet) pour faire des corrections éventuelles. Par exemple, lister toutes les règles ou apparaît un fait.

3.2.3.3 Validation

Les générateurs de systèmes experts sont appelés à gérer des bases de connaissances de plus en plus importantes tant sur le plan quantitatif que qualitatif, par conséquent ils sont censés offrir des outils assurant le regroupement des règles en plusieurs contextes spécialisés.

Exemple de contexte :

Soit une base de règles spécialisée dans le dépannage d'une voiture, on distingue des sous ensembles traitant des problèmes électriques (contexte électrique) et d'autres sous ensembles pour des problèmes mécaniques (contexte mécanique).

C'est dans cette phase de validation que le contrôle de la cohérence d'une base de connaissances s'effectue.

3.2.3.4 Exploitation

L'exploitation consiste à mettre en œuvre via le moteur d'inférences une base de connaissances validée; la communication avec l'utilisateur est assurée par l'interface utilisateur. Le moteur d'inférences doit offrir plusieurs stratégies de contrôle :

- Chaînage avant
- Chaînage arrière
- Chaînage mixte

L'interface utilisateur doit permettre le dialogue en : mode textuel classique, mode graphique statique ou dynamique.

3.3 Présentation de JESS

JESS (Java Expert System Shell) est un Shell de systèmes experts écrit entièrement en Java. Son utilisation a pris de l'ampleur depuis quelques années. Il peut être exploité de deux façons différentes :

- Pour développer des systèmes à base de connaissances en formulant l'expertise d'un domaine donné sous forme de règles exprimées par des prédicats;
- Ou encore comme un langage de programmation pour le développement rapide d'applications. Son langage est donc interprété (contrairement à Java), et permet aussi d'accéder à toutes les librairies Java.

Dans le cadre de ce travail, nous avons exploité son aspect relatif aux systèmes à base de connaissances.

Il faut aussi mentionner que JESS est un clone de CLIPS (C Language Integrated Production System) mais plus rapide. Il est plus facile à intégrer dans des applications Java exploitant la technologie des SBC. Plusieurs application industrielles utilisent JESS sans même que cela soit indiqué explicitement.

Le moteur de JESS utilise l'algorithme RETE (du latin qui signifie réseau) qui est à la base de l'amélioration des performances dans le processus de déclenchement de règles dans les systèmes d'ordre 1. Cet algorithme intervient dans l'étape de filtrage des règles (choix de la règle à utiliser parmi les règles candidates). JESS fut l'un des premiers systèmes d'ordre 1(utilisant la logique des prédicats) à utiliser cet algorithme. Pour cela il est considéré comme le moteur le plus performant

Dans les prochaines sections, nous parlerons des « artefacts » de JESS [5] utilisés dans ce travail. Ensuite, nous présentons les formalismes utilisés pour représenter les connaissances.

3.3.1 L'algorithme RETE

Le but de cette section est de comprendre le fonctionnement de RETE (algorithme de filtrage de JESS) afin de pouvoir écrire les règles de manière à ce qu'elles soient exécutées efficacement par JESS. Aussi, la structure de données utilisée dans RETE nous permet de générer les plans de composition de services web.

RETE est un algorithme incrémental d'unification qui permet de réduire le temps d'exécution des règles en évitant l'itération.

Une implémentation sans RETE consiste à tester toutes les conditions de toutes les règles et ajouter les conclusions des règles déclenchées à la mémoire de travail. Cette façon de procéder est inefficace car le résultat des tests effectués sur les conditions des règles sont toujours les mêmes et ne changent pas par rapport à l'itération précédente. La complexité

engendrée par ce type d'implémentation est O(RF^P) où R est le nombre de règles, F le nombre de faits dans la mémoire de travail et P le nombre moyen de « patterns » par règle.

Ceci nous permet de constater que la mémoire de travail est donc relativement stable et que la plupart du temps est passé à refaire les mêmes tests pour vérifier si une règle est déclenchable.

C'est pourquoi plusieurs systèmes d'ordre 1 (JESS, OPS5 et CLIPS) utilisent RETE pour sauvegarder le résultat des tests précédents pour ne pas les refaire à chaque cycle. Et ainsi la complexité est réduite à O(RFP).

Dans RETE, on construit un réseau dont chaque nœud représente un ou plusieurs tests effectués sur les conditions des règles. L'ajout d'un ou plusieurs faits est propagé à travers ce réseau de nœuds. Les feuilles du réseau représentent les conditions instanciées des règles. On les appelle les « activations »

Dans la figure 3.3, on représente un exemple de réseau RETE pour 2 règles :

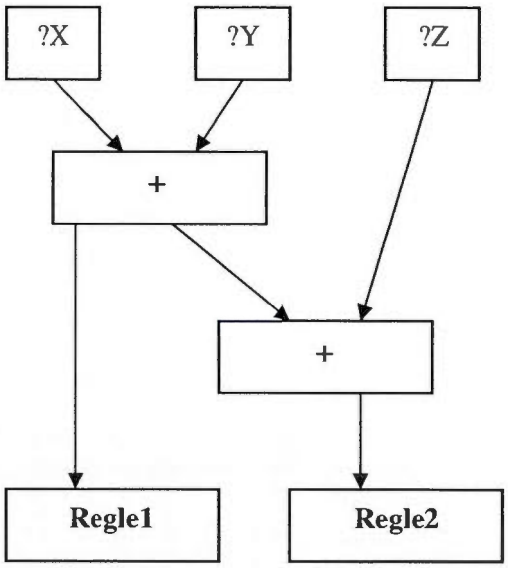

Figure 3.3 Un exemple de réseau RETE

3.3.2 La mémoire de travail (ou la base de faits)

JESS maintient une collection de bribes de connaissances appelés faits. Cette collection est la mémoire de travail. Toutes les règles de la base de connaissances réagissent aux changements apportés à la mémoire de travail. Ces changements définissent le fonctionnement de JESS. On peut ajouter, modifier ou effacer des faits de la mémoire de travail. Chaque fait est représenté par un objet Java. Ce qui nous permettra de manipuler les faits dans une application. Cette possibilité est exploitée dans le prototype que nous avons implémenté dans le cadre de notre travail. Nous y reviendrons en détails dans le chapitre 4.

Ils existent deux types de faits : ordonnées et non ordonnés.

3.3.2.1 Les faits non ordonnés

Ils sont l'équivalent des objets dans les langages orientés objets. Les données sont stockées dans des slots (genre d'attributs). Voici un exemple de fait non ordonné en JESS qui définit une voiture de marque « ford », du modèle « focus » et sortie l'année 2001:

```
(voiture (marque ford) (modele focus) annee(2001))
```

Avant de pouvoir manipuler les faits non ordonnées, il faut d'abord définir le gabarit (genre de classe en java) avec la directive deftemplate. Le gabarit ou le moule pour définir un fait non ordonné pour une voiture est le suivant :

```
(deftemplate voiture
 (slot marque)
 (slot modele)
 (slot annee (type INTEGER)))
```

3.3.2.2 Les faits ordonnés

Un slot en JESS est l'équivalent d'un attribut d'une classe en java ou en C++. Un fait non ordonné peut avoir plusieurs slots. Un slot contient une seule valeur. Si le slot contient plusieurs valeurs, il est appelé un multi-slot.

Dans le cas des faits ordonnés, il n'est pas nécessaire de définir des gabarits.

Ces derniers sont générés automatiquement par JESS. Ces gabarits possèdent un seul multi-slot. Le nom de ce slot est « __data » Les faits ordonnés sont des listes dont le premier élément peut être considéré comme une relation. Par exemple (Père a b) signifie que a est le père de b.

3.3.2.3 La manipulation de faits

Il existe plusieurs façons de manipuler des faits :

- L'ajout d'un fait dans la mémoire de travail se fait par l'opérateur « assert ». par exemple (assert (Père a b))
- La suppression d'un fait se fait par l'opérateur « retract ». ex : (retract (Père a b)
- Modification d'un fait se fait par l'opérateur « modify »

Lorsqu'on veut effectuer une série d'insertion de faits dans la mémoire de travail, il est préférable d'utiliser le constructeur « deffacts ». En général cela se fait à l'initialisation d'une session. La commande « reset » permet d'insérer tous les faits indiqués par « deffacts » dans la mémoire de travail.

3.3.3 Les règles en JESS

Les règles sont des connaissances ayant deux parties : les conditions et les conclusions. Une règle n'est exécutée que si sa partie « condition » est vérifiée. L'ordre dans lequel les règles sont exécutées ne peut être déterminé à l'avance. C'est ce qui caractérise ce genre de

système. JESS permet donc d'avoir un comportement non déterministe contrairement à un programme classique.

L'exécution d'une règle par JESS se fait donc en essayant de trouver des faits dans la mémoire de travail qui satisfont sa partie condition.

Dans l'exemple suivant, on définit une règle permettant de déduire que X est grand père de Z s'il existe une personne Y telle que X est père de Y et qu'en même temps Y est père de Z. Notez l'utilisation de la directive « assert » qui permet d'ajouter un nouveau fait à la base de faits.

```
(defrule parenté
(PERE ?X ?Y)
(PERE ?Y ?Z)
=>
(assert (GRAND_PERE ?X ?Z))
)
```

Il faut noter aussi que la partie conclusion utilise presque toujours un appel de fonction permettant de modifier la base de faits. L'opérateur de relation existant entre les prédicats de la condition est implicitement un ET (AND). Il est également possible d'utiliser une combinaison d'opérateurs AND, OR et NOT.

Il existe plusieurs autres opérateurs dans JESS; pour plus d'informations veuillez consulter la documentation détaillée de JESS.

3.4 Conclusion

Dans ce chapitre, nous avons présenté les principes des systèmes à base de connaissances tout en mettant l'accent sur les aspects de JESS qui sont pertinents pour la composition de services. Notre solution est basée sur l'exploitation des fonctionnalités de

JESS. Le choix de ce dernier comme outil est stratégique vu la nécessité de disposer d'une certaine flexibilité au niveau de l'intégration de la technologie des systèmes à base de connaissances avec les standards industriels relatifs aux services web. Cette intégration nous a permis de rassembler la puissance de WSDL (pour la description de services) et le pouvoir déductif de JESS dans la génération des plans de composition de services web. Nous avons étudié également les mécanismes offerts par JESS pour représenter les éléments permettant de décrire les services web à savoir les règles et les faits.

Dans le chapitre 4, nous montrerons comment nous allons décrire les services web par les mécanismes de JESS abordés dans ce chapitre.

CHAPITRE IV

UN ENVIRONNEMENT DE COMPOSITION DE SERVICES

4.1 Introduction

L'analyse de la littérature sur la composition de services, faite dans le chapitre 2, a montré qu'il existe deux tendances qui ne sont pas convergentes. La première met l'accent sur la définition des standards pour la description de services et de processus. C'est l'approche industrielle. Les standards continuent d'évoluer pour répondre aux contraintes reliées à la mise en œuvre des services web et de la composition. La deuxième tendance utilise aussi des standards, certes différents, mais plus riches en termes d'informations sémantiques sur la description de services. Il s'agit de l'approche web sémantique.

Les deux approches possèdent des standards mais n'abordent pas la problématique de la composition de services dans une même perspective. Car à notre sens, l'approche web sémantique définit des standards pour représenter les informations liées aux contextes dans lesquels les services web peuvent être combinés et assemblés dynamiquement pour construire des services à valeur ajoutée. Par contre l'approche industrielle élabore des standards pour donner un support permettant de construire des processus manuellement.

Nous pensons que la définition des standards est une étape nécessaire mais non suffisante pour répondre aux besoins de la composition dynamique de services.

Dans ce chapitre, nous allons présenter une approche « mixte » basée sur des standards industriels et utilisant les techniques de composition dynamique inspirées de celles appliquées dans l'approche web sémantique. Le fossé séparant les deux approches sera rempli d'un certain nombre de modèles que nous définirons pour représenter les services.

Notre approche se base essentiellement sur l'utilisation des règles modélisant les connaissances de la composition des services web. JESS, tel que décrit dans le chapitre 3, est utilisé comme outil d'exécution de règles pour la génération des flux de composition. Les règles de JESS représentent l'aspect dynamique des services web. Dans les règles, nous spécifions les conditions dans lesquelles les opérations de chaque service peuvent être invoquées.

4.2 Constats sur les approches actuelles de la composition

La composition de services nécessite la représentation des informations décrivant les contextes dans lesquels les services sont appelés à collaborer et à échanger des informations afin de répondre à un ou plusieurs besoins spécifiques. La nature évolutive de ces besoins et leur variété rend la tache de définition de services très difficile. Pire encore, la prise en compte des informations contextuelles pour combiner les services et prendre en charge de nouveaux besoins devient plus compliquée car les services n'étaient pas conçus, du moins à leurs créations, pour être utilisés ensemble.

La modélisation des informations contextuelles, relative à la composition de services, est abordée différemment dans les approches de composition de services qui existent dans la littérature et ne répondent pas nécessairement aux exigences et à la rigueur des applications du monde réel. A notre sens, les objectifs poursuivis par les méthodes de composition actuelles ne sont pas les mêmes.

D'une part, l'approche industrielle modélise les services web sous une forme exclusivement syntaxique et ne prend pas en compte la sémantique liée à l'utilisation des services. La définition de services uniquement par leurs interfaces WSDL indique seulement les opérations disponibles sur les différents ports. WSDL n'indique pas l'ordre dans lequel ces opérations devraient être appelées. Il faut aussi noter que les appels effectués par une application cliente à un même service pendant la même session sont indépendants. Aucune trace sur l'état du service entre deux appels n'est gardée. La difficulté est encore plus importante si on veut utiliser un service web dans un processus d'affaire pour lequel il n'a pas été conçu. Un processus d'affaire faisant appel à plusieurs services web doit faire collaborer les différents acteurs ou services de façon cohérente car chaque acteur peut

changer le flux d'exécution ou le comportement du processus par les informations reçues de ces acteurs. De l'ensemble des services mis en collaboration peut résulter un comportement de processus qui aurait pu être différent si chaque service avait été utilisé séparément. Si par exemple, le comportement anormal du processus n'est pas traité, on pourrait aboutir à une situation incohérente de l'état final des données.

Les mêmes problèmes restent posés pour l'approche web sémantique. Néanmoins, elle utilise des mécanismes de représentation de la sémantique liée à la composition beaucoup plus riches que la méthode industrielle. Les ontologies et les systèmes à base de connaissances permettent de prendre en charge certaines difficultés liées à la modélisation de la composition. La description de services agrégés sous forme de buts à atteindre simplifie énormément le processus de composition car la transformation de ces buts en faits vérifiables permet de savoir si un service agrégé est réalisable à partir d'un ensemble de services donné.

L'approche web sémantique présente certains inconvénients au niveau de la séparation des informations liées à la composition. Il n'établit pas de niveaux d'abstractions entre les informations de description propres aux services et les informations de composition de l'ensemble des services. Cette liaison entre les deux niveaux d'abstraction empêche l'établissement de standards. Ce qui conduit à une panoplie de pseudo standards et ne permettent pas de bâtir une infrastructure de déploiement de services provenant de plusieurs fournisseurs. Il est donc indispensable de définir des standards. Ce pari est réussi par la méthode industrielle.

L'état actuel de la composition des services web est caractérisé par l'absence d'outils de composition dynamiques, particulièrement pour la méthode industrielle, pouvant être utilisés pour des applications du monde réel. L'établissement de standards, au niveau industriel, est déjà une étape importante dans le domaine de la composition. Les outils existant, dans la sémantique web, ne sont pas appropriés car ils ne s'intègrent pas dans les environnements de développement.

La modélisation de la composition des services web est aussi un autre problème qu'il faut résoudre. L'intérêt de la modélisation de la composition est de pouvoir gérer les changements subis par les processus de façon rapide et efficace. Ces changements sont

fréquents et nécessitent des outils de conversion entre différents modèles afin de faciliter la collaboration entre les différents acteurs qui interviennent dans un processus. Il est important de représenter la composition par différents modèles car les acteurs ne considèrent pas le problème de la composition sous une même perspective. Différents concepts sont manipulés et donc la représentation de ces derniers en dépend. Il serait alors souhaitable de disposer d'outils permettant de passer automatiquement d'un modèle à l'autre selon la nature de l'acteur qui intervient dans la modélisation de la composition. La définition des processus, étant complexe, nécessite plusieurs étapes avant d'arriver à un résultat satisfaisant.

La complexité de la composition est liée à l'aspect multidimensionnel de la modélisation des services. Non seulement la représentation syntaxique et sémantique devront être prises en compte, mais il faut aussi disposer d'une métrique permettant de faire des choix répondant aux critères de qualités établis pour le développement d'applications réelles (Figure 4.1).

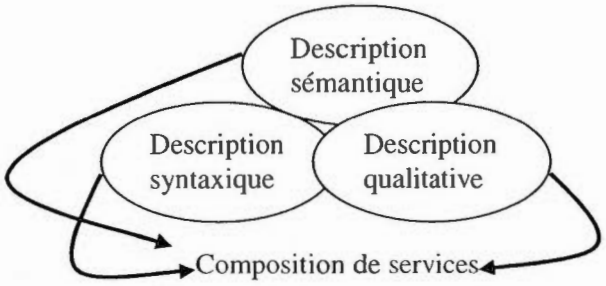

Figure 4.1 L'aspect multidimensionnel de la composition

4.3 Présentation de l'approche mixte

Notre vision de la composition a été abordée d'un point de vue pratique, c'est-à-dire que nous avons conçu une approche pragmatique supportée par un système facile à utiliser et pouvant s'intégrer facilement dans un environnement de développement.

L'idée est de considérer la composition comme étant une solution à un problème complexe pouvant être décomposé en plusieurs sous problèmes et ce processus de décomposition peut s'appliquer à chacun des sous problèmes jusqu'à ce qu'on arrive à un niveau de détail suffisamment simple pour trouver un service pouvant prendre en charge le

sous problème en question. Les services qui interviennent dans la composition ne sont pas nécessairement atomiques, i.e., donnant un simple résultat suite à un appel d'opération. Ils peuvent, donc, eux même être composés d'autres services.

La méthode de composition utilisée dans l'approche mixte se base sur les principes suivants :

- **Utilisation des standards industriels :** Dès le début, nous avons choisi de nous conformer à l'utilisation des standards industriels de déploiement de services web car il serait inutile de définir des formats propriétaires auxquels il faut développer des outils les supportant. Un tel développement exigerait beaucoup d'efforts et de temps. La description de services web dans notre approche se fait donc en WSDL. L'autre avantage est de pouvoir bénéficier des services déployés sur Internet qui seront de plus en plus nombreux et utiles. La réutilisation de composantes sous forme de services justifie amplement ce choix. La définition de la composition par BPEL4WS est faite pour les mêmes raisons. Des outils de déploiement de services agrégés sont déjà disponibles. Nous en avons choisi celui d'IBM appelé BPWS4J (Business Process Web Services For Java).

- **L'utilisation d'un système à base de connaissances :** Pour supporter l'aspect dynamique de la composition, il faut disposer d'un moyen permettant de compléter la description de services, par WSDL, qui est en mesure de représenter les services web sous une forme déclarative. Il s'agit alors d'utiliser un système à base de connaissances permettant de représenter les services web de base par des règles de production. Chacune de ces règles indique les pré-conditions et les post-conditions d'exécution ou de lancement de l'opération qu'elle représente. Le système à base de connaissances permet aussi de mettre au point les règles de description de services en les exécutant et en les modifiant de manière souple. Cette souplesse est nécessaire car les changements qu'on effectue dans la base de règles sont nombreux et fréquents. La mise au point concerne aussi la cohérence des services web mis en contribution pour la composition. Il est, par exemple, utile de vérifier si une règle représentant une opération dans le fichier WSDL rentre bien dans le contexte des autres opérations.

S'il s'avère qu'elle n'est jamais interpellée, sa présence dans la base de règles ne sera pas nécessaire. L'autre avantage d'utiliser un système à base de connaissances est de pouvoir disposer d'un deuxième modèle d'exécution qui est surtout exploité lors de la mise au point des règles. On n'est donc pas lié au modèle final qui est BPEL4WS pour le test d'une base de connaissances car le processus risque de devenir plus lent.

• **Liaison entre les règles et les fichiers WSDL :** Pour retrouver la description WSDL d'un service à partir d'une règle de la base, il est indispensable d'adopter une convention de nomination des règles. Cette convention utilise les noms des fichiers, des ports et des opérations des différentes descriptions WSDL de services. Les informations retrouvées concernent les types de données échangées et les paramètres de retour des opérations.

Le schéma de la figure 4.2 donne une vue globale de l'approche mixte.

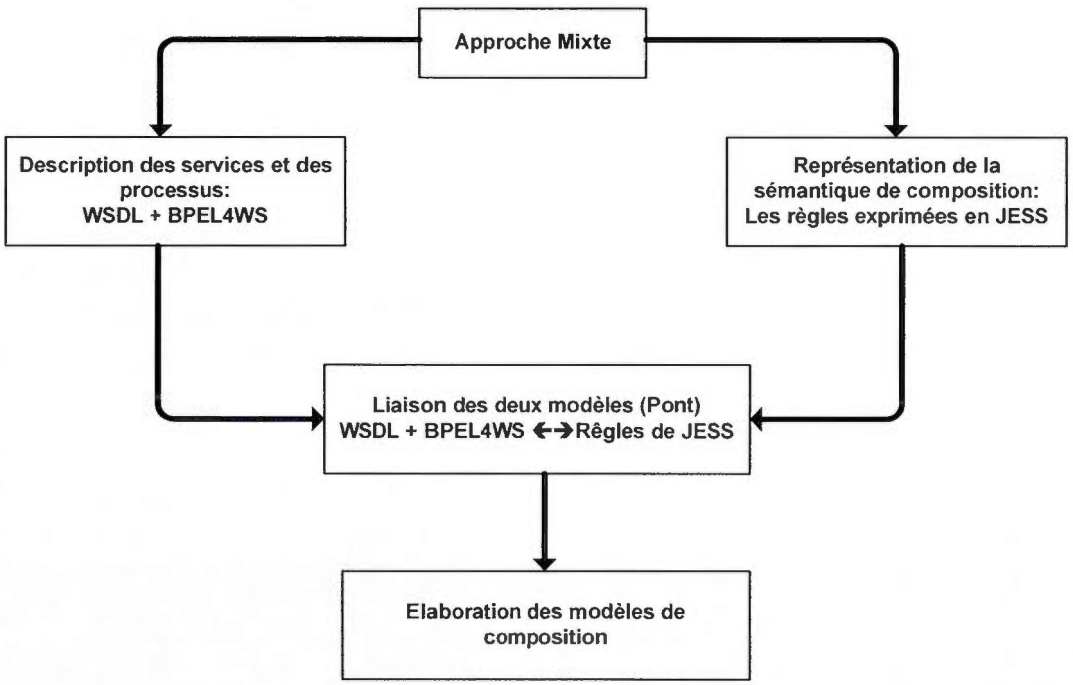

Figure 4.2 Approche mixte

4.4 Positionnement de l'approche mixte dans les standards

L'apport de notre approche réside essentiellement dans la modélisation des aspects dynamiques de la composition. La place dans laquelle nos modèles se situent par rapport à la pile des standards montre que nous complétons les niveaux de description de services et de processus. Les règles de productions sont exploitées comme un mode de représentation de la sémantique relative à la composition (Figure 4.3).

Les descriptions WSDL sont reliées aux règles de telle sorte à pouvoir accéder aux informations descriptives des services (WSDL) à partir des règles. Les règles déclenchées lors de la composition nous permettent de générer les plans de composition. Les plans de composition, à leur tour, vont être utilisés pour générer le code BPEL4WS représentant les services composites.

Il faut aussi remarquer qu'il est possible d'exploiter UDDI pour mettre en place un modèle d'invocation dynamique de services en utilisant les règles.

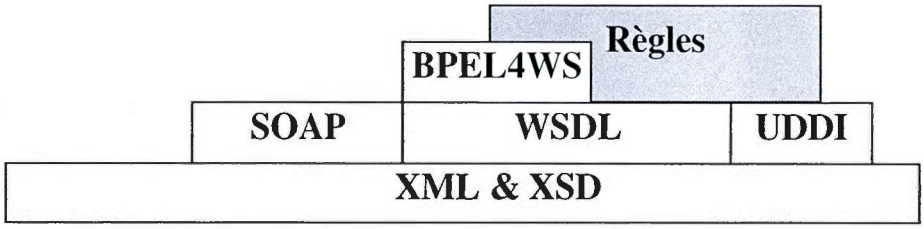

Figure 4.3 Modèle de l'aspect dynamique des services web

La modélisation de l'aspect dynamique de la composition par les règles se fait par le concepteur de la base des connaissances. Son rôle consiste à analyser le domaine d'affaires et de choisir les services qui s'y rapportent. L'implémentation du service composé se fait en BPEL et l'exécution des règles est réalisée par JESS (Figure 4.4). Il s'agit alors de faire l'intégration des règles et du langage d'exécution des processus. Le choix des services de base à utiliser se fait lors de l'analyse. Ce qui donne plus de flexibilité au niveau de

l'évolution du service composé qui utilise un ou plusieurs services de base. Le changement d'un service de base n'affecte pas le service composé pourvu qu'il soit remplacé par un autre.

La composition suit un cycle composé d'un ensemble d'étapes. Les frontières entre les étapes ne sont pas nécessairement délimitées car le cycle est itératif (Figure 4.5). On procède par raffinement successifs jusqu'à ce qu'on arrive à un service composé répondant aux critères de qualités spécifiés. Chaque étape du cycle peut être supportée par des outils d'aide à la composition. L'idéal serait de disposer d'un environnement qui permet d'automatiser complètement la composition.

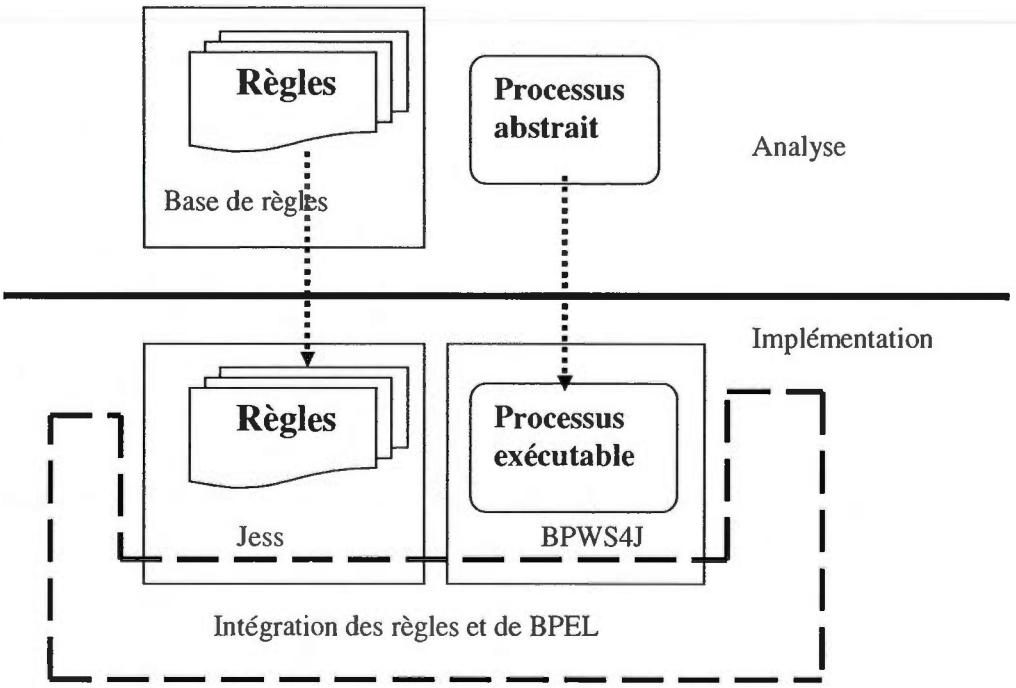

Figure 4.4 Approche de composition

La découverte de services se fait en choisissant les services qui sont sémantiquement composables. La planification consiste à rédiger les règles des différents services. L'exécution est l'étape de mise en œuvre des règles par la génération des plans de composition. L'optimisation consiste à choisir le plan qui correspond aux critères de qualité spécifiés pour un cas donné de composition

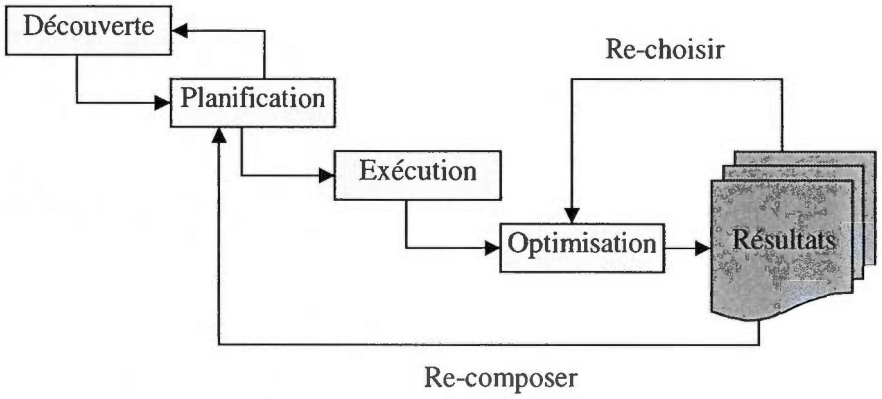

Figure 4.5 Le cycle de composition de services

4.2 La description de l'environnement de composition

Le système de composition est utilisé par différents acteurs intervenant pendant le processus de composition depuis la modélisation jusqu'à l'exécution. Nous distinguons trois types d'acteurs :

- **Le concepteur des règles de composition :** Il assure la modélisation des services de base qui seront utilisés lors de la composition. Il écrit les règles en définissant l'ensemble des pré-conditions et de post-conditions de chaque opération appartenant à un service web donné. Il fait la mise au point de la base de règles en testant la cohérence des règles représentant les services et en faisant des simulations de composition de plans. L'étape de mise au point est nécessaire car la qualité des plans en dépend lors de l'exploitation du système. La maintenance des règles est également assurée par le concepteur. Il ajoute de nouvelles règles pour inclure de nouveaux services. Il supprime celles qui correspondent aux services obsolètes ou devenus indisponibles. Il peut aussi modifier les règles si les services qu'elles représentent ont changé. Le travail du concepteur peut être fastidieux c'est pourquoi il est nécessaire de mettre à sa disposition des outils d'assistance. JESS en fournit quelques uns comme l'exécution des fonctions.

- **Le développeur de services agrégés :** La composition de services se fait par le développeur d'applications en recherchant les combinaisons possibles entre les services de base modélisés par le système. La composition peut comporter plusieurs itérations avant d'arriver à un résultat satisfaisant. La génération de plans est validée graphiquement par le développeur. Il peut aussi effectuer des simulations sur les plans générés. Il peut modifier les requêtes en spécifiant différentes entrées/sorties pour les services qu'il veut composer. Il faut souligner le fait que le développeur dispose toujours de l'option de compléter le plan généré par le système manuellement en utilisant un service extérieur non modélisé par le système. Cette option permet au développeur de partir d'un modèle de plan de départ et ainsi bénéficier de l'expertise de composition mise à sa disposition par le système. Souvent un développeur novice trouve des difficultés dans la composition, surtout s'il ne connaît pas les problèmes liés à la génération de plans. De ce point de vue le système peut être considéré comme un outil d'apprentissage. Le développeur peut suggérer au concepteur d'ajouter un service à la base de règles.

- **L'utilisateur final :** Il utilise les services composés que le développeur a construits. L'utilisation des services consiste à les exécuter pour répondre à des requêtes formulées par l'utilisateur. Avant d'exécuter un service, il faut le retrouver en faisant des recherches sur le registre. Un modèle de classification de services permet d'effectuer des recherches par des mots clés.

Le schéma de la figure 4.6 montre les différents cas d'utilisation du système.

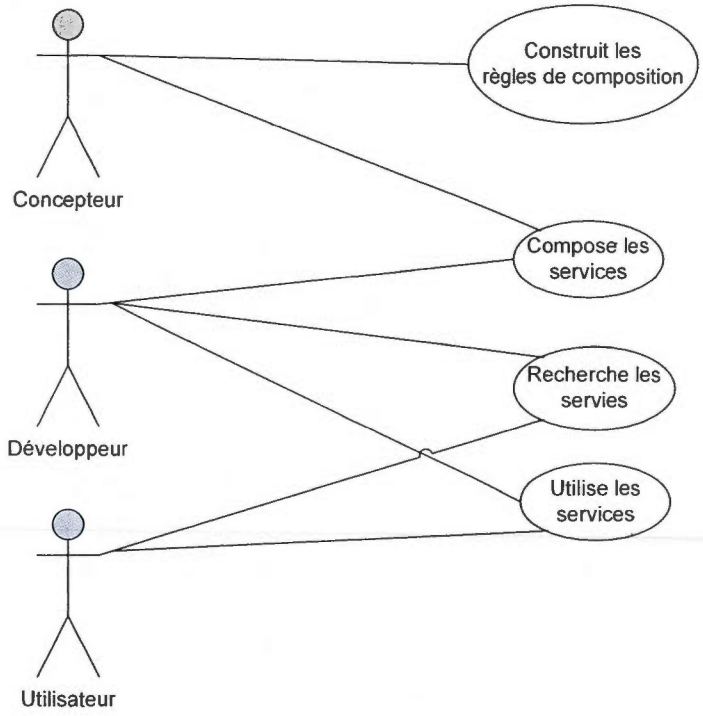

Figure 4.6 Les différents cas d'utilisation du système

Le système de composition comprend plusieurs modules intervenant dans la composition ou l'exécution de services agrégés. La communication se fait par des interfaces graphiques adaptées à chaque type d'utilisateurs. Le module d'interprétation des requêtes analyse la demande et détermine s'il s'agit d'une requête de composition ou d'exécution de services. Dans le premier cas, le module de composition est appelé et le processus de composition est enclenché.

L'analyse d'une requête de composition consiste à extraire les entrées et les sorties de service composite que l'on veut implémenter. L'interpréteur de commandes transforme ces entrées en un état initial de la base de faits de JESS. Ces faits sont insérés par le compositeur qui lance aussi l'exécution de JESS. A l'arrêt de JESS, le compositeur vérifie si la sortie, transformé en un état final, est atteinte lors de l'exécution. Dans le cas du succès, le flux de composition est présenté sous forme graphique; après quoi la génération du code BPEL peut être enclenchée.

Le descripteur du service composite, peut être enregistré dans le registre afin qu'il puisse être utilisé par des utilisateurs potentiels.

Si la requête provient de la part d'un utilisateur voulant effectuer des recherches sur le registre afin de retrouver un service, le module d'invocation de services est appelé. L'ensemble des descripteurs de services sont stockés dans la base des registres. Lors de la composition, ces descripteurs y sont retrouvés pour extraire les informations sur les types de données et les noms des opérations pour générer des plans de composition.

Dans la figure 4.7 sont indiqués les différents modules qu'on vient de décrire. Il faut remarquer la présence de deux niveaux de services. Les services composés que les développeurs ont générés et déployés pour être utilisés et les services de base qui sont décrit dans la base de règles. Les services composés générés par les développeurs ne sont pas tous déployés. Seuls ceux que le concepteur considère pertinents seront déployés. Tout changement de service de base doit être contrôlé par le concepteur car des services composés déjà déployés peuvent utiliser le service de base en question. Sa suppression ne doit se faire que si un autre service de base peut le remplacer. Le contrôleur d'accès permet de gérer l'authentification des utilisateurs et de leur donner l'accès uniquement aux services auxquels ils ont droit selon leurs profils.

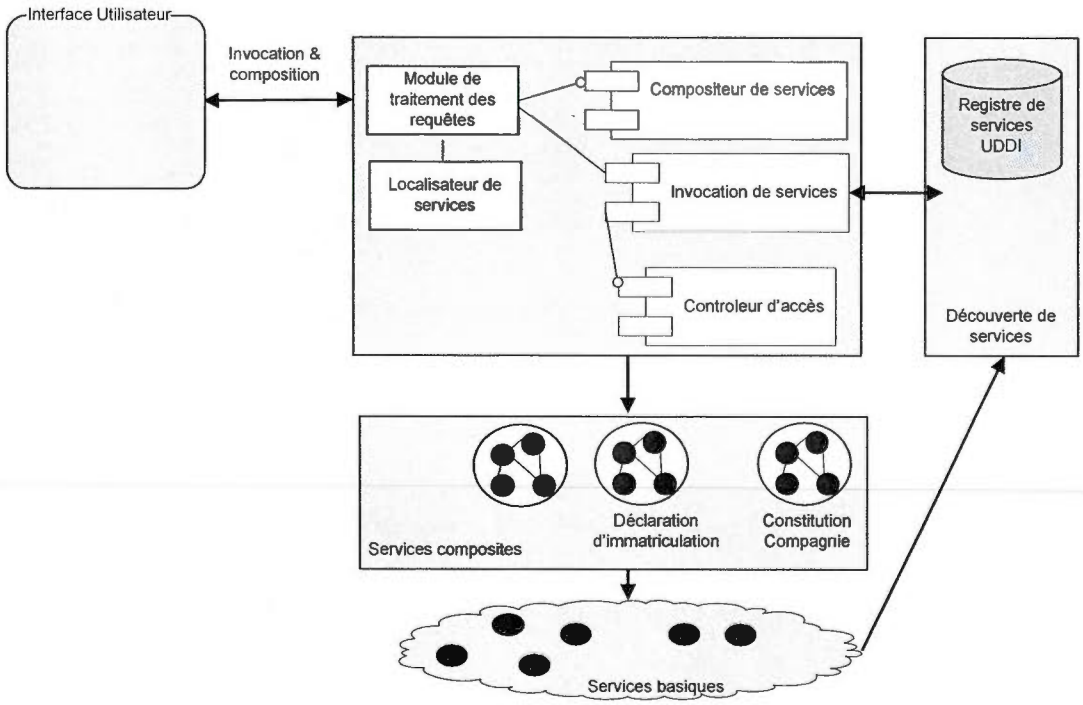

Figure 4.7 Le système de composition

4.4 Le chois de JESS comme engin d'exécution de règles

Nous avons déjà présenté JESS au chapitre 3. Son rôle dans la composition de services est primordial car il possède des caractéristiques suivantes:

- **Utilisation de règles et des prédicats :** Cette caractéristique permet de couvrir le besoin de formuler les règles de description sémantique de services et leur exécution. L'utilisation des règles de JESS pour exprimer la logique de composition de services est une caractéristique importante de notre système dans le sens où ces règles utilisent le formalisme de la logique du premier ordre. L'expression des contraintes liées à la composition par les prédicats est très riche. Contrairement à la logique des propositions qui est limitée à de simples faits ayant une valeur de vérité vraie ou fausse et qui ne traite pas les

variables. Nous allons voir dans la suite de ce chapitre comment représenter les opérations des services sous forme de règles et de prédicats. Dans la version actuelle, nous avons choisi de ne pas utiliser les fonctions et les négations dans les règles. Cette dernière possibilité pourrait être exploitée pour introduire de nouvelles fonctionnalités intéressantes dans le processus de composition particulièrement pour la mise au point des bases de règles décrivant les services.

- **Facilité d'intégration et d'interaction :** L'intégration des modèles que nous avons élaborés dans JESS constitue un atout majeur car il nous permet de construire des plans de composition à partir des structures internes de JESS et en même temps d'accéder à nos propres structures à partir de JESS. Étant écrit en Java (code source), JESS nous ouvre également d'autres possibilités d'interaction avec les différents API disponibles de java notamment l'API d'analyse de documents XML JAXP (Java API for XML Processing). Lors de la génération de code BPEL4WS, il est important de disposer de ce genre d'outils pour analyser les différents fichiers WSDL des services qui interviennent dans la composition. L'extraction des informations relatives aux types de données et aux paramètres des opérations, ainsi que les ports et les adresses des services, est nécessaire pour la génération de plans de composition. L'accès à l'arbre d'exécution généré au moment de déclenchement des règles représentant les services permet de disposer de l'ordre dans lequel les services impliqués seront invoqués. Les relations de dépendance entre les appels, les invocations parallèles et séquentielles seront retrouvées dans l'arbre d'exécution de JESS. Il faut noter que les performances de JESS dépendent de la structure des règles de composition c'est pourquoi le concepteur de règles doit prêter attention à la façon dont les pré-conditions sont exprimées. En général il est recommandé de garder le même ordre de pré-conditions lorsque deux règles utilisent les mêmes pré-conditions.

- **Utilisation du paradigme objet :** L'exploitation de l'orienté objet (de Java) pour compléter le modèle sémantique lié à la composition constitue également

un atout majeur offert par JESS. Dans la modélisation de la composition, en plus des règles, il est possible de représenter les entités de description de services par les objets Java et exploiter les concepts d'héritage et de surcharge ou de sur-définition de fonctions. Le formalisme objet est intéressant pour la représentation des connaissances liées à la composition. Sa combinaison avec les règles constitue un apport majeur. Nous allons voir dans la suite de ce chapitre comment exploiter les objets.

- **Utilisation des infrastructures logicielles (Framework) :** en Java, il existe des Frameworks supportant les services web pouvant être exploités dans l'implémentation du compositeur. WSFI (Web Services Framework Interface) en est un exemple. Il permet l'interaction avec les services en utilisant plusieurs protocoles de communication entre les services web et leurs applications clientes. La description de services se fait en WSDL mais la communication n'est pas limitée à SOAP, il peut également utiliser JMS (Java Message Service). Cette possibilité n'est pas exploitée en ce moment mais serait facilement intégrée dans une éventuelle extension.

L'intégration de JESS dans des applications Java est une caractéristique très utile dans la mise en pratique des idées avancées dans ce travail. Il est possible de créer des règles dynamiquement à partir d'une classe Java et de les exécuter. Cela est particulièrement intéressant pour le concepteur de règles qui maintient la base de connaissances. L'écriture du module de génération automatique de règles peut très bien être envisagée afin d'améliorer le processus de modélisation de la composition.

4.3 Le modèle de composition

Notre modèle de composition se base sur la représentation des opérations d'un service sous forme de règles. Dans la règle correspondant à une opération, on spécifie la sortie pouvant être déduite à partir d'une ou de plusieurs entrées. Un service est donc représenté par plusieurs règles; il existe autant de règles que le nombre d'opérations d'un service donné.

Une règle possède un nom significatif qui permet de faire l'association entre le fichier WSDL et la représentation sémantique pour extraire les informations concernant les messages et les types de données échangés entre le client et le service.

Le schéma de la figure 4.8 montre cette association.

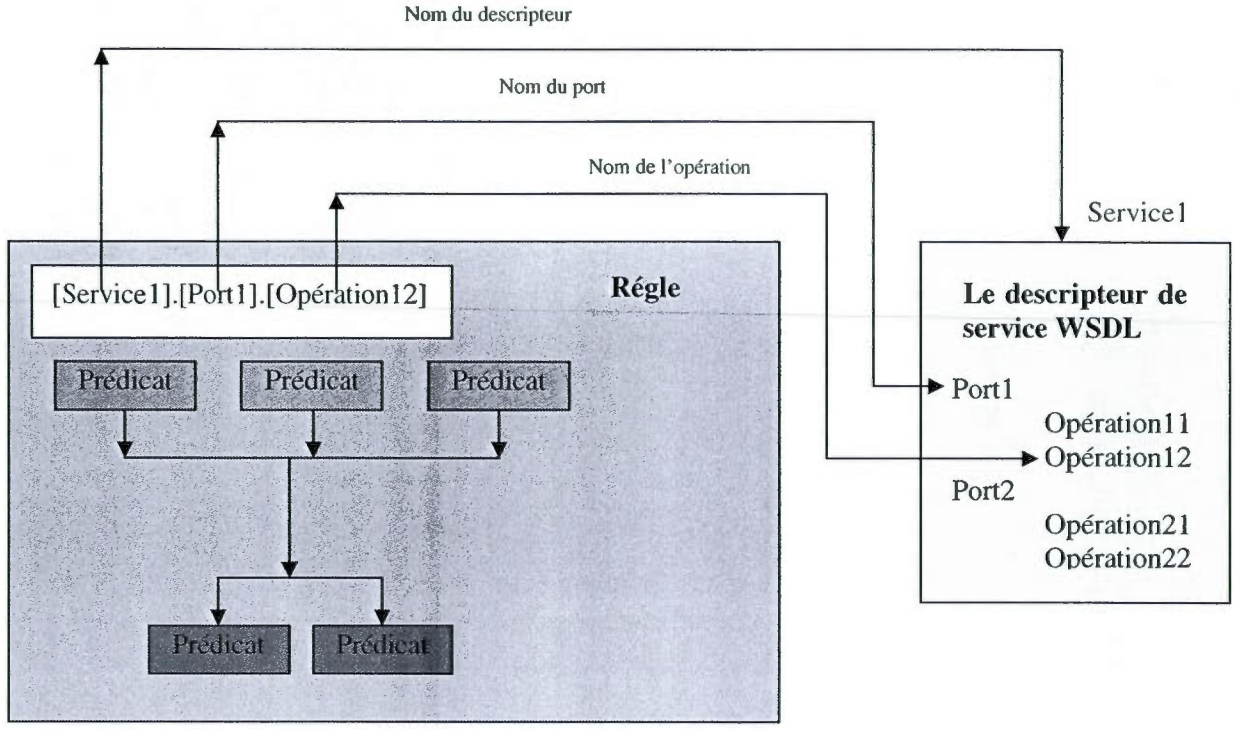

Figure 4.8 Association d'une règle à une opération de WSDL

Ce modèle permet de retrouver les informations pertinentes à la composition lors de la génération de plans. L'indexation des fichiers de descripteurs de services par les règles en utilisant cette convention, nous donne un moyen de remonter vers la description syntaxique d'un service. La recherche se fait dans l'ordre suivant :

1. La première partie du nom de la règles retrouve le fichier WSDL du service;
2. la deuxième partie retrouve dans le fichier WSDL le port dans lequel l'opération est décrite;
3. la troisième partie retrouve l'opération en question

4. une fois l'opération est localisée, tous les messages et leurs types respectifs sont également retrouvés et analysés.

4.3.1 Un exemple d'illustration

Nous partons d'un exemple [6] d'un service web simple ayant une seule opération. Nous allons nous en servir pour expliquer le modèle de composition du système. Le contenu du fichier WSDL est donné dans la figure 4.9. Il s'agit d'un service qui indique la valeur des indices boursiers.

Un document WSDL est une suite de définitions. L'élément « definitions » est la racine de fichier WSDL.

Les services sont définis en utilisant principalement 6 éléments :

- Types, utilisés pour définir les types de données servant à décrire les messages échangés;
- Messages, qui représentent une définition abstraite des données transmises. Un message est constitué de plusieurs parties dont chacune est associée avec un type donné;
- PortType, qui est un ensemble d'opérations abstraites; chaque opération référence les messages d'entrées et le message de sortie;
- Binding, spécifie le protocole et les spécifications des formats de données pour les opérations et les messages d'un portType donné;
- Port, spécifie l'adresse où le service est disponible;
- Service, utilisé pour agréger un ensemble relié de portTypes.

Dans l'exemple de la figure 4.9, l'élément « definitions » indique le nom du service qui est « StockQuote ». La spécification des espaces de noms permet de différencier les éléments et de référencer des spécifications externes comme SOAP, WSDL et des schémas

XML. L'attribut « targetNamespace » est une convention qui permet de faire des références au document WSDL (auto-reference). Dans l'exemple, on a targetNamespace="http://example.com/stockquote.wsdl".

Il faut noter que le document WSDL ne se trouve pas à cette adresse. Il s'agit uniquement de spécifier une valeur unique différente des autres valeurs d'espaces de noms utilisés dans le document.

Deux messages sont définis, le premier « GetLastTradePriceInput » représente la requête envoyée par les clients et le deuxième « GetLastTradePriceOutput » représente la réponse au client.

```
<message name="GetLastTradePriceInput">
  <part name="body" element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput">
  <part name="body" element="xsd1:TradePrice"/>
</message>
```

Chacun des messages de cet exemple contient une seule partie. Dans le message de la requête, cette partie représente le paramètre indiquant le nom de l'indice boursier. Pour le message de réponse, la partie représente la valeur de l'indice retournée par l'opération.

Les types des deux parties des deux messages sont définis dans la section « types » et sont référencés par l'attribut « element ».

Si la fonction avait plusieurs paramètres ou plusieurs valeurs de retour, on pourrait définir plusieurs parties pour les messages.

La seule opération du service est définie dans la section « PortType ». Elle possède un message d'entrée et un message de sortie. Le code suivant montre la déclaration de la méthode.

```
<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
```

```
</portType>
```

Le fichier WSDL du service est donné dans la figure 4.9

```
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
      <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/2000/10/XMLSchema">
         <element name="TradePriceRequest">
           <complexType>
              <all>
                 <element name="tickerSymbol" type="string"/>
              </all>
           </complexType>
         </element>
         <element name="TradePrice">
           <complexType>
              <all>
                 <element name="price" type="float"/>
              </all>
           </complexType>
         </element>
      </schema>
    </types>
    <message name="GetLastTradePriceInput">
       <part name="body" element="xsd1:TradePriceRequest"/>
    </message>
    <message name="GetLastTradePriceOutput">
       <part name="body" element="xsd1:TradePrice"/>
    </message>
    <portType name="StockQuotePortType">
       <operation name="GetLastTradePrice">
         <input message="tns:GetLastTradePriceInput"/>
         <output message="tns:GetLastTradePriceOutput"/>
       </operation>
    </portType>
    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
       <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
```

```
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>
</definitions>
```

Figure 4.9 Le descripteur WSDL du service des taux boursiers

4.3.2 Description de services par les règles

La description d'un service par des règles, consiste d'abord à définir un ensemble d'entités du monde réel sur lesquelles les règles agissent en modifiant les différents attributs des entités impliquées.

Si on reprend l'exemple de la section précédente, on peut définir une entité « IndiceBoursier » qui a deux attributs, le nom et la valeur de l'indice.

La règle correspondant à la seule opération du service est :

1. (defrule StockQuoteService. StockQuotePortType. GetLastTradePrice
2. (IndiceBoursier ?X)
3. (Connu Nom ?X)
4. ===>
5. (assert (Connu Valeur ?X)))

La ligne 1 indique le nom de la règle selon la convention que nous avons adopté dans notre modèle. Après l'exécution de la règle, on peut retrouver toutes les informations décrivant l'opération dans le fichier WSDL.

La ligne 2 indique une condition devant être vérifiée pour pouvoir déclencher la règle. Cette condition est vérifiée si l'entité ?X est un indice boursier et possède la valeur vrai.

La ligne 3 indique la deuxième condition qui doit être vérifiée pour déclencher la règle. Cette condition est vérifiée si le nom de l'indice boursier est connu (possède donc la valeur vrai)

La ligne 4 veut dire que si toutes les conditions sont vérifiées, on peut déduire le fait de la ligne 5. Dans l'exemple il y a un seul fait mais on peut en avoir plus.

La ligne 5 indique que la valeur de l'indice boursier ?X possède maintenant (après l'exécution de la règle) la valeur vrai. Noter l'opérateur « assert » qui permet d'insérer le nouveau fait dans la mémoire de travail.

La description de services par les règles est complétée par un modèle orienté objet. L'ensemble des objets correspondent à des entités du monde réel pour lequel on veut appliquer la composition de services.

Le seul objet de l'exemple est l' « IndiceBoursier » ayant deux attributs : Nom et Valeur

Il est important de constater que dans une règle il n'est pas nécessaire de connaître tous les attributs des entités impliqués car se sont les faits associés aux attributs et les relations qui existent entre les entités qui changent l'état de la base de faits. Dans l'exemple précédent, on a les faits suivants : IndiceBoursier X, premier fait, qui signifie que tout objet X qui est un indice boursier et dont on connaît le nom, deuxième fait, on peut connaître la valeur, troisième fait.

Le système n'impose aucune restriction sur la définition des entités et les relations entre les entités à partir desquelles on doit modéliser la sémantique relative à la composition.

Le concepteur est libre de définir ses propres entités et relations liées à la sémantique de composition des services qu'il veut modéliser. Le système offre donc un mécanisme de représentation de services sous forme de règles et d'objets.

Il existe deux types d'entrées d'une opération d'un service. Les entrées portant sur les conditions devant être spécifiées sur les entités et les relations entre ces entités. Les entrées portant sur les attributs ou les données de ces entités. Les mêmes types s'appliquent pour les sorties des opérations d'un service. Pour expliquer ce modèle, prenons l'exemple de service d'annuaire de yahoo. Il permet de retrouver l'adresse et le téléphone d'une personne à partir de son nom, prénom, ville et état. Il existe une seule entité qui intervient dans la description de la règle et qui est appelée X.

- Les entrées-conditions portant sur les entités : Personne(X) indique que X est une entité Personne
- Les entrées-données portant sur les attributs : Nom(X), Prénom(X), Ville(X), État(X) indiquent que l'opération du service a besoin de connaître ces 4 données.
- Les sorties-conditions : aucune pour cette opération
- Les sorties-données : Adresse(X), Téléphone(X) indiquent que ce service retourne l'adresse et le téléphone d'une personne X.

Dans l'exemple précédent, une seule entité est impliquée dans la règle modélisant le service d'annuaire. Dans l'exemple suivant, on va modéliser un service d'envoi de courriels où deux entités sont impliquées et une sortie-condition est spécifiée.

- Entités impliquées : X et Y
- Entrées-conditions : MessageCourriel (X) qui indique que X est un message

- Entrées-données : Sujet(X), Contenu(X), AdresseCourriel(Y) qui indique qu'on a besoin de spécifier un sujet et contenu pour le message X et une adresse courriel valide Y.

- Sorties-conditions : MessageEnvoyé(X,Y) signifie que le message X est envoyé à l'adresse Y.

Ces deux exemples montrent la puissance d'expression des prédicats et de l'orienté objet dans la représentation de la sémantique de composition de services.

4.3.2.1 Généralisation du modèle de composition

Les exemples précédents montrent que le problème de la composition de services peut se décrire sous-forme d'actions représentant les opérations d'un service web donné. Ces actions prennent la forme de règles de production exprimées par des prédicats. Les pré-conditions des règles sont des conjonctions entre les conditions portant sur les entités et les relations entre les entités d'une part; et d'autre part les faits connus portant sur un attribut ou plusieurs attributs des différentes entités impliquées.

Le fait connu correspondant à un attribut, *Attribut(X1,X2,...Xm)*, est définis par *Connu(Attribut,X1,X2,...Xm)*. Et ainsi le fait représentant une entrée d'une opération de service est *Entrée(X1,X2,....Xn)*. Dans la deuxième notation, il s'agit simplement d'un changement de variable (Attribut par Xn).

Les post-conditions sont des conjonctions des conditions portant sur les entités et les relations entre les entités et aussi entre les attributs des sorties. En d'autres termes, les mêmes principes s'appliquent pour les post-conditions des actions.

L'état initial de la base de faits est la conjonction entre tous les faits des entrées spécifiées pour le service composé que l'on veut générer.

L'état final de la base de fait est la conjonction de tous les faits représentant la sortie du service que l'on veut composer.

Dans ce modèle, nous n'avons pas utilisé la négation des faits ce qui a pour conséquence de générer toujours de nouveaux faits sans jamais en supprimer. La gestion de la négation peut rendre l'algorithme de composition complexe et par consequent la generation des plans de composition sera difficile.

4.3.3 La génération d'un plan de composition

Lorsqu'on veut générer un plan, le système lit les entrées et les sorties que le développeur a spécifié pour le service composite qu'il veut développer. Il transforme, ensuite, les entrées en faits initiaux qu'il insère dans la base de faits. Le système entame l'exécution des règles en partant des faits initiaux. Le déclenchement des règles, en chaînage avant, génère de nouveaux faits. Ces faits correspondent à des sorties des différentes opérations de services représentées par la base des règles. L'ensemble des règles déclenchées dépend des faits initiaux et des faits intermédiaires qui sont insérés pendant l'exécution. L'arrêt de l'exécution signifie qu'il n'y a plus de règles pouvant être déclenchées. Pour savoir si un plan correspond aux spécifications données par le développeur, le système vérifie si la base de faits contient les faits qui représentent la sortie du service composite.

Il faut noter que l'ordre d'exécution des règles n'est pas important car l'arbre d'exécution permet de savoir l'enchaînement de ces règles. Les mêmes faits sont déduits indépendamment de l'ordre de déclenchement des règles.

Dans le cas de succès, un plan est généré à partir de l'arbre d'exécution qui représente l'ensemble des opérations des différents services impliquées. L'analyse de l'arbre d'exécution permet de remonter toute la chaîne des règles et de générer un plan de composition. Ce plan peut être visualisé sous forme graphique. Si le développeur le désire, il peut transformer le plan. Une fois le plan graphique est visualisé, la génération du code BPEL peut être lancée. L'étape de génération de code sera abordée dans les sections prochaines. L'ensemble de ce processus est résumé dans la figure 4.10.

Pour illustrer le processus de génération de plan, nous allons modéliser 2 services de base à partir desquels on effectuera une composition. Le premier service permet de trouver

l'adresse d'une personne. Le deuxième donne l'itinéraire entre 2 points géographiques en spécifiant leurs adresses respectives.

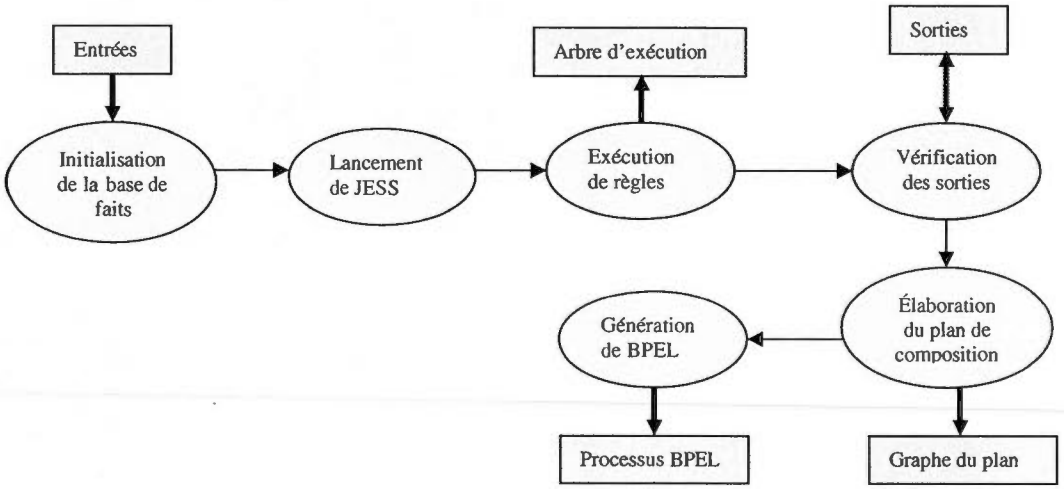

Figure 4.10 Le processus de génération de plans de composition

Les 2 services sont modélisés par les règles suivantes :
Règle 1 :

```
( TrouverCoordonnes.pt1.adresseTel

(Personne ?X) (Connu Prenom ?X) (Connu Nom ?X) (Connu ville ?X) (Connu Etat ?X)
===>
(assert (Connu adresse ?X)) (assert (Connu telephone)))
```

Règle 2 :

```
( TrouverItinairaire.pt2.Chemin
 (Connu ville ?X) (Connu Etat ?X) (Connu adresse ?X)
 (Connu ville ?Y) (Connu Etat ?Y) (Connu adresse ?Y)
===>
(assert (Connu Itiniraire ?X ?Y)))
```

Supposons que l'on veut trouver le chemin entre le domicile d'une personne A et celui d'une personne B. On spécifie alors les faits suivants :

(Personne A)
(Connu Prenom A)
(Connu Nom A)
(Connu Ville A)
(Connu Etat A)
(Personne B)
(Connu Prenom B)
(Connu Nom B)
(Connu Ville B)
(Connu Etat B)

Le lancement du chaînage avant déclenche 2 fois la règle 1 ensuite la règle 2 qui permet de trouver le chemin entre les 2 domiciles. L'arbre d'exécution correspondant est représenté dans la figure 4.11

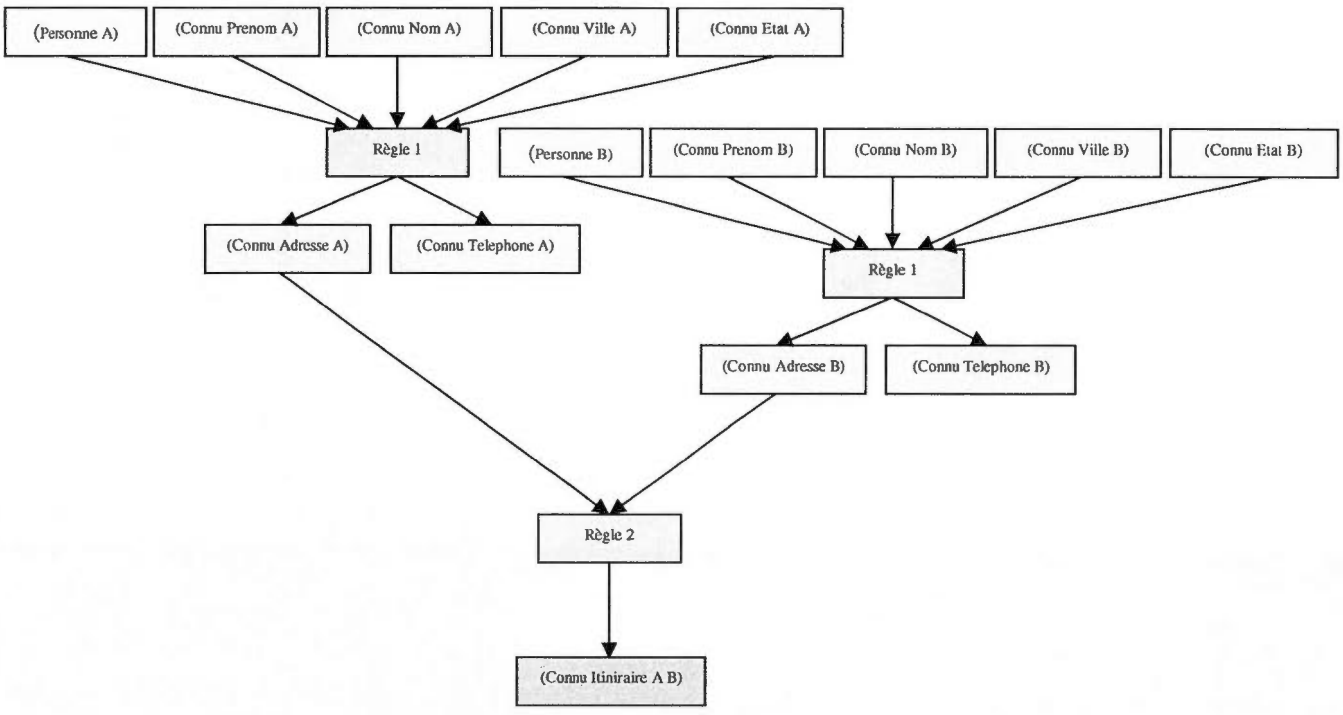

Figure 4.11 Représentation de l'arbre d'exécution

L'exemple précédent montre les caractéristiques suivantes du modèle de composition basé sur des règles :

- Une règle peut être déclenchée plusieurs fois avec des données différentes. Cela signifie qu'une opération d'un service peut être exécutée plusieurs fois pour une même requête;

- Le système peut ne pas générer de plan de composition pour un service composite donné. La richesse de la base des règles permet d'améliorer le traitement des requêtes de composition en répondant à plus de cas possibles

4.4 Le modèle conceptuel

Il existe une différence entre la modélisation de services de base et celle des services composés. Les services « basiques » sont en général une collection d'opérations qui sont appelées de manière synchrone ou asynchrone et se limitent donc à un simple échange de messages. L'ordre d'invocation des opérations n'est pas important et se fait selon les sollicitations de l'application cliente.

Par contre, les services composites sont dotés d'un comportement définis par le séquencement des messages échangés. L'interaction obéit à la logique de l'application cliente et l'ordre d'appel des opérations devient important.

Considérons, par exemple, une compagnie aérienne qui veut publier un service d'achat de billets d'avion composé de trois opérations : une opération de consultation des offres, une deuxième pour la réservation du choix et une éventuelle troisième pour le paiement. Dans la description WSDL du service, aucune information n'interdit le fait de réserver un vol qui n'existe pas ou de payer un autre qui n'a pas été réservé. Toutes les opérations sont au même niveau et elles sont, techniquement, toutes invocables.

La modélisation des services composés et des processus ne peut donc se faire uniquement par WSDL. Il faut compléter ce dernier par un autre modèle qui permet de prendre en compte les contraintes de la composition. Cependant les modèles qui nous intéressent dans le cadre de la composition sont ceux qui nous permettent de passer de l'arbre

d'exécution généré par JESS au code BPEL qui représente le service composé (Figure 4.12). Le service composé peut donc être considéré comme un processus.

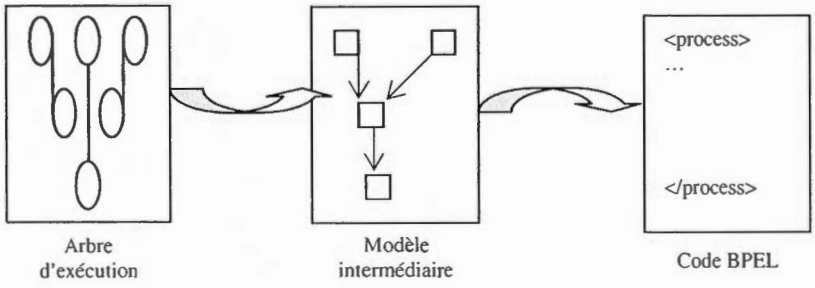

Figure 4.12 Transformation de l'arbre d'exécution en code BPEL

De l'analyse de ces modèles, ressort principalement deux types de représentation de processus :

- Les modèles qui sont basés sur UML
- Les modèles basés sur les systèmes de transition d'états

4.4.4 Les modèles UML

Les modèles UML et l'extension de sa notation utilisent les diagrammes d'activités pour décrire une composition de services. Cela suppose que le modèle de départ soit le diagramme d'activité UML dans lequel le concepteur du service composé modélise son service manuellement. Or il s'agit dans notre cas de rendre cette tâche de composition automatique. Il faut donc trouver un mécanisme de transformation du modèle généré par JESS vers le diagramme d'activité.

Pour illustrer la représentation des processus par UML, nous avons repris l'exemple du chapitre 2 auquel on a apporté des modifications pour illustrer certains constructeurs de BPEL. Le processus consiste à acheter des billets pour des employés d'une compagnie en spécifiant le nom de l'employé, sa destination, les dates du voyage ainsi que d'autres informations. Le processus invoque un service qui permet de vérifier le statut de l'employé.

On l'appellera « employeeTravelStatus ». En fonction du statut, il sélectionne le type de la réservation (1er classe, 2em classe). Le processus va ensuite faire appel aux 2 autres services de deux autres compagnies pour réserver une place en choisissant celle qui coûte le moins chère. Ces deux derniers services sont : « AmericanAirlines » et « DeltaAirlines » (Figure 4.13)

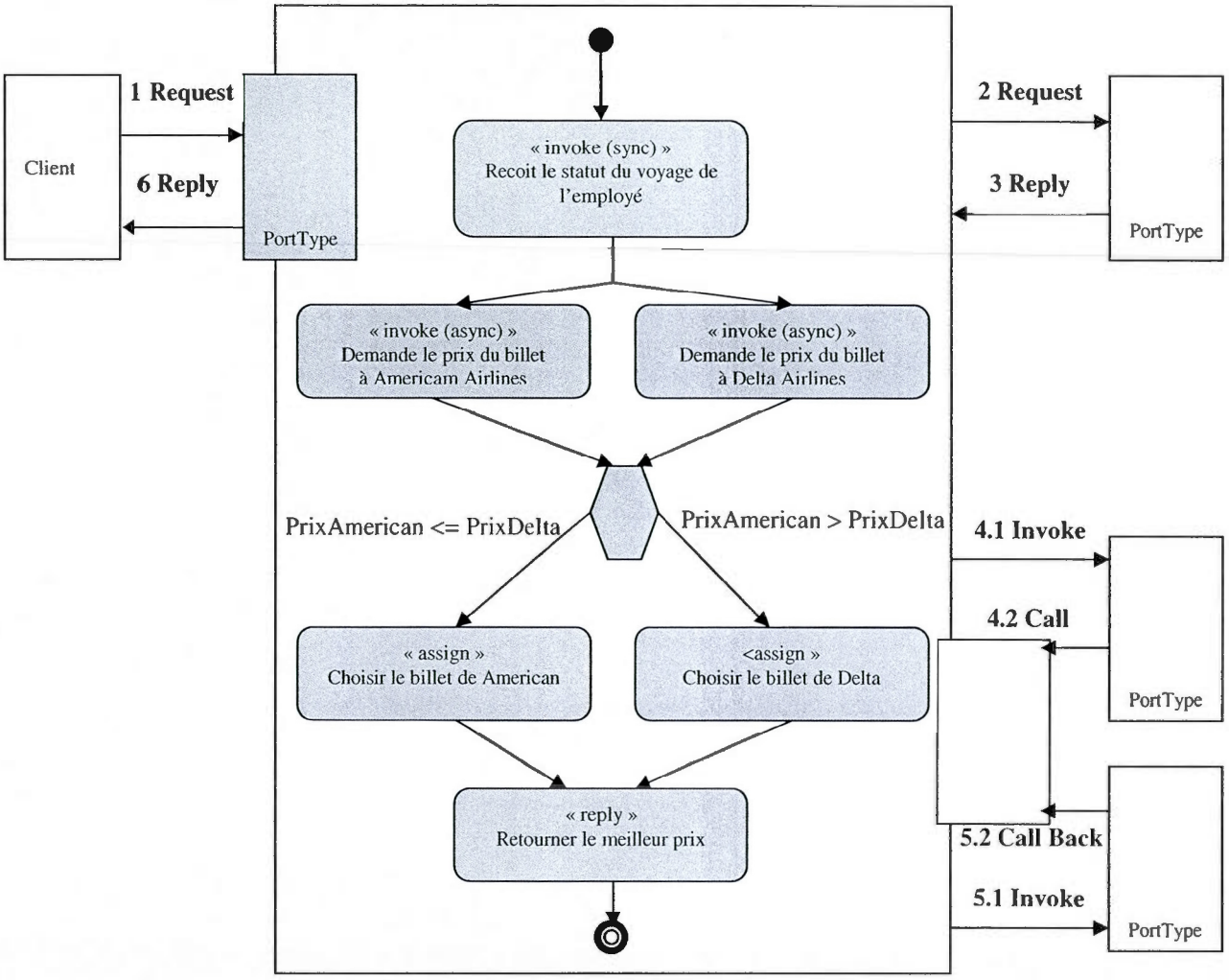

Figure 4.13 Exemple de processus BPEL pour la réservation de billets

Les modèles UML répondent en partie au besoin de la représentation de processus. Il y a même des prototypes de conversion de modèles UML directement en BPEL. Mais il y a un

long chemin à parcourir avant d'arriver à des résultats satisfaisants. De ce fait, des outils de conception sont rarement utilisés de bout en bout, de la modélisation à l'exécution.

La solution de composition que nous avons élaborée nous oblige à prendre comme modèle de départ l'arbre d'exécution généré par JESS. Pour faire le choix entre les deux types de modèles précédents, il faut trouver un moyen de faire le lien de l'un ou de l'autre de ces modèles avec l'arbre d'exécution. C'est pourquoi le choix du modèle basé sur UML a été vite écarté.

4.4.5 Le modèle de systèmes de transition d'états

La génération d'un plan de composition peut se décrire par un système de transition d'états. Le système de composition passe d'un état à l'autre lors de la génération d'un plan de composition en partant d'un état initial. Les différentes transitions sont provoquées par l'exécution des règles qui représentent les opérations des services et qui correspondent à des actions. La modélisation de services sous forme de règles nous permet justement de bénéficier du modèle conceptuel utilisé dans les systèmes de planification. Dans notre cas, nous avons utilisé JESS comme système de génération de plans.

Pour pouvoir générer le code BPEL qui correspond à un service composé généré par le système, il est nécessaire d'établir un modèle conceptuel représentant les différents opérateurs de BPEL qui lie ce dernier au plan de composition généré par le système. Ce modèle conceptuel nous permettra de rattacher des routines sémantiques aux règles de composition afin de construire le code BPEL relatif au service composé. Ce service est donc déployé pour être utilisé

Le système de composition travaille en réalité avec deux modèles différents mais qui se complètent. Le premier est le modèle de planification dont les éléments de la composition sont des opérateurs. Ces opérateurs sont les règles qui opèrent sur la base de faits qui reflète à chaque étape du processus de composition l'état dans lequel le système de composition est rendu. Le deuxième est le modèle de composition dont les éléments sont des services atomiques. Il indique les différentes opérations impliquées avec les paramètres de chaque

opération. L'ensemble des opérations des services relatifs à une requête de composition sont représentés par des activités en BPEL (Figure 4.14).

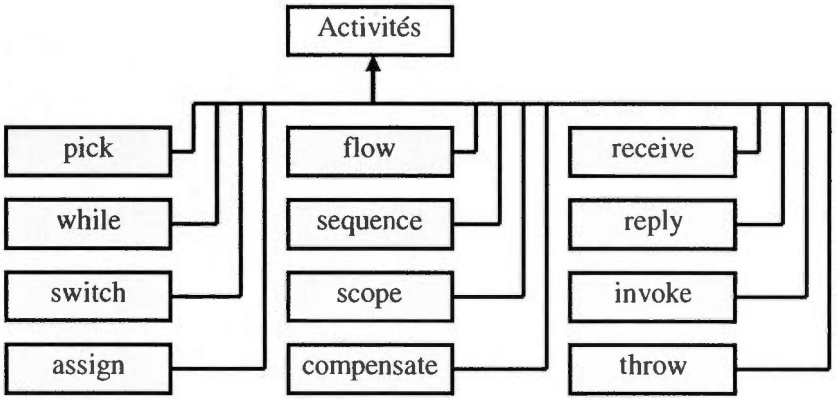

Figure 4.14 Les activités de BPEL

La représentation des activités de BPEL par le modèle conceptuel que nous décrivons dans cette section nous permettra de générer le code BPEL, qui représente le service composé, à partir de ce modèle.

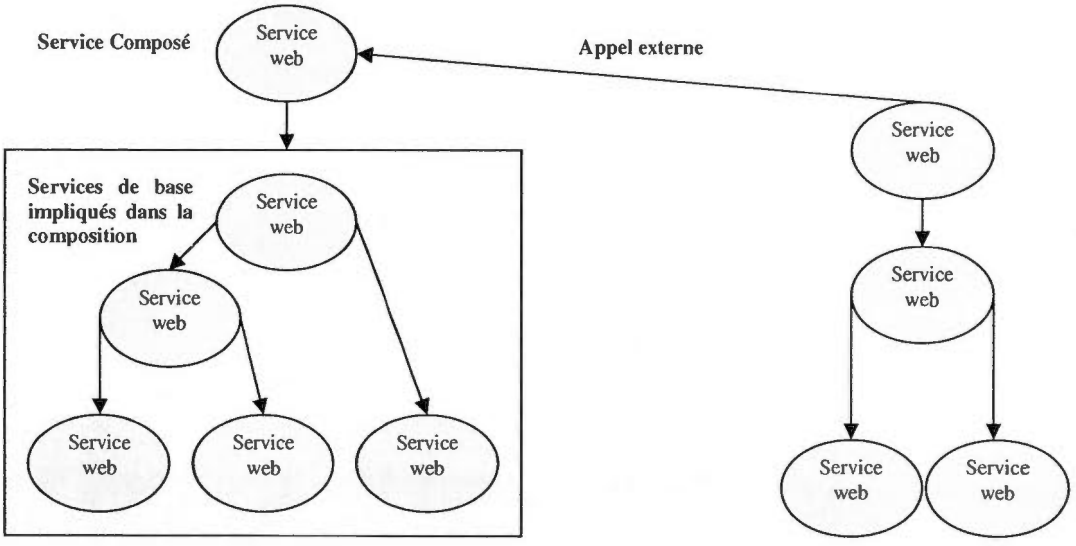

Figure 4.15 Invocation d'un service composé de l'extérieur

Le service composé pourrait alors être représenté par un processus BPEL qui va être appelé de l'extérieur comme tous les autres services existants (Figure 4.15).

Un processus défini en BPEL est une composition d'activités (Figure 4.16) dont le modèle est un tuple *(V, D, R)* où :

- *V* est un ensemble fini de variables
- *D* est le domaine de définition des variables de *V*
- *R* est un ensemble fini de règles défini de la façon suivante :

  Action : *Pre(V) → post(V)*

  Pre(V) : est une pré-condition

  Post(V) : est une post condition

  Les pré-conditions et les post-conditions sont des expressions booléennes sur la variable *V*

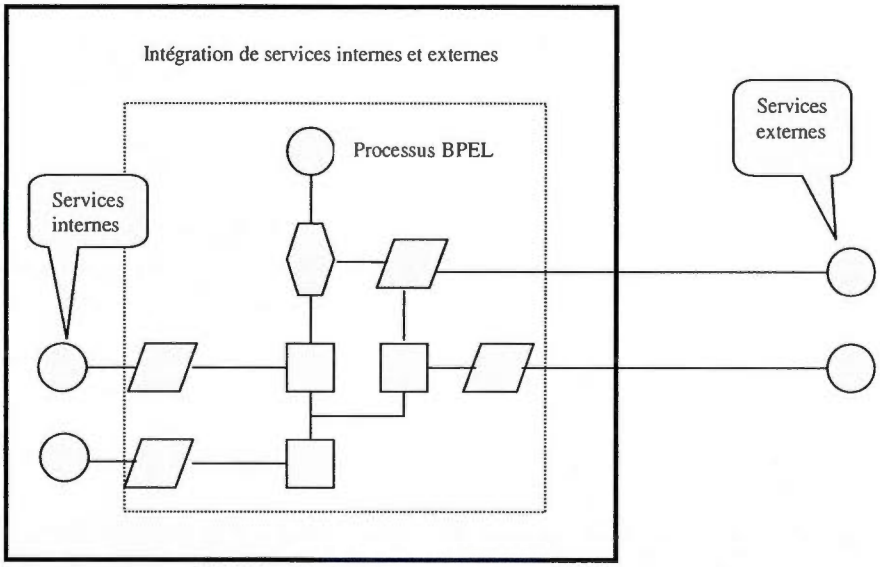

Figure 4.16 Représentation d'un processus BPEL par des activités

Le modèle de processus est donc un système de transition d'états car pour modéliser un processus, il faut représenter toutes ses activités et l'ordre d'exécution de ses activités en

utilisant des variables et des règles. Dans les sections suivantes, nous allons représenter les différents types d'activités par des modèles formels.

4.4.5.1 Modélisation des activités

On distingue sept activités de base en BPEL qui ne peuvent comporter aucune autre activité. Elles constituent les blocs de construction d'un processus et plus précisément d'un service composé dans notre cas. Chaque activité est translatée dans le modèle conceptuel selon ses propres règles de transition. Nous avons élaboré un modèle basé sur les diagrammes d'états pour représenter la sémantique de construction de code BEPL liée à la composition en partant de l'arbre d'exécution généré par JESS.

Ce modèle consiste à considérer D comme un domaine de variables fini et la valeur vide 0 comme appartenant au domaine et chaque variable possède un domaine D.

Nous pouvons alors définir une activité dans un processus, qui est aussi un service composé, comme une règle de transition. Il fait changer l'état du système en partant d'un état initial « activité-début » vers un état final « activité-fin ». L'activité dispose d'une variable d'entrée « inVar » et d'une variable de sortie « outVar » qui appartiennent à V et qui sont impliquées dans la transition. La transition est étiqueté par un nom « nomTransition ». Le schéma de la figure 4.17 montre comment représenter une activité dans le modèle conceptuel. inVar, etatVar et outVar représentent les variables d'états. nomTransition représente le nom de l'action ou de la règle qui a provoqué la transition.

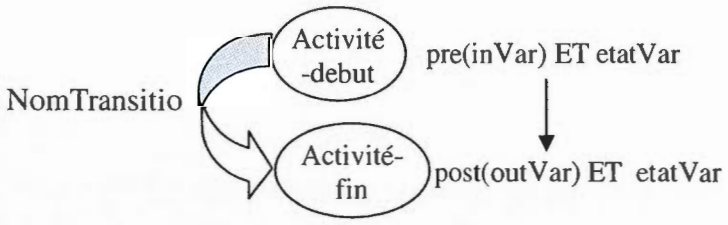

Figure 4.17 Modélisation d'une activité

Les noms affectés aux états et transitions doivent être uniques. A chaque lancement d'une requête de composition de services, on peut générer un numéro qui sera inclus dans les noms des états et des transitions afin de s'assurer de l'unicité des noms attribués.

Dans certains cas, la modélisation du comportement interne d'une activité est nécessaire si elle comporte plusieurs états intermédiaires. Donc, une activité disposant de plusieurs états intermédiaires se représente par le passage d'un état de début vers un état de fin en passant par plusieurs états intermédiaires (Figure 4.18)

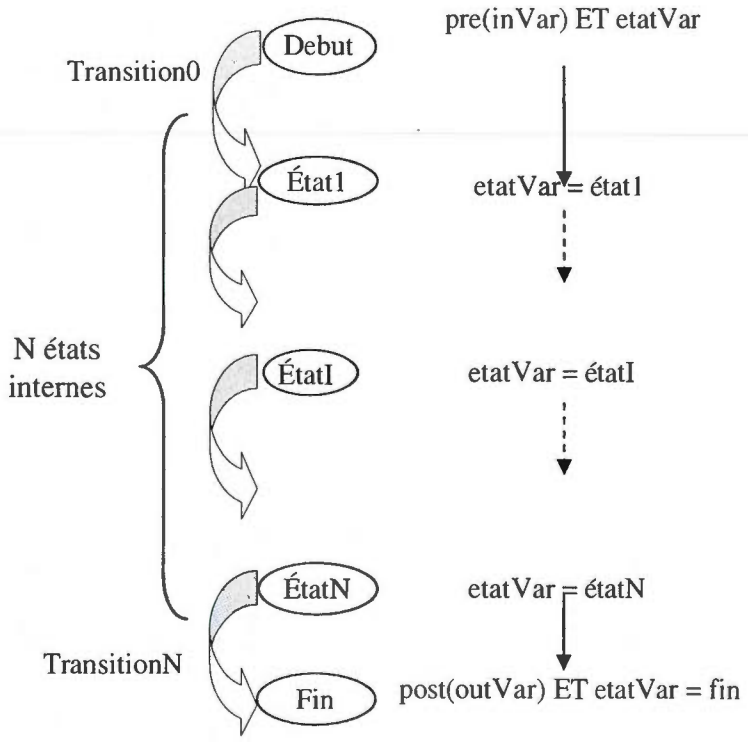

Figure 4.18 Représentation d'une activité avec plusieurs états intermédiaires

4.4.5.2 Modélisation des activités de base

Dans cette section nous allons modéliser les activités de base de BPEL. Toutes les activités seront modélisées par les éléments définis dans la section précédente à savoir les variables d'états, les actions et les transitions. Cependant, nous allons spécifier les représentations de trois activités. Pour les autres activités, nous allons les donner en annexe (Annexe A).

L'activité de réception d'une requête « receive »

Elle est utilisée pour recevoir des requêtes d'exécution du processus (service composé). La requête en question constitue le message d'entrée dans le processus.

Lors de la réception d'un message SOAP, on vérifie le type du message pour savoir si ce dernier correspond au type prédéfini dans le processus. Si c'est le cas, on initialise une variable messageRecu pour indiquer le fait qu'on a reçu le message afin de déclencher une transition. Le tableau 1.1 montre les différents éléments de représentation de l'activité

Les variables d'états	messageSoap, messageRecu, etatVar = {debut_receive,fin_receive}
Les variables internes	typeMessage est une chaine de caractères représentant le type accepté par l'activité receive
Action ou événement	Receive
Les règles de transition	(etatVar = debut_receive ET messageSoap.type = typeMessage) → (MessageRecu = messageSoap ET etatVar = fin_receive)

Tableau 4.1 Représentation de l'activité « Receive »

L'activité d'invocation de services « invoke »

Elle sert à invoquer une opération disponible sur un port de l'un des partenaires impliqués dans le processus. L'ensemble des paramètres de cette opération constitue le message soap envoyé au service et représente l'état avant que l'opération d'invocation soit effectuée (voir le tableau 4.2 pour les détails).

Les variables d'états	inVar,outVar, etatVar = {debut_invoke,fin_invoke,wait}
Action ou événement	invoke, receive
Les règles de transition pour l'invocation asynchrone	(etatVar = debut_invoke ET exist (inVar)) → (etatVar = fin_invoke)
Les règles de transition pour l'invocation synchrone	• (etatVar = debut_invoke ET exist(inVar))→ (etatVar = wait) • (etatVar = wait) → (etatVar = fin_invoke ET exist(outVar))

Tableau 4.2 le modèle de représentation de l'activité « invoke »

Il faut noter la différence entre l'invocation synchrone et l'invocation asynchrone. Dans le premier cas, il est nécessaire d'avoir une variable d'entrée inVar et une variable de sortie outVar. Dans le deuxième cas, uniquement la variable d'entrée car l'application cliente n'attend pas de réponse de la part du service.

L'envoi de messages entre le service composé et les services de base dépend de trois paramètres :

1. **Le modèle d'échange de messages ou MEP (Message Exchange Patterns) :** Les messages envoyés entre deux parties, sont reliés les uns aux autres et donc dépendants entre eux ou indépendants les uns par rapports aux autres. Par exemple on peut avoir un modèle de type « In-only » qui consiste à envoyer un message sans attendre de réponse. Un deuxième modèle « in-out » peut être un envoi de message suivi d'une réception de réponse. Le concept MEP change constamment et le nombre de modèles est illimité.

L'implémentation de ces modèles dans le cadre des services web se limite à certains d'entre eux.

2. **Le mode synchrone ou asynchrone :** Quand un service est appelé, soit on bloque sur l'appel jusqu'à ce qu'on reçoit une réponse (blocking) ou on continue le traitement sans attendre un message de retour (nonblocking). L'attente de messages, dans le cas asynchrone peut se faire par un autre thread qui roule en arrière plan.

3. **Le comportement de la couche transport ou one-way/two-way :** Les protocoles de transport sont catégorisés comme étant « one-way » ou « two-way ». Le type « one-way » réduit la complexité de communication entre les services web car les messages reliés empruntent des canaux de transport différents. Par contre, les services web utilisant le type « two-way » ont la possibilité d'exploiter le protocole dans un sens seulement. Par exemple si on utilise http, on peut retourner « http 200 » pour indiquer qu'on n'a pas de réponse à donner ou établir une autre connexion http pour le message de réponse.

4.4.4.3 Modélisation des activités structurées

Les activités structurées indiquent l'ordre dans lequel une collection d'activités est exécutée. Le flux de contrôle du processus est décrit par la composition des activités de base en activités structurées. On distingue les activités structurées suivantes :

- Le contrôle de flux séquentiels est fourni par les activités « sequence », « switch » et « while »
- La concurrence et la synchronisation entre les activités sont assurées par « flow »
- Les choix non déterminés basés sur des événements externes est fourni par « pick »

Les activités structurées sont modélisées par la combinaison des règles de transition qui expriment le comportement de chaque activité imbriquée (ou incluse dans l'activité structurée) et l'ordre d'exécution de l'ensemble des activités imbriquées. Dans la suite de cette section, nous allons décrire l'activité structurée « while » et les règles exprimant l'ordre d'exécution. Les autres activités structurées seront décrites en annexe A.

L'activité « while »

Cette activité répète l'exécution d'une activité imbriquée qu'on va designer par A. La figure 4.19 montre le graphe qui la représente.

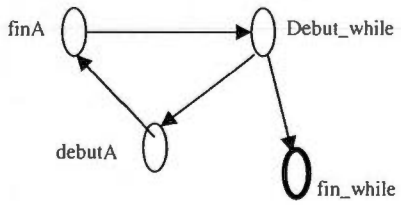

Figure 4.19 Diagramme de l'activité « while »

Son modèle est donné dans le tableau 4.3

4.5 La construction des services composites

La génération d'un processus BPEL se base sur l'arbre d'exécution de JESS. La structure de l'arbre reflète exactement le flux d'exécution du processus. Pour illustrer cette correspondance, nous allons partir de l'arbre d'exécution de la section 4.3.3 qui représente le service composite de la recherche d'itinéraire entre deux adresses et construire un arbre général représentant n'importe quel cas de composition.

Les variables d'états	W inclus ou egal à V, etatVar = {debut_while, fin_while, debutA,finA}
Action ou événement	while, while_fin et A.nomEvent
Les transitions	Debut_while → debutA DebutA → finA FinA → debut_while Debut_ while → fin_while
Les règles de transition	EtatVar = debut_while ET pre(W) → etatVar = debutA EtatVar = finA → etatVar = debut_while EtatVar = debut_while ET non pre(w) → etatVar = fin_while

Tableau 4.3 Le modèle de représentation de l'activité « while »

L'examination de l'arbre d'exécution montre qu'il existe deux types de nœuds :

- Les nœuds de type ET qui représentent une conjonction d'une ou de plusieurs conditions qui ont été vérifiées et correspondent aussi aux opérations des différents services impliqués.

- Les nœuds de faits qui représentent les conditions de déclenchement des règles. Ces conditions correspondent aux entrées des opérations des différents services impliqués

Si on parcourt l'arbre du bas vers le haut, on constate qu'il existe 2 niveaux de nœuds de type ET. La numérotation de ces niveaux en partant du bas vers le haut nous donne le niveau 0 suivi du niveau 1. L'ordre dans lequel les opérations sont invoquées est l'ordre inverse des niveaux; c'est à dire que le niveau le plus faible (0) correspond à la dernière opération qui doit être invoquée par le processus. Et celui le plus élevé représente la ou les premières opérations qui doivent être invoquées. La figure 4.20 généralise le modèle.

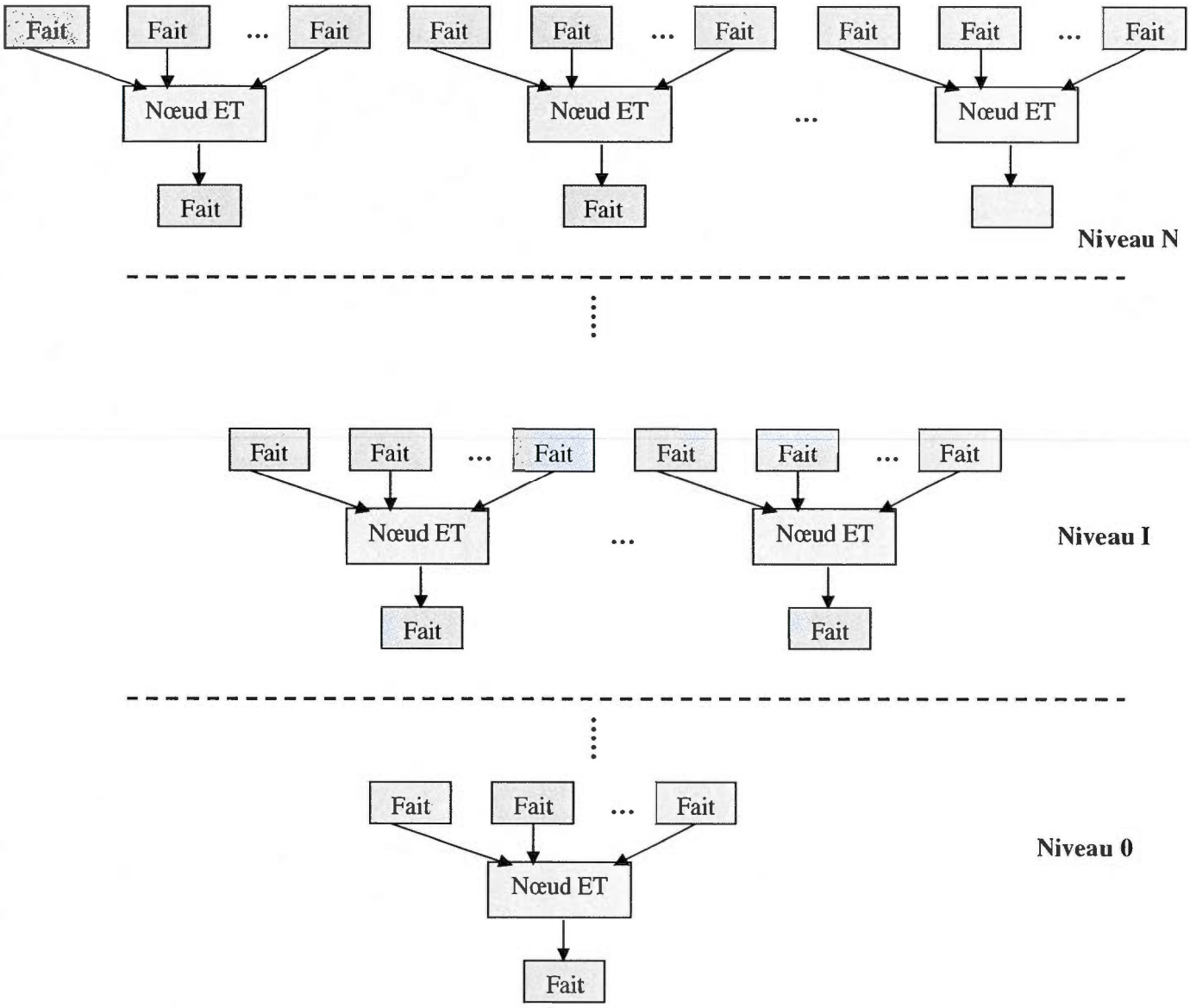

Figure 4.20 La généralisation du modèle de représentation de l'arbre d'exécution

La figure précédente montre que les opérations du niveau N sont invoquées en premier et l'opération du niveau 0 en dernier. Les opérations du même niveau peuvent être invoquées en parallèle.

Il faut noter que si le niveau 0 comporte plusieurs nœuds de type ET, cela veut dire que l'on dispose de plusieurs plans de composition; ce qui correspond à plusieurs processus. Le traitement de ces cas peut se faire sur la base d'un modèle qualitatif en choisissant le plan ayant un coefficient de qualité le plus élevé par exemple. Des algorithmes utilisant l'incertain peuvent être appliqués. Dans le travail que nous présentons, nous n'avons pas abordé cette question. Une extension de ce travail explorant cette possibilité, peut être entreprise.

L'algorithme de génération de code peut se décrire de la façon suivante :

1. On initialise le niveau courant i à 0
2. On parcourt l'arbre d'exécution à partir du niveau 0
3. On définit une activité de séquence
4. On examine le nœud représentant le fait courant (celui du niveau représentant la sortie du service) soit :
    a. Retrouver la règle ayant déduit ce fait
    b. Examiner la description WSDL de l'opération correspondant à la règle courante
    c. Définir les variables ayant pour types respectifs les types de messages définis dans WSDL. Ces types étant ceux des paramètres de l'opération.
    d. Mettre la définition de la variable dans la section des définitions
    e. Générer une instruction d'invocation (invoque) à l'opération en spécifiant les variables de d comme variables d'entrée
    f. Définir la variable de sortie qui correspond à la valeur de retour de l'opération. Le type de cette variable étant le même que celui de cette valeur

g. Mettre la variable dans la section des définitions

5. Si toutes les règles du niveau courant ont été examinées et elles sont au moins en nombre de deux, alors mettre les invocations de ce niveau dans une activité dans la portée du l'instruction « flow ». Ces invocations peuvent s'effectuer en parallèle

6. Passer au niveau i +1

7. Reprendre le traitement à partie de 4

8. On ferme la séquence

L'algorithme précédent est récursif et reflète la structure de l'arbre d'exécution des règles déclenchées lors de l'élaboration du plan de composition.

4.6 L'incertitude de la composition

La composition de services peut conduire parfois à des résultats incertains lors de la génération de plans de composition. Pour comprendre comment cela peut se produire, considérons une entité E ayant trois attributs a, b et c.

Supposons qu'on a deux services S1 et S2 ayant respectivement les opérations O1 et O2. Si :

- l'opération O1 de S1 donne b(E) à partir de a(E),
- l'opération O2 de S2 donne c(E) en partant de b(E) et
- l'on veut composer S1 et S2 pour obtenir S3 ayant l'opération O3 qui à partir de a(E) fournit c(E),

alors cela peut nous conduire à des résultats incertains (Figure 4.21).

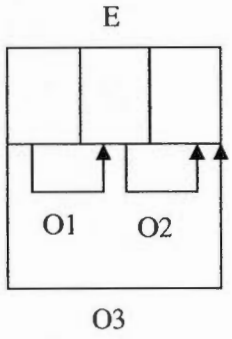

Figure 4.21 Cas d'incertitudes

Un exemple de services causant l'incertitude

Si on considère S1 comme un service donnant l'adresse d'une personne à partir de son nom et S2 comme un service donnant le téléphone d'une personne à partir de son adresse. Supposons que plusieurs personnes habitent à la même adresse et possèdent des numéros de téléphones différents. Si deux personnes P1 et P2 habitent la même adresse et ayant chacune un numéro, alors le service S3 retourne les deux numéros quand on lui demande celui de la personne P1. La raison de cela est que l'adresse ne permet pas de déterminer de façon unique le numéro de téléphone ni le nom d'une personne.

4.7 Conclusion

La composition de services offre un cadre de travail intéressant pour supporter la gestion des processus et l'intégration d'applications. Des standards sont définis et arrivent au stade de maturité. Des outils exploitant ces standards commencent à émerger. Le prototype réalisé dans le cadre de ce travail rentre dans ce contexte. Nous avons présenté une approche basée sur les standards industriels et utilisant les mécanismes de systèmes à base de règles. La mise en œuvre des idées de l'approche mixte a été rendue possible grâce à l'utilisation de JESS comme moteur de règles et du langage Java qui offre une infrastructure de librairies très riche en matière de manipulation de standards de services web. L'implémentation de ce prototype a montré qu'il est possible d'exploiter les techniques utilisées dans les systèmes

experts et de les intégrer dans l'infrastructure industrielle d'implémentation des services web pour réaliser un système de composition de services.

Nous avons utilisé un cycle de composition itératif pour répondre aux contraintes de développement d'applications industrielles. L'architecture du système de composition repose sur un modèle ouvert qui permet de greffer d'autres outils pouvant améliorer la qualité des services composés.

CHAPITRE V

IMPLEMENTATION ET MISE EN ŒUVRE

5.1 Introduction

Le prototype réalisé dans le cadre de ce travail, nous a permis de consolider nos idées sur la mise en pratique de l'approche mixte et de l'utilité d'utiliser les standards des services web.

Le prototype a été testé sur un ensemble de services web relevant du domaine de distribution d'énergie électrique (Hydro Québec). Ces services sont utilisés pour remettre en charge le réseau électrique après avoir subi une panne quelconque, due à une défectuosité, d'un ou plusieurs équipements. Les équipements sont nombreux et variés et leur comportement évolue au cours de temps. Leur fiabilité dépend de leur âge. Des décisions sont prises sur une base périodique pour leur remplacement ou leur réparation.

D'autres tests ont été effectués sur les exemples présentés dans le chapitre 4. Nous allons présenter les plans sous forme graphique générés par ces exemples.

5.2 Description de l'environnement d'implémentation

Le prototype a été réalisé en utilisant Eclipse comme environnement de développement et java (JDK 1.4) comme langage de programmation. Le déploiement des services web sur lesquels nous avons expérimenté le prototype s'est fait sur Axis. Nous avons utilisé son outil de génération automatique de clients à partie des interfaces WSDL (WSDL2Java).

Les services composés seront déployés sur un outil d'IBM, BPWS4J (Business Process Web Services for Java) qui est un engin d'exécution de processus BPEL4WS. Cependant, la

phase de génération de code n'est pas terminée, vue l'ampleur de travail de programmation exigé.

Le prototype est développé sous forme d'une application web et suivant le modèle MVC (Model View Controler) voir figure 5.1

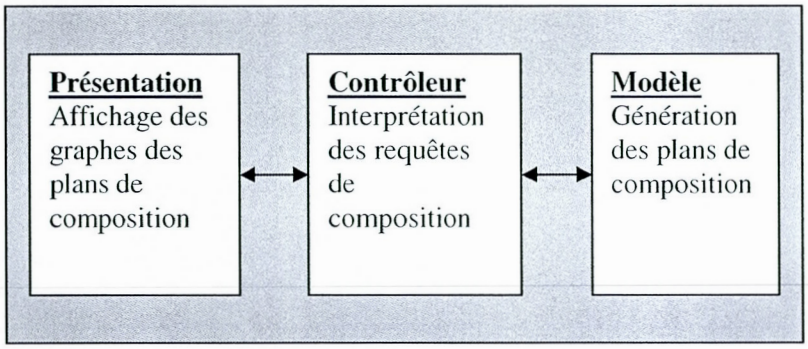

Figure 5.1 Le compositeur de services web

Nous avons utilisé JESS comme moteur d'interprétation de règles de description sémantique de services, voir chapitre 3.

L'ensemble de ces outils ont été déployés sur Tomcat 4.1. La figure 5.2 montre l'environnement technologique sur lequel le compositeur est construit.

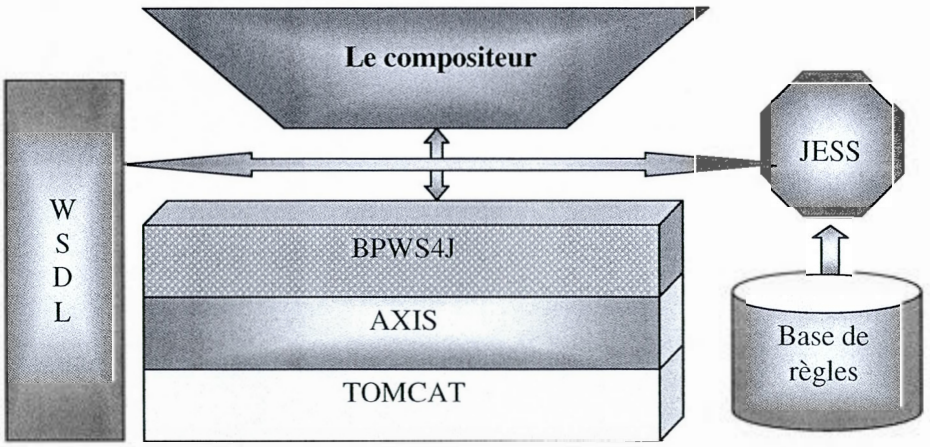

Figure 5.2 Environnement technologique du compositeur

5.2.1 Aperçu d'axis

Axis [7] est un engin SOAP qui est offert par Apache en « open source ». Il a d'abord existé sous le nom d'Apache SOAP. Il a gagné beaucoup d'intérêt auprès de la communauté des développeurs de services web. Axis2, qui est la nouvelle version, a apporté beaucoup de nouveautés en termes de développement de services web. Il possède de nombreux outils facilitant l'implémentation de services web. Ses principales caractéristiques sont :

- Offre plusieurs modèles d'échange de messages (Message Exchange Patterns)
- Permet de faire des appels synchrones et asynchrones
- Offre un support de déploiement de services par des archives sans redémarrage de conteneur de servlets
- Supporte SOAP1.1 et 1.2 et les protocoles HTTP, SMTP, JMS, TCP

5.2.2 Fonctionnement d'Axis

En SOAP, les acteurs qui prennent part dans une interaction de service web sont appelés les nœuds SOAP. Il y a en général un émetteur et un récepteur. Chaque nœud SOAP est écrit dans un langage quelconque (Java, C++ ...). Axis prend en charge tous les aspects de gestion de messages et de sécurité. Le développeur intervient dans la gestion de la logique de son application qui se trouve avant l'envoi de message par le client et après la réception de message par le récepteur. La figure 5.3 montre le modèle de traitement des messages soap par Axis et les niveaux d'intervention d'un développeur.

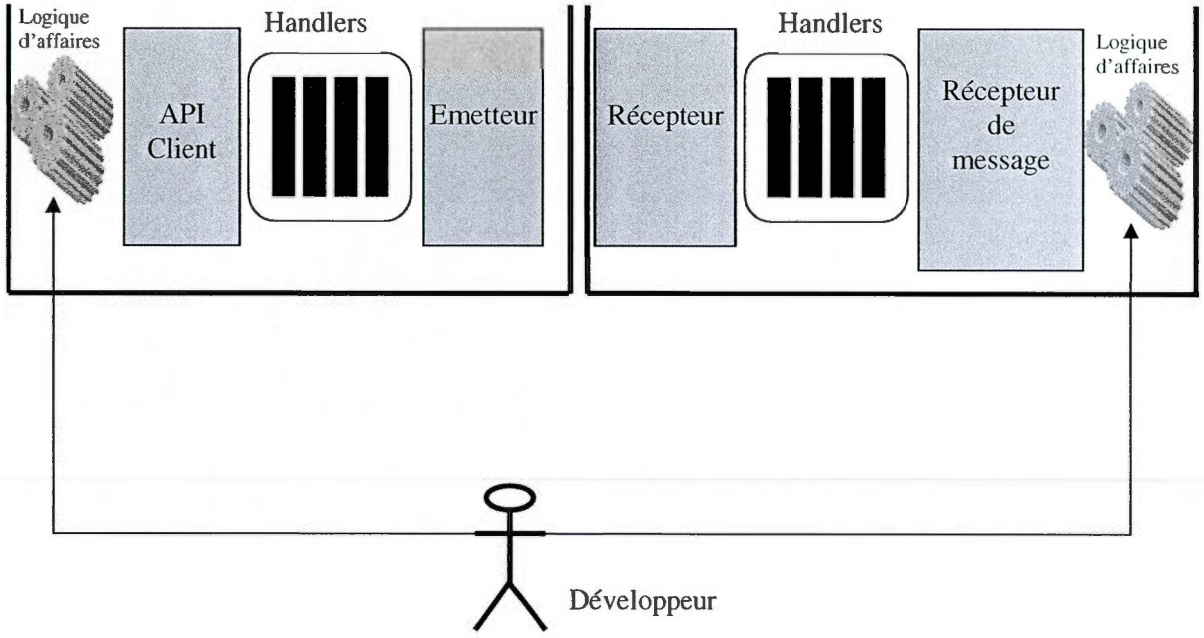

Figure 5.3 Le modèle de traitement des messages SOAP par axis

5.3 Définition des services web de remise en charge

Nous somme partis d'un ensemble d'applications que nous avons transformé sous forme de services web. Ces applications étaient utilisées dans des contextes différents et par des acteurs différents. Il fallait définir l'interface d'interaction de ces services et identifier un sous ensemble de fonctionnalités de chaque application qui seront offertes sous forme de services web. La partie interface graphique a été éliminée. Nous avons considéré les différents scenarios d'utilisation de chaque service afin d'établir une interface selon laquelle chaque service devrait être exposé. La définition des interfaces d'interaction d'un service web consiste essentiellement à spécifier comment les requêtes sont reçues et comment les acheminer vers la couche de traitement. Dans la figure 5.4, nous avons présenté les deux couches d'un service web. L'interface d'interaction est la couche qui reçoit les requêtes et les transmet vers la couche de traitement après avoir effectué un certain prétraitement. La formulation de la réponse est également assurée par l'interface avant d'être envoyée vers le

client. La couche de traitement implémente toute la logique des fonctionnalités offertes par le service. Il s'agit seulement d'isoler le code de traitement de celui de l'interface graphique.

L'implémentation des différents services sous forme de couches permet de séparer les responsabilités et de découpler les traitements des applications dans lesquels ils sont contenus.

Nous avons identifié deux types de services :

1. Les services servant d'accès à des données qui sont la plupart de temps lues mais rarement mises à jour;

2. les services qui sont très sollicités par les utilisateurs en apportant des changements fréquents à des données relatives à la maintenance des équipements.

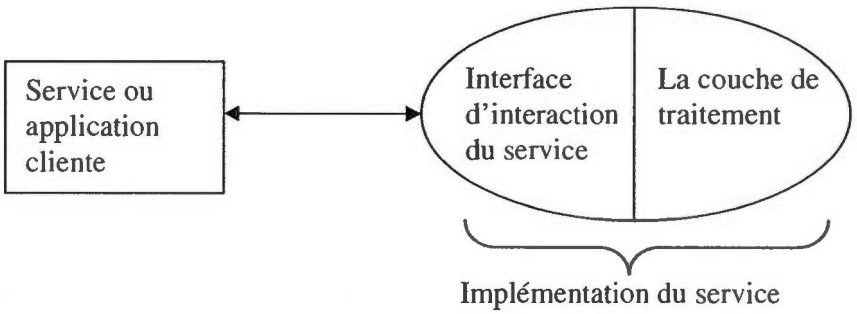

Figure 5.4 Les couches d'un service web

L'implémentation de services web peut se faire de deux façons différentes :

- **Approche descendante :** Elle consiste à définir le fichier WSDL qui représente le contrat entre le service et ses clients. C'est cette approche qui est recommandée car il permet d'assurer la bonne évolution du service. Tout changement qui peut survenir dans le service n'affecte en rien les clients tant que le contrat (l'interface WSDL) n'est pas changé;

- **Approche ascendante :** Dans cette approche, on part de la classe qui représente l'interface du service pour générer l'interface WSDL. L'avantage est qu'il est plus simple puisqu'il n'est pas nécessaire de connaitre les détails liés au WSDL. Des outils fournis dans les environnements de développement permettent de générer automatiquement les interfaces WSDL. L'inconvénient de cette approche est de ne pas pouvoir assurer la maintenance de services facilement puisqu'on est obligé de faire des ajustements au niveau des clients qui utilisent le service

Dans le cas des services de remise en charge, nous avons adopté l'approche ascendante parce que nous sommes partis d'applications existantes que nous avons transformées en services web. En fait, nous n'avons pas le choix car la partie « traitement » des services existe déjà. Nous avons donc utilisé l'outil Java-to-WSDL pour générer les interfaces WSDL.

Il faut noter que certaines considérations sont importantes lors de la définition de l'interface WSDL à partir du code Java. Notamment la surcharge des méthodes qui consiste à utiliser le même nom pour deux ou plusieurs méthodes en variant le nombre ou les types des paramètres. Les outils de génération ne font pas de distinction entre ces méthodes et risquent de causer des problèmes lors de déploiement des services. Il est donc conseillé de ne pas utiliser le même nom pour différentes méthodes d'un même service web.

Les services de remise en charge que nous avons implémentés sont en nombre de 4 :

1. **Le service de disponibilité des équipements :** Il retourne la liste des équipements disponibles pour une station ou tout le réseau. L'opération de ce service `getAllAvEquipments(String networkName)` donne la liste des équipements disponibles pour le réseau spécifié en paramètre.

2. **Le service de connectivité des équipements;** Il retourne pour chaque équipement la liste des équipements auxquels il est connecté. L'opération

getAllCnEquipments(String networkName) donne la liste des équipements pour le réseau spécifié en paramètre

3. **Le Service des règles de gestion de la remise en charge :** Ce service donne l'ensemble des règles de gestion qui décrivent la connaissance liée à la remise en charge d'un réseau. Ces règles sont maintenues par un expert de la remise en charge. L'opération getAllKwNetwork(String networkName) donne la connaissance liée à la remise en charge du réseau spécifié en paramètre.

4. **Le service des schémas des stations :** Il permet d'obtenir la configuration des équipements dans une plusieurs stations sous forme graphique. Ces schémas servent à montrer la progression de la remise en charge. L'opération getAllScNetwork(String networkName) retourne les schémas de toutes les stations du réseau.

5. **Le service de génération de plans :** Il permet de générer un plan pour un réseau ou une station. On dispose de l'operation getNetworkPlan(String networkName, List connectivity, List schema, List knowlege, List Availibility).

Le schéma de la figure 5.5 montre les différents services et leur interaction

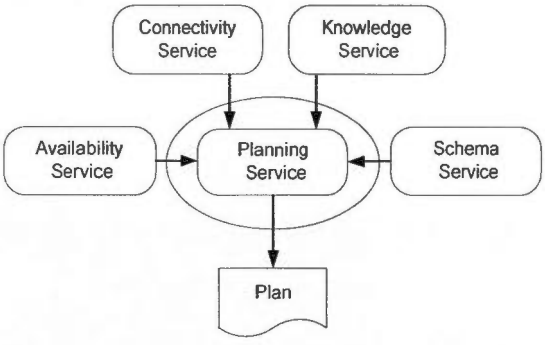

Figure 5.5 Les services de remise en charge

5.4 Le descripteur WSDL du service de connectivité

Le service de connectivité permet d'avoir pour chaque équipement l'ensemble des équipements voisins auxquels il est connecté. Nous donnons ci-après un extrait du descripteur WSDL montrant l'opération qui permet de récupérer la liste des équipements voisins d'un équipement donné. Pour le descripteur détaillé veuillez consulter l'annexe B1.

```
<wsdl:operation name="getConnectivite">

<wsdl:input message="impl:getConnectiviteRequest"
name="getConnectiviteRequest" />

<wsdl:output message="impl:getConnectiviteResponse"
name="getConnectiviteResponse" />

 </wsdl:operation>
```

5.5 Le descripteur WSDL du service de disponibilité

Le service de disponibilité d'équipements permet d'obtenir la liste des équipements non disponibles pour une remise en charge du réseau. La description WSDL de l'opération d'accès à l'état d'un équipement (disponible ou non) est donné ci-après. La description detaillée du service est donnée dans l'annexe B2.

```
<wsdl:operation name="getEtatEqp">

<wsdl:input message="impl:getEtatEqpRequest" name="getEtatEqpRequest" />

<wsdl:output message="impl:getEtatEqpResponse" name="getEtatEqpResponse" />

</wsdl:operation>
```

5.6 Les règles des services de remise en charge

Dans cette section, nous allons donner la liste des règles qui modélisent la sémantique de composition de services de rétablissement du réseau de distribution d'énergie.

Le service de disponibilité des équipements

```
(defrule  AvailibilityService.AvailibilityPort.getAllAvEquipments

(Network ?X)
```

(Known name ?X)

=>

(assert (Known Availability ?X)))

Le service de connectivité des équipements

(defrule ConnectivityService.ConnectivityPort.getAllCnEquipments

(Network ?X)

(Known name ?X)

=>

(assert (Known Connectivity ?X)))

Le service de gestion des règles de remise en charge:

(defrule KnowledgeService.KnowledgePort.getAllKwEquipments

(Network ?X)

(Known name ?X)

=>

(assert (Known Knowledge ?X)))

Le service des schémas des stations

(defrule SchemaService.SchemaPort.getAllScNetwork

(Network ?X)

(Known name ?X)

=>

(assert (Known Schema ?X)))

Le service de génération de plans des réseaux

(defrule PlanningService.PlanningPort.getNetworkPlan

(Network ?X)

(Known name ?X)

(Known Schema ?X)

(Known Knowledge ?X)

(Known Connectivity ?X)

(Known Availability ?X)

=>

(assert (Known Plan ?X))

(assert (Colored Schema ?X)))

Le service de génération de plans des stations

(defrule PlanningService.PlanningPort.getStationPlan

(Network ?X)

(Known name ?X)

(Station ?Y)

(Known name ?Y)

(Contains ?X ?Y)

(Known Plan ?X)

=>

(assert (Known Plan ?Y))

(assert (Colored Schema ?Y)))

L'ensemble des règles précédentes utilisent les entités suivantes:

- « Network » est l'entité qui contient une liste de stations
- « Station » est l'entité qui possède un ensemble d'équipements
- « Schema » est l'entité qui contient les coordonnées de chaque équipements. Ces coordonnées servent à positionner les équipements sur le graphe d'une station
- « Knowledge » est l'entité qui contient l'ensemble des règles de gestion de la remise en charge d'un réseau
- « Availability » est l'entité qui contient une liste d'équipements indisponibles.
- « Connectivity » est l;entité qui contient une liste d'équipements avec leurs voisins respectifs. Elle sert à propager le rétablissements des équipements d'une station à l'autre

L'ensemble des entités précédentes utilisent les différents objets représentant les types d'équipements. La figure 5.6 donne les différents types d'équipements

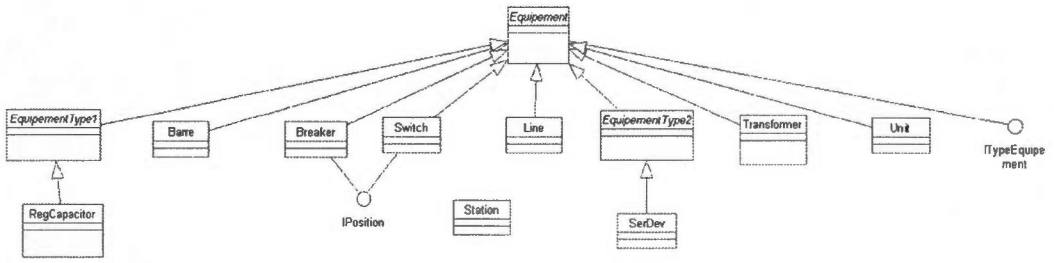

Figure 5.6 les différents objets d'équipements constituant les entités des règles des services web de remise en charge

5.7 Exécution du prototype

Dans la section précédente, nous avons présenté les règles représentant les services web de remise en charge. Dans cette section, nous présentons le résultat d'exécution de ces règles.

Pour générer le plan de composition du service composé qui consiste à rétablir le réseau, il faut préciser les entrées et les sorties de ce dernier. Les entrées de ce service sont insérées dans la base de faits comme faits initiaux. Les deux faits qui constituent cette entrée : (Network A) (Known name A))

Pour un réseau A dont on donne le nom, on veut trouver les instructions qui permettent de rétablir le réseau à un moment donné. Le résultat est le graphe correspondant au plan de composition (Figure 5.7)

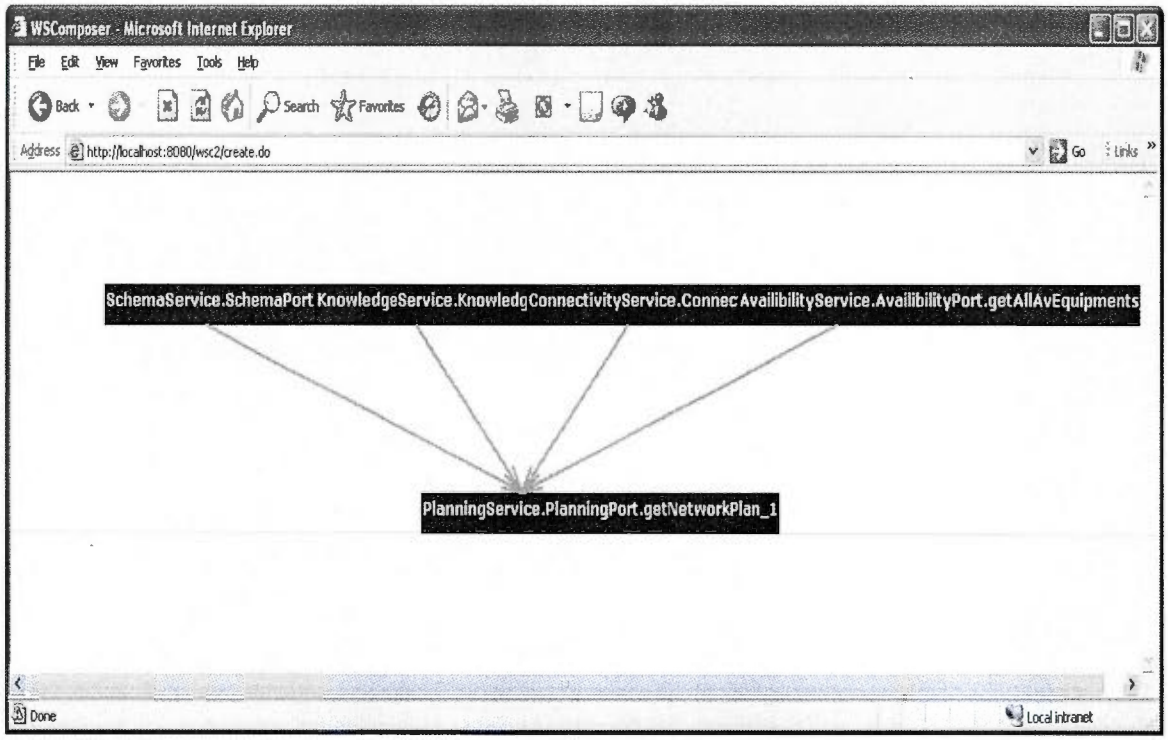

Figure 5.7 Le graphe de composition des services de remise en charge du réseau

Le service de recherche d'itinéraire entre 2 domiciles de 2 personnes (donné au chapitre 4) est représenté par le graphe de la figure 5.8

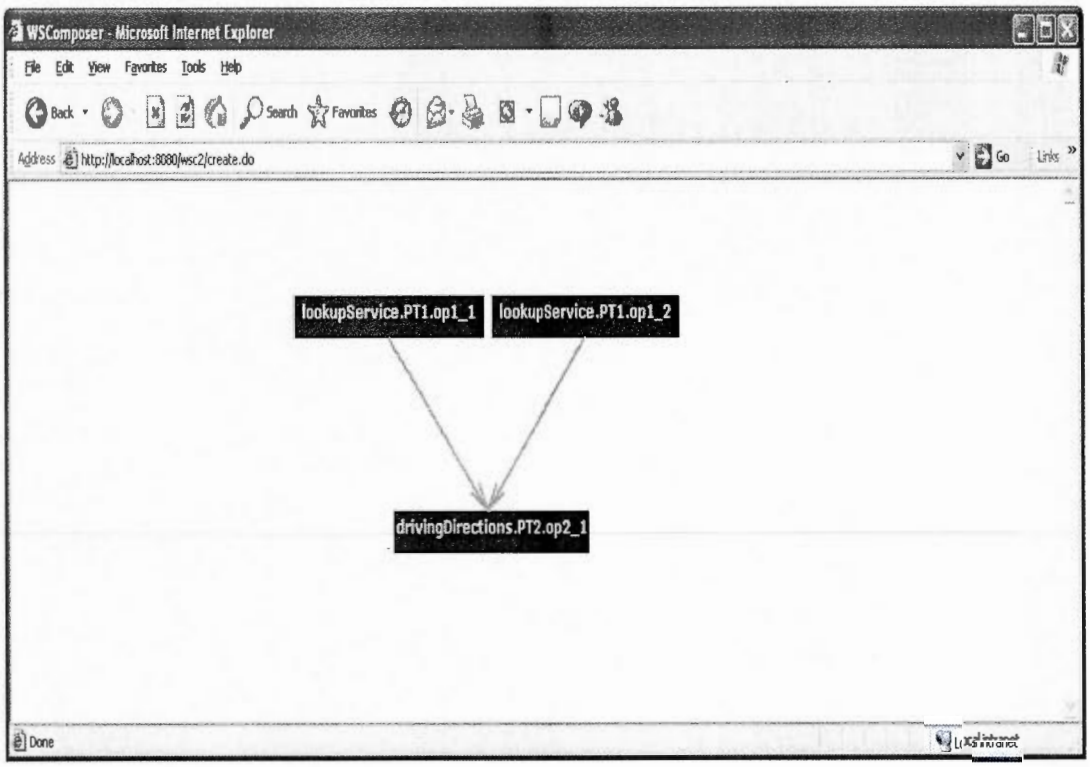

Figure 5.8 Le graphe de composition des services de recherche d'itinéraire

5.8 Conclusion

L'élaboration de l'exemple de remise en charge du réseau nous a permis de faire la preuve du concept de l'approche de composition proposée dans le travail de ce mémoire et de valider le fonctionnement du prototype.

La génération des graphes de composition est une étape importante du processus de composition de services. Nous avons montré par la réalisation de ce prototype qu'il est possible de réaliser un système de composition utilisant un système à base de connaissances et les standards des services web industriels.

CHAPITRE VI

CONCLUSION

L'objectif principal du travail présenté dans ce mémoire est le développement d'un outil de composition. La solution que nous avons élaborée met en œuvre les standards des services web et les mécanismes des systèmes à base de règles. L'utilisation de WSDL comme outil de description de services et son enrichissement par le modèle de règles a permis de remédier à plusieurs problèmes relatifs à la composition dynamique de services.

L'approche mixte regroupe plusieurs concepts qui ont été implémentés grâce à l'utilisation de JESS. Ce dernier a facilité l'exploitation de la structure d'exécution des règles pour l'extraction des informations de génération de plans de composition. Cette structure est utilisée aussi pour la génération du code BPEL en faisant une transposition des règles exécutées en termes d'invocations d'opérations des services impliqués lors d'une requête de composition d'un service.

Le prototype réalisé constitue une preuve de concept pour l'écriture d'un environnement de composition pour des applications du domaine industriel. Nous avons montré qu'il est possible de lier la description syntaxique d'un service en WSDL à la description sémantique par les règles. Nous nous sommes forcés de trouver cette liaison vu la complétude des ces deux derniers aspects.

L'analyse que nous avons effectuée sur les solutions existantes pour la composition de services nous a montré les aspects les plus importants de la problématique de composition. Ce qui nous permis de faire des choix appropriés pour atteindre l'objectif principal que

voulions atteindre dans le cadre de ce travail à savoir une approche de composition pragmatique.

Notre solution est caractérisée par l'utilisation de l'infrastructure de services web aussi bien au niveau des standards utilisés qu'au niveau des outils de développement. L'apport de JESS pour la modélisation de la composition par des règles est incontestable. Son utilisation offre une flexibilité accrue pour le concepteur des connaissances de la composition de services

Plusieurs idées de l'approche mixte peuvent être étendues pour consolider le modèle de composition :

- L'utilisation d'un modèle d'invocation dynamique permettra de découpler les applications clientes des services. Ce modèle peut être implémenté sous-forme d'un médiateur entre les clients et les services de telle sorte que les changements apportés aux services n'entraînent pas des modifications sur les clients.

- L'utilisation de la logique floue au niveau des règles représentant les opérations des services peut améliorer la qualité des plans de composition générés. Ainsi, le choix des règles à exécuter sera fait sur des critères qualitatifs relatifs au domaine traité ou à la fiabilité des services modélisés.

- L'utilisation d'un annuaire offrant un système de classification de services permettant de retrouver ces derniers plus facilement aidera les utilisateurs ou les développeurs dans leurs recherches. UDDI, à notre sens, n'est pas suffisant car il ne prend pas complètement en compte l'organisation sémantique des services. Les ontologies peuvent être une bonne piste à explorer.

- Le modèle de règles pour la représentation des opérations peut être enrichi afin de prendre en charge le traitement des exceptions et la gestion des services transactionnels. Ceci peut par exemple se faire par l'association aux opérations d'une règle traitant les exceptions et/ou les transactions.

APPENDICE A

LES MODÈLES DES ACTIVITES DE BPEL

A.1 L'activité de réponse à une requête

Les variables d'états	rep, messageSoap, etatVar = {debut_reply,fin_reply}
Action ou événement	Reply
Les règles de transition	(etatVar = debut_reply ET exist(rep)) → (messageSoap = rep ET etatVar = fin_reply)*

- Exist est un prédicat qui vérifie que la réponse est initialisée pour être retournée comme résultat.

A.2 L'activité d'affectation d'une variable « assign »

Elle affecte de nouvelles données aux variables définies dans le processus pendant l'exécution de ce dernier.

Les variables d'états	inVar,outVar, etatVar = debut_assign, fin_assign}
Action ou événement	assign
Les règles de transition	(etatVar = debut_assign ET exist(inVar)) → (etatVar = fin_asign ET outVar = inVar)

	(messageSoap = rep ET etatVar = fin_reply)*

A.3 L'activité de génération d'exception « throw »

Elle permet de capter les cas d'exception ou d'erreurs afin d'entreprendre des actions de recouvrement.

Les variables d'états	fault.mode appartient à {on,off}, etatVar = {debut_throw, fin_throw}
Action ou événement	Throw (événement déclenchant l'exception)
Les règles de transition	(etatVar = debut_throw ET fault.mode = off) → (etatVar = fin_throw ET fault.mode = on)

A.4 L'activité de suspension de l'exécution « wait »

Elle suspend l'exécution du processus pour une certaine période de temps ou jusqu'à ce que un temps donné (date, heure) est atteint.

Les variables d'états	etatVar = {debut_wait, fin_wait}
Les variables internes	wait_mode appartient {on,off}
Action ou événement	wait, fin_wait
Les règles de transition	{etatVar = debut_wait ET wait_mode = off) → (wait_mode = on) (wait_mode = on) → (etatVar = fin_wait ET wait_mode = off)

Il faut noter que le temps n'est pas pris en considération dans ce modèle

A.5 L'activité de synchronisation « empty »

Les variables d'états	etatVar = {debut_empty, fin_empty}
Action ou événement	empty
Les règles de transition	(etatVar = debut_empty) → (etatVar = fin_empty)

A.6 L'activité Séquence « sequence »

Une activité « sequence » peut comporter n activités de base dans sa portée qui vont être exécutées dans un ordre séquentiel si les conditions d'exécutions sont vérifiées. Si on définit l'ensemble des activités imbriquées par {Ai}, on a alors le modèle suivant

Les variables d'états	etatVar = {debut_sequence,fin_sequence, debutAi,finAi, et i appartenant à {1,…,n} }
Les transitions	Debut_sequence → debutA1 etatA1 → finA1 finA1 → debutA2 … finAi →debutAi+1 … finAn → fin_sequence
Action ou événement	appel(Ai), fin, Ai.event avec i dans {1,..,n}
Les règles de transition	(etatVar = debut_sequence) → (etatVar = debutA1) (etatVar = finAi) → (etatVar = debutAi+1) (etatVar = finAn) → (etatVar = fin_sequence)

A.7 L'activité switch

Cette activité choisit une branche parmi les n activités {A1,...,An} et une autre branche pour le cas otherwise correspond à l'activité An+1. Une activité Ai transforme l'état etatAi en l'état finAi

Les variables d'états	V1,..,Vn sont les domaines des variables associés au n branches etatVar= {debut_switch,fin_switch,debutAi,finAi} avec 1<i<n}
Les transitions	Debut_switch → debutAi debutAi → finAi finAi → finSwitch
Action ou événement	switchAi, finSAi, Ai.nonEvent avec 1<i<n
Les règles de transition	(etatVar = debut_switch ET Non Pre(V1) ET ...Non Pre(Vi)) → etatVar = debutAi) (etatVar = debut_switch ET Non Pre(V1) ET ...Non Pre(Vn)) → etatVar = debutAn+1) (etatVar = debut_switch ET Non Pre(V1) ET ...Non Pre(Vi)) → etatVar = debutAi)

A.8 L'activité flow

Cette activité exécute toutes les activités de base se trouvant dans sa porté. On désigne par {A1,..,An} l'ensemble de ces activités. Chaque activité Ai possède une variable d'entrée et une variable de sortie inVari et outVari. Son modele est donné dans le tableau suivant :

Les variables d'états	{inVari, outVari} pour l'activité {Ai}, etatVar = {debut_flow, fin_flow}

Les variables internes	{interneEtatVari = {debutAi, finAi}, i dans {1,..,n}
Action ou événement	debutFlow,Ai.nomEvent, finFlow, i dans {1,..,n}
Les transitions	debut_flow → debutAi debutAi → finAi finAi → fin_flow
Les règles de transition	(EtatVar = debut_flow)→ (ET interneEtatVari = debutAi) (pre(inVari) ET interneEtatVari = debutAi) →(interneEtatVari = finAi ET post(outVari)) (ET interneEtatVari = finAi) → (etatVar = fin_flow)

A.9 L'activité pick

Cette activité permet de bloquer l'exécution du processus jusqu'à ce qu'un message arrive ou une alarme de dépassement de temps maximal est signalée. On est en présence d'un cas d'indéterminisme car on ne sait pas à l'avance quelle est la prochaine branche à suivre. Les activités {A1,A2,...,An} correspondent à ces branches. Son modèle est donné dans le tableau suivant :

Les variables d'états	V1,V2,...Vn sont les variables utilisées par les n branches, etatVar = {debut_pick,fin_pick, debutAi, finAi avec i dans {1,..,n}}
Les variables internes	{interneEtatVari = {debutAi, finAi}, i dans {1,..,n}
Action ou événement	debutPick, , finPick, Ai.nomEvent, avec i dans {1,...,n}
Les transitions	debut_pick → debutAi debutAi → finAi finAi → fin_pick

Les règles de transition	(etatVar = debut_pick ET exist(Vi) → (etatVar = debutAi) (etatVar = finAi) → (etatVar = fin_pick)

APPENDICE B

B.10 Le descripteur WSDL du service de connectivité

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
targetNamespace="urn:recre" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="urn:recre" xmlns:intf="urn:recre"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="urn:recre.BeanService" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <wsdl:types>
      <schema targetNamespace="urn:recre"
xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://xml.apache.org/xml-soap" />
      <import namespace="urn:recre.BeanService" />
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="ArrayOf_soapenc_string">
      <complexContent>
      <restriction base="soapenc:Array">
       <attribute ref="soapenc:arrayType" wsdl:arrayType="soapenc:string[]" />
       </restriction>
  </complexContent>
  </complexType>
<complexType name="ArrayOf_tns1_EquipementBean">
```

```
<complexContent>
<restriction base="soapenc:Array">
<attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:EquipementBean[]" />
 </restriction>
 </complexContent>
 </complexType>
 </schema>
<schema targetNamespace="http://xml.apache.org/xml-soap"
xmlns="http://www.w3.org/2001/XMLSchema">
 <import namespace="urn:recre" />
 <import namespace="urn:recre.BeanService" />
 <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="Vector">
<sequence>
<element maxOccurs="unbounded" minOccurs="0" name="item"
type="xsd:anyType" />
 </sequence>
 </complexType>
 </schema>
<schema targetNamespace="urn:recre.BeanService"
xmlns="http://www.w3.org/2001/XMLSchema">
 <import namespace="urn:recre" />
 <import namespace="http://xml.apache.org/xml-soap" />
 <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="EquipementBean">
      <sequence>
```

```
<element name="MName" nillable="true" type="soapenc:string" />
<element name="MStation" nillable="true" type="apachesoap:Vector" />
<element name="MType" nillable="true" type="soapenc:string" />
<element name="MVoisins" nillable="true" type="apachesoap:Vector" />
</sequence>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="mainResponse" />
<wsdl:message name="getConnectiviteResponse">
<wsdl:part name="getConnectiviteReturn"
type="impl:ArrayOf_tns1_EquipementBean" />
</wsdl:message>
<wsdl:message name="mainRequest">
<wsdl:part name="in0" type="impl:ArrayOf_soapenc_string" />
</wsdl:message>
<wsdl:message name="getConnectiviteRequest" />
<wsdl:portType name="Chargement">
<wsdl:operation name="main" parameterOrder="in0">
<wsdl:input message="impl:mainRequest" name="mainRequest" />
```

```
<wsdl:output message="impl:mainResponse" name="mainResponse" />

</wsdl:operation>

<wsdl:operation name="getConnectivite">

<wsdl:input message="impl:getConnectiviteRequest"
name="getConnectiviteRequest" />

<wsdl:output message="impl:getConnectiviteResponse"
name="getConnectiviteResponse" />

</wsdl:operation>

</wsdl:portType>

<wsdl:binding name="EquipementServiceSoapBinding"
type="impl:Chargement">

<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />

<wsdl:operation name="main">

<wsdlsoap:operation soapAction="" />

<wsdl:input name="mainRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://equipement" use="encoded" />

</wsdl:input>

<wsdl:output name="mainResponse">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:recre" use="encoded" />
```

```
</wsdl:output>

</wsdl:operation>

<wsdl:operation name="getConnectivite">

<wsdlsoap:operation soapAction="" />

<wsdl:input name="getConnectiviteRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://equipement" use="encoded" />

</wsdl:input>

<wsdl:output name="getConnectiviteResponse">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:recre" use="encoded" />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name="EquipementService">

<wsdl:port binding="impl:EquipementServiceSoapBinding"
name="EquipementService">

<wsdlsoap:address
location="http://localhost:8080/axis/services/EquipementService"/>

</wsdl:port>

</wsdl:service>
```

```
</wsdl:definitions>
```

B.2 Le descripteur WSDL du service de disponibilité

```
<?xml version="1.0" encoding="UTF8" ?>
<wsdl:definitions targetNamespace="urn:wsc.etat"
xmlns:apachesoap="http://xml.apache.org/xmlsoap" xmlns:impl="urn:wsc.etat"
xmlns:intf="urn:wsc.etat"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://etat" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>

 <schema targetNamespace="urn:wsc.etat"
xmlns="http://www.w3.org/2001/XMLSchema">

        <import namespace="http://etat" />

        <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />

       <complexType name="ArrayOf_soapenc_string">

       <complexContent>

       <restriction base="soapenc:Array">

        <attribute ref="soapenc:arrayType" wsdl:arrayType="soapenc:string[]" />

        </restriction>

        </complexContent>

        </complexType>
```

```
<complexType name="ArrayOf_tns1_EtatBean">
<complexContent>
<restriction base="soapenc:Array">
 <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:EtatBean[]" />
 </restriction>
 </complexContent>
 </complexType>
 </schema>
 </wsdl:types>
 <wsdl:message name="getEtatEqpRequest" />
 <wsdl:message name="mainResponse" />
<wsdl:message name="getEtatEqpResponse">
 <wsdl:part name="getEtatEqpReturn" type="impl:ArrayOf_tns1_EtatBean" />
 </wsdl:message>
<wsdl:message name="mainRequest">
 <wsdl:part name="in0" type="impl:ArrayOf_soapenc_string" />
 </wsdl:message>
<wsdl:portType name="ChargerEtat">
<wsdl:operation name="main" parameterOrder="in0">
```

```
<wsdl:input message="impl:mainRequest" name="mainRequest" />
<wsdl:output message="impl:mainResponse" name="mainResponse" />
</wsdl:operation>
<wsdl:operation name="getEtatEqp">
<wsdl:input message="impl:getEtatEqpRequest" name="getEtatEqpRequest" />
<wsdl:output message="impl:getEtatEqpResponse" name="getEtatEqpResponse" />
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="EtatServiceSoapBinding" type="impl:ChargerEtat">
<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="main">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="mainRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://etat" use="encoded" />
</wsdl:input>
<wsdl:output name="mainResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:wsc.etat" use="encoded" />
```

```
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getEtatEqp">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getEtatEqpRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://etat" use="encoded" />
</wsdl:input>
<wsdl:output name="getEtatEqpResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:wsc.etat" use="encoded" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="EtatService">
<wsdl:port binding="impl:EtatServiceSoapBinding" name="EtatService">
<wsdlsoap:address location="http://localhost:8080/axis/services/EtatService" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Bibliographie

[1] Web Service Composition - Current Solutions and Open Problems. Biplav Srivastava Jana Koehler 2004, http://www.zurich.ibm.com/pdf/ebizz/icaps-ws.pdf

[2] Service-Oriented Composition in BPEL4WS Rania Khalaf,,Nirmal Mukhi,Sanjiva Weerawarana 2003, http://www.unizh.ch/home/mazzo/reports/www2003conf/papers/P768/p768-khalaf.pdf

[3] web services orchestration Chris Peltz Hewlett Packard, Co. January 2003

[4] A Developer Toolkit for Web Service Composition. Shankar R. Ponnekanti and Armando Fox 2002, http://www2002.org/CDROM/alternate/786/

[5] Java Expert System Shell (Jess). http://herzberg.ca.sandia.gov/jess/.

[6] W3C. WSDL specification. http://www.w3.org/TR/WSDL/

[7] web services Axis2. http://ws.apache.org/axis2/

[8] A survey of middleware, discovery & composition technologies to aid in goal of distributed service composition in a pervasive environment. JON ROBINSON, 2006, http://www.cse.unsw.edu.au/~jyu/ieee-ic/UI_Integration.pdf

[9] Semi-automatic Composition of Web Services using Semantic Descriptions, 2004, http://www.mindswap.org/papers/composition.pdf

[10] Business process execution language for web services. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

[11] web services orchestration. Chris Peltz Hewlett Packard January 2003

[12] W3C. SOAP 1.2 specification. http://www.w3.org/TR/soap/

[13] OASIS. UDDI 3.0.2. specification http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[14] Adaptative and Dynamic Service Composition in eFlow. Fabio casati, Ski Ilnicki, LiJie Jin. HP march, 2000

[15] web services orchestration. Chris Peltz. HP, Co. January 2003

[16] BPEL4WS specification. ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf

[17] web service composition. Sateya Sanket Sahoo. University of Georgia 2004

[18] Interopérabilité des services web complexes. Tarek Melliti 2004, http://www.lamsade.dauphine.fr/~melliti/TheseTarak.pdf

[19] Adapting Golog for semantique web services. S. Macllraith, T.Cao Sun

[29] Software Engineering, Ian Sommerville 7th edition 2005

[30] W3C. RDF specification. http://www.w3.org/RDF/

[31] BPEL : Service composition for SOA. Matjaz B. Juric 2006

[32] describe business process activities as web services. http://www.javaworld.com/javaworld/jw-10-2005/jw-1031-webservices-p2.html. Ash parikh, vivek kondur 2005.

[33] Plængine: A System for Automated Service Composition and Process Enactment. Harald Meyer, Hagen Overdick, andMathiasWeske, Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany

[34] BPES4J http://www.alphaworks.ibm.com/tech/bpws4j

[35] Future e-business solutions : Top Down or Bottom Up? Technological vs Business Driven Approach to e-Business. Jyrki Haajanen 2004

[36] BPEL: Service composition for SOA. Matjaz B. Juric 2006

[37] Integrating dynamic resources in corporate semantic web: an approach to enterprise application integration using semantic web services. Moussa LO, Fabien Gamdon 2005

[38] Developing Real World Web Services-based Applications. Sandeep Chatterjee

[39] Web Service Orchestration with BPEL. Christoph Schittko

[40] Tools for Design of Composite Web Services. Richard Hull, Jianwen Su 2006

[41] Web Services invocation sans SOAP. Nirmal Mukhi 2001

[42] What is Wrong With Web services discovery. Dieter Fensel, Uwe Keller 2005

[43] Web Services Modeling Framework WSMF. D.Fensel, C.Bussler

[44] Business Process Management. Tanguy Crusson 2003

[45] UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. Tracy Gardner

[46] Monitoring Web Services Networks in a Model-based Approach. Yan,Y.,Pencole, Y., Grastien, A 2005

[47] J2EE Web Services. Richard Monson-Hefel 2006

[48] Software Engineering 6th edition. IAN Sommerville 2001

[49] What is Web Service Architecture. Hao He 2003

[50] Implementing Service Oriented Architecture with Apache Axis2 and Web Services. Paul Fremantle 2006

[51] The Java Web Services Tutorial. Sun Microsystems 2005

[52] A planner for composing services described in DAML-S. In Workshop on Web services And Agent-based engineering, M. Sheshagiri, M. desJardins, and T. Finin. 2003.

[53] Artificial Intelligence & Software Engineering.Derek Parteridge 1991

[54] Making BPEL Processes Dynamic. Sean Carey.

http://www.oracle.com/technology/pub/articles/bpel_cookbook/carey.html

[55] Web Services Composition (An AI-based Semantic Approach). Satya Sanket Sahoo 2004

[56] Integration with Web Services. Steve Vinoski. 2003

[57] Sélection et Optimisation pour la Composition de Services Web. Daniela Barreiro Daniela Barreiro Claro. 2006

[58] A Framework for Architecture-driven Service Discovery. A. Kozlenkov V. Fasoulas F. Sanchez G. Spanoudakis A. Zisman

[59] Service Composition with Semantic Web Service.Mark Burstein, Christoph Bussler. Proceeding 2005