République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane MIRA de Béjaïa

Faculté des Sciences Exactes
Département d'Informatique
École Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère En Informatique

Option : **Réseaux et Systèmes Distribués**

Thème

Vers une approche CSP pour la compatibilité des politiques dans les services Web.

Présenté par : MIR Foudil

Devant le jury composé de :

- Président	**: KHELFAOUI**	**Youcef**	**M.C**	**Université A. M Bejaia. Algérie.**
- Rapporteur	**: BENBERNOU**	**Salima**	**M.C**	**Université C. B Lyon-1. France.**
- Examinateurs	**: BOUKERRAM**	**Abdallah**	**M.C**	**Université F. A Sétif. Algérie.**
	MOUSSAOUI	**Abdelouahab**	**M.C**	**Université F. A Sétif. Algérie.**
- Invité	**: AMROUN**	**Kamal**	**M.A**	**Université A. M Bejaia. Algérie.**

Promotion 2005-2006

Résume

L'évolution d'Internet et la compétitivité entre les entreprises ont été les facteurs de l'explosion des services Web. Les services Web sont des applications accessibles sur Internet réalisant chacune une tâche spécifique. Pour répondre a des besoins complexe, nous pouvons regrouper des services web pour n'en former qu'un seul ; nous parlons alors de composition de services Web. La grande croissance du nombre de services Web disponibles, l'environnement dynamique, l'évolution des besoins des utilisateurs et l'existence d'un grand nombre de services qui fournissent les mêmes fonctionnalités, augmentent la complexité de rechercher un service approprié qui peut réaliser une tache bien spécifiée. Une telle composition ne devrait pas seulement choisir le service disponible le plus approprié, mais devrait également respecter et adhérer aux politiques des services composants, des clients et du service Web composite. Dans notre approche nous nous sommes concentrés sur le processus de sélection d'un flux de services Web compatibles en terme de politiques, en exploitant une approche CSP (*Constrait Satisfaction Problem*) pour formaliser et résoudre ce problème. Cette formalisation CSP du problème du choix des services Web compatibles, se définit comme un ensemble de contraintes et de relations impliquant un ensemble de variables. L'objectif est de trouver une valeur pour chaque variable afin de satisfaire l'ensemble des contraintes. Notre travail introduit une nouvelle formalisation du problème de la sélection de l'ensemble des services Web compatibles à composer, en exploitant les avantages du formalisme CSP.

Mots clés : Service Web, Composition horizontale, politique de services Web, langages de politiques, Compatibilité de politique, CSP.

Abstract

The evolution of Internet and competitiveness between the companies were the main factors of the explosion of the Web services. The Web services are accessible applications on Internet carrying out each one a specific task. To provide a solution to a complex task, one can gather Web services to form a unique one; we speak so about composition of Web services. The great growth of the number of Web services available, the dynamic environment, evolution of the user's needs, and the existence of a great number of services which provide the same functionalities, increases complexity to seek a suitable service which can carry out a well specified spot. Such a composition should not only choose the more adapted service available, but should also respect and adhere to the policies component services, customers and composite Web service. In our approach we focused on the selection process of a compatible Web services flow in term of policies by exploiting the CSP approach (Constrait Satisfaction Problem) to formalize and solve this problem. This formalization CSP is defined as a set of constraints and relations implying a set of variables. The objective is to find a value for each variable in order to satisfy the set of the constraints. Our work introduces a new formalization of the selection problem of the compatible Web services set, by exploiting the CSP formalism advantages.

Key words: Web service, horizontal composition, Web services policies, policy languages, policy Compatibility, CSP.

Dédicaces

À mes très chers parents,
À mes frères et soeurs,
À toutes ma famille,
À mes amis et collègues.

Remerciements

Je remercie Dieu de m'avoir donné la santé, la patience et la détermination ; sans les quelles je n'aurais jamais pu porter mon projet à terme.

Si ce mémoire a pu voir le jour, c'est certainement grâce au soutien et à l'aide de plusieurs personnes. Je profite de cet espace pour les remercier tous.

Je tiens à remercier avant tout mon encadreur BENBERNOU Salima MC F à l'*université lyon-1*, pour m'avoir proposé ce sujet très intéressant, pour ses conseils et ses orientations.

Mes très vifs remerciements vont à Mr TARI Kamel, le responsable de l'école doctorale ReSyD et chef de département d'informatique de l'université de A/ MIRA de Béjaia, pour les efforts qu'il a fournit pour notre formation. Merci pour son aide très précieuse, ses conseils et sa disponibilité.

Je remercie les membres de Jury qui ont accepté de juger mon travail. J'adresse mes très sincères remerciements à : Mr Y. Khelfaoui maître de conférences à l'université A.MIRA, Béjaïa, Mr A. Boukerram, maître de conférences à l'université F/abbas de Setif et Mr A. Moussaoui, maître de conférences à l'université F/abbas de Setif, ainsi que Mr K. Amroun maître assistant à l'université A.MIRA, Béjaïa.

Je remercie mes amis et collègues de formation Magistère (ReSyD 3) ; en particulier, A. Sebaa, A. Arab, S. Haddad et S. Kedjar et surtout M. Ait Amokhtar pour son aide et ses remarques pertinentes, sans oublier S. Khanouche de ReSyD 5.

Je remercie vivement tous mes amis (es) cursus de graduation et celui de la poste graduation, en particulier Madjid, Sofiane et Kaci.

Je ne pourrais oublier de remercier profondément les êtres qui me sont les plus chers, qui ont eu un rôle primordial dans ma réussite, et qui sans eux aucune réussite n'aurait été possible : mon père et ma mère. Un grand merci à mes sœurs et frères ; surtout Mouhand, Okba et Douda, pour leur encouragement, leur aide et leur amour depuis ma tendre enfance.

ReSyD, Université de A/MIRA Béjaia. *Foudil MIR*
Mai, 2008

Table des matières

Chapitre 3 Les langages de politiques dans les services Web

Chapitre 4 La compatibilité de politiques

Liste des figures

Liste des algorithmes

Introduction générale

Dans le domaine du management d'entreprise et des systèmes d'information, un « buzzword » (mot à la mode) fait référence à un néologisme - et au nouveau concept associé - ayant la capacité à fédérer tout un écosystème : une fois créé, il est utilisé abondamment, et chacun cherche à se l'approprier.

« Services Web », ou « Web Services », succède à « e-business » dans le palmarès des « buzzword » les plus utilisés dans le monde des systèmes d'information. Et il y a fort à parier que d'ici quelques années, l'utilisation de ce terme aura disparu tant le concept qu'il sous-entend est appelé à se généraliser et à devenir le modèle de référence pour les systèmes d'information.

Ce passage extrait de l'introduction de *« Les Services Web, pourquoi ? Dix questions essentielles »*, une étude publiée en 2003 par *BearingPoint*, *Sun Microsystems* et *SAP*, montre l'importance et l'utilisation grandissante des services Web dans les activités des entreprises. L'intérêt des services Web est de permettre à une entreprise d'exporter, via le réseau Internet, ses compétences et son savoir-faire, ou encore d'ouvrir de nouveaux marchés et de nouveaux supports à la vente. Nous pouvons citer, par exemple, la société *Amazon* qui propose un service Web permettant de rechercher et de commander un article sans passer par son site Internet ; ou encore *Google* qui permet aux développeurs du monde entier d'utiliser le célèbre moteur de recherche directement dans leurs applications. Inversement, ces services Web permettent aux entreprises d'utiliser à moindre frais les compétences et le savoir-faire des autres entreprises.

Mais pour exploiter au maximum les avantages de ce nouveau type d'applications, la communauté des services Web doit se doter d'outils donnant la possibilité de pouvoir assembler les différents services entre eux. Cet assemblage, de la même façon qu'un développeur agence les méthodes de son programme, a pour but d'effectuer des tâches qu'un simple service Web ne saurait résoudre. C'est la composition de services Web. L'approche actuelle de la composition de services Web consiste à définir des processus métier, i.e., des enchaînements réutilisables de services. Cette approche est la plus utilisée dans le monde industriel. Une autre approche envisageable est la composition « dynamique ». Les services Web sont composés dynamiquement en fonction de leurs fonctionnalités, de leur disponibilité et des aléas pouvant survenir au moment du processus de composition.

La composition dynamique (automatique) de service web implique la recherche d'une combinaison de services web existant pour réaliser la tâche globale attendue par l'utilisateur. Cependant, chaque sous tâche peut être réalisé par plusieurs services web ce qui introduit l'importance d'un processus de sélection de service web approprié en tenant compte de plusieurs contraintes ; telle que la compatibilité de politiques (comportement, contrôle d'accès, confidentialité, business, sécurité….) qui jouet un rôle important pour que les services Web seront composable.

Nous assistons de plus en plus, aujourd'hui, à l'utilisation des politiques pour décrire les contraintes d'utilisation d'un service Web; ce qui implique la nécessité de vérification de la compatibilité de politique dés qu'il s'agit d'interagir deux services Web, par exemple dans le cas d'une composition.

Dans le cadre de ce mémoire de magister, Nous allons nous intéresser au processus de sélection d'un flux de services Web compatibles en terme de politiques.

Contribution

Dans le cadre de ce mémoire de magister, nous allons discuter le problème de prise en compte de la compatibilité de politiques dans la composition horizontale de services Web. Nous nous sommes concentrés sur la recherche d'un flux de service Web compatible en utilisant l'approche CSP pour formaliser et résoudre ce problème. Nous avons nommé cette approche la ***CSP4WSC** : Constraint Satitisfaction Problem for Web Services Composition*.

Plan du mémoire

Nous avons organisé notre mémoire en cinq chapitres :

Le premier chapitre est consacré au concept et technologies de services Web. Nous avons introduit la notion des Architectures Orientées Services (SOA), ainsi que les services Web et ses standards.

Dans le deuxième chapitre, nous avons donné quelques définitions et concepts sur la composition de services Web. Ainsi dans cette partie, nous avons présenté les techniques de la composition ainsi que les deux approches de la description d'une composition.

Après ceci, nous présentons dans le chapitre trois l'origine et les motivations de la gestion basée sur la politique, définition et l'association de politiques avec les services Web. Nous terminons ce chapitre par quelques travaux relatifs à l'utilisation des politiques dans la composition de service.

Le chapitre quatre est consacré a la présentation des deux types de langages de description de politiques dans les services, a savoir les langages basés sur l'approche Web

sémantique et les langages basés sur l'approche de combinaisons booléennes de prédicats de politiques. Nous détaillons quelques langages dans les deux approches.

Dans le chapitre cinq, nous proposons une nouvelle approche que nous avons nommée *CSP4WSC : une approche CSP pour la mise en œuvre de la compatibilité de politique dans la composition de services web*. Après avoir présenté l'approche CSP, nous avons montré comment le problème de la composition horizontale peut être formalisé comme un CSP permettant ainsi l'exploitation des algorithmes de résolution des CSP, notamment le *Backtrack* pour la recherche d'un flux de services compatible.

Nous terminons ce mémoire par une conclusion générale ou nous émettons des perspectives de notre travail.

Chapitre 1

Les services Web : Concepts et technologies

Chapitre 1

Les services Web : Concepts et technologies.

1.1 Introduction

A l'origine, le commerce électronique concernait principalement la vente de biens et de services au client (B2C, Business to Consumer), mais rapidement ces échanges se sont étendus aux échanges entre entreprises (B2B, Business to Business). Créés pour faciliter les échanges commerciaux, les services Web prennent leurs racines dans l'informatique distribuée et dans l'avènement du Web. La technologie des services Web a pour objectif d'uniformiser la présentation des services offerts par une entreprise et d'en rendre l'accès transparent pour tout type de plate-forme, a travers un certain nombre de standards d'interopérabilité. Cette notion de services Web désigne essentiellement une application mise a disposition sur Internet par un fournisseur de service, et accessible par les clients a travers des protocoles Internet standards.

Nous présentons, dans un premier temps, les a*rchitectures orientées services*, le concept de service Web puis nous introduisons la notion du Web sémantique et son application aux services Web.

1.2 Architectures orientées services

Un service définit une entité logicielle (*e.g*, ressource, application, module, composant logiciel, etc.) qui communique via un échange de messages et qui expose un contrat

d'utilisation. De façon similaire aux approches par objet ou par composant, l'approche service cherche à fournir un niveau d'abstraction encore supérieur, en encapsulant des fonctionnalités et en permettant la réutilisation de service déjà existant. Nous donnons ici plusieurs définitions, qui ne sont pas *a priori* contradictoires, mais qui se placent selon différents points de vue[1] :

– *A service represents some functionality (application function, business transaction, system service,etc.) exposed as a component for a business process* [1].
– *Service : The means by which the needs of a consumer are brought together with the capabilities of aprovider* [2].
– *A service in SOA is an exposed piece of functionality with three properties : (1) the interface contract to the service is platform-independent, (2) the service can be dynamically located and invoked, (3) the service is self-contained, that is, the service maintains its own state* [3].

Cette approche introduit une propriété très importante ; c'est la notion du couplage faible qui est à l'origine des architectures agiles qui sont les architectures orientées services.

1.2.1 Qu'est qu'une architecture orientée services ?

Une architecture orientée services (SOA pour *Service Oriented Architectures*) est un style d'architecture fondée sur la description de services et de leurs interactions. Les caractéristiques principales d'une architecture orientée services sont : le couplage faible, l'indépendance par rapport aux aspects technologiques et l'extensibilité. La propriété de couplage faible implique qu'un service n'appelle pas directement un autre service, les interactions sont gérées par une fonction d'orchestration. Il est donc plus facile de réutiliser un service puisqu'il n'est pas directement lié à d'autres services. L'indépendance par rapport aux aspects technologiques est obtenue grâce aux contrats d'utilisation associés qui sont indépendants de la plate-forme technique utilisée par le fournisseur du service. Enfin, l'extensibilité est rendue possible par le fait que de nouveaux services peuvent être découverts et invoqués à l'exécution. La notion d'architecture orientée service n'est pas nouvelle, elle est apparue dès le développement des approches client/serveur. Ici encore les définitions sont nombreuses, alors nous retiendrons les suivantes :

[1] Les définitions sont ici données en anglais pour éviter toute *interprétation* lors de la traduction.

-A service-oriented architecture is a style of multitier computing that helps organizations share logic and data among multiple applications and usage modes, donnée en 1996 par le groupe Gartner [4].

-Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations, établie dans le modèle de référence OASIS (Organization for the Advancement of Structured Information Standards) [2].

-SOA enables flexible integration of applications and resources by : (1) representing every application or resource as a service with standardized interface, (2) enabling the service to exchange structured information(messages, documents, "business objects"), and (3) coordinating and mediating between the services to ensure they can be invoked, used and changed effectively [1]**.**

Malgré le manque de spécification officielle pour définir une architecture orientée services, trois rôles clés sont communément identifiés : producteur de services, répertoire de services et consommateur de services. L'interaction entre ces trois rôles est décrite par la *figure 1.1*. Le producteur de services a pour fonction de déployer un service sur un serveur et de générer une description de ce service qui précise à la fois les opérations disponibles et leur mode d'invocation. Cette description est publiée dans un répertoire de services, aussi appelé annuaire. Les consommateurs de services peuvent découvrir les services disponibles et obtenir leurs descriptions en lançant une recherche sur le répertoire. Un consommateur peut alors utiliser la description du service obtenue pour établir une connexion avec le fournisseur et invoquer les opérations du service souhaité.

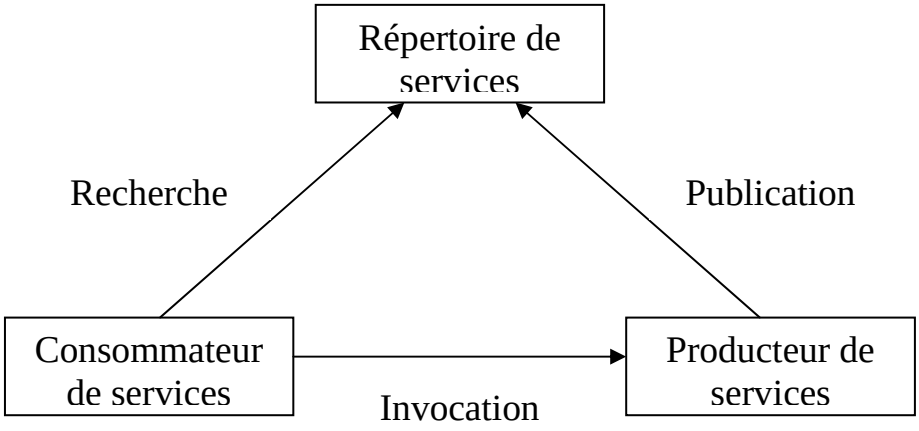

Figure 1.1 – Architecture orientée services

Bien que les SOAs n'ont vraiment pris leur essor que ces dernières années. Il existe des propositions plus anciennes sur ce même modèle tels que CORBA (Common Object Request Broker Architecture) ou DCOM. Les SOAs sont en effet plus connues sous leur version Web services (Web Services Oriented Architectures ou WSOA) car elles reposent alors sur des technologies basées sur des standards XML (eXtensible Markup Language) et offrent ainsi une solution indépendante des implémentations, des plates-formes et des langages ou encodage de données propriétaires. Les trois protocoles de base sont UDDI (Universal Description, Discovery and Integration), WSDL (Web Services Description Language) et SOAP (Simple Object Access Protocol).WSDL permet de décrire un service, UDDI de l'enregistrer et de le découvrir, et SOAP est utilisé comme protocole de transport pour envoyer les messages entre les consommateurs et les producteurs de services.

1.2.2 La notion de couplage faible

Tout comme le concept de SOA, le concept de couplage faible ne donne pas lieu à une définition officielle qui fasse l'unanimité. L'objectif communément admis est cependant d'introduire le minimum de dépendances entre services pour leur permettre un assemblage plus aisé. Dans le contexte SOA, il s'agit notamment de favoriser la réutilisabilité de sous systèmes existants, déjà déployés, et de leur combinaison afin de répondre rapidement, et à faible coût, à de nouveaux besoins métier. Pour atteindre cet objectif, un certain nombre de règles d'ingénierie, spécifiques ou non au contexte SOA, ont été identifiées [1, 5, 6, 7, 8, 9] comme favorisant le couplage faible et la réutilisabilité.

Du paradigme orienté objet, sont repris les principes d'encapsulation et d'abstraction. L'idée est de cacher aux utilisateurs l'information contenue dans un objet et de ne proposer qu'une interface stable mettant l'accent sur les détails jugés nécessaires à sa manipulation. L'objet est vu de l'extérieur comme une boîte noire. Cela permet de découpler sa spécification (ce qu'on voit) de l'implémentation pratique de ces spécifications. On peut ainsi changer son implémentation sans changer son comportement externe. Des systèmes distribués ont notamment retenu le modèle Publication/Souscription et le principe de la granularité à gros grain.

- Publication/Souscription est un modèle de communication où des producteurs de données publient des messages auxquels souscrivent des consommateurs. Ce modèle,mis en œuvre par des brokers tels que par exemple les bus EAI (Entreprise Application Integration) et ESB (Entreprise Service Bus), favorise le découplage entre producteurs et consommateurs distribués, le passage à l'échelle et le déploiement incrémental des services.

- L'idée de la granularité à gros grain est que les opérations exposées doivent refléter des besoins métier - une notion relative mais qui vise à améliorer la réutilisabilité, l'efficacité et la robustesse des communications permettant de réaliser un besoin métier en évitant de multiplier les échanges distribués et de gérer un état transitoire (*e.g.,* une seule opération pour passer un ordre d'achat, au lieu d'accesseurs individuels suivis d'une validation).

Les règles suivantes sont plus spécifiques au contexte SOA :

- L'indépendance aux aspects technologiques (*e.g.,* pas de références distribuées) : Les contrats d'utilisation qu'exposent les services doivent être auto-descriptifs et indépendants de la plate-forme technique utilisée par le fournisseur du service. Par exemple, le langage WSDL permet d'exposer les fonctionnalités des Web Services en assurant cette indépendance.

- L'absence d'état : En toute rigueur, un service bien formé est sans état. Il reçoit les informations nécessaires dans la requête d'un client et ne le reconnaît plus, une fois la réponse est renvoyée. Cette règle qui peut sembler très contraignante doit être

nuancée. Il est recommandé que la conservation des états (gestion de contextes) ainsi que la coordination des actions (transactions) soient localisées dans une fonction spécifique de l'architecture SOA ; telle que l'orchestration. C'est le processus qui est avec état et non pas les services orchestrés. L'application d'une telle règle favorise la réutilisabilité, le passage à l'échelle et la robustesse des services.

- La cohésion : Cette règle, délicate à définir, traduit le degré de proximité fonctionnel des opérations exposées par un service. Elle vise à favoriser la facilité de compréhension et la réutilisabilité d'un service en y regroupant des opérations homogènes constituant une unité métier, ou de même catégorie comme des alternatives pour effectuer un paiement.

- L'idempotence : Un service idempotent ignore les réceptions multiples d'une même requête. Celui-ci est rendu correctement que la requête arrive une ou plusieurs fois. L'idée est que l'utilisation d'un tel service permet de relâcher les hypothèses de fiabilité sur la couche de communication.

On notera que, si certaines règles sont bien maîtrisées (*e.g.,* encapsulation et indépendances aux aspects technologiques), d'autres sont plus contraignantes (*e.g.,* idempotence) et à ce titre moins appliquées.

1.2.3 Infrastructures et modèles d'exécution pour SOA

SOA est souvent présentée comme un style architectural permettant aux entreprises de créer rapidement de nouvelles applications, en transformant leurs ressources existantes en services réutilisables et en les composant aisément avec d'autres services en fonction des besoins. Si l'accent est effectivement mis sur les notions de services « métier » et de couplage faible, dans la pratique ; le besoin d'infrastructures et de modèles d'exécution facilitant l'intégration et l'administration d'applications orientées services, ainsi que la prise en compte de services techniques tels que la sécurité, les transactions, ou la QoS, s'est rapidement fait sentir. Plusieurs initiatives sont apparues pour répondre à ce besoin, parmi lesquelles on peut citer SOC, ESB et SCA que nous présentons brièvement ci-dessous. Nous nous intéresserons spécifiquement aux Services Web dans la partie suivante.

SOC (Service Oriented Computing) [10] est une initiative académique qui vise à étendre SOA pour permettre d'administrer et composer les services de façon flexible. L'architecture proposée distingue trois niveaux. Le premier couvre SOA avec ses fonctions minimales de publication, découverte et liaison de services. Le deuxième porte sur la composition dynamique de services. Il est chargé d'adapter en cours d'exécution les services composites (processus, workflow, etc.) pour tenir compte d'observations telles que des mesures de QoS sur les applications coopérantes, de leurs contraintes (SLA, etc.) ou de la découverte de nouveaux services. Le troisième niveau couvre les fonctions d'administration nécessaires à la supervision globale des applications. Une conférence annuelle (ICSOC) fédère les développements et les applications de l'approche dans des contextes variés, comme la découverte et la composition dynamique de services dans les environnements mobiles.

ESB (Enterprise Service Bus) [11] est une réponse industrielle aux approches SOA qui porte plus spécifiquement sur le domaine de l'intergiciel et de l'intégration des systèmes d'information.
Apparu fin 2002, le concept donne lieu à plusieurs implémentations industrielles ou open-source pour les Web services. Les services minimaux d'un ESB servent a :

- faciliter les communications et les interactions de services. Un ESB supporte au moins un style de communication par message (*e.g.*, request/response, publish/subscribe, etc.), un protocole et un format de définition d'interfaces (*e.g.*, SOAP/WSDL). Il gère le routage, l'adressage, la transparence à la localisation, la substitution de services, et permet de découpler la vue utilisateur d'un service de son implémentation.

- faciliter l'intégration et la supervision de services. Un ESB gère les adapteurs permettant d'interopérer avec des systèmes *legacy*. Il permet d'ajouter des fonctions de supervision, par injection au niveau de points d'interception ou de dérivation, sans actions intrusives auprès des applications.

Les ESBs plus avancés intègrent des services à valeur ajoutée (mais propriétaires) allant de mécanismes variés pour la QoS tels que : l'écrêtage de flux, plage d'ouverture ou de remontée d'alarmes, au traitement sémantique des messages tel que la réécriture et l'agrégation d'information. Ils peuvent également intégrer des fonctions d'orchestration.

SCA (Service Computing Architecture) [12] est une initiative récente de plusieurs éditeurs logiciels, incluant notamment BEA, IBM, IONA, Oracle et SAP, qui vise à proposer un modèle à composants pour construire des applications dans un contexte SOA. SCA encourage une organisation basée sur des composants explicites qui implémentent la logique métier et communiquent à travers des interfaces de services fournies et requises masquant tout détail ou dépendance sur des APIs techniques. Le modèle distingue l'étape d'implémentation des composants de celle d'assemblage d'un ensemble de composants, consistant à lier leurs services. Il est récursif comme les modèles à composants de type Fractal. Les spécifications SCA couvrent également le moyen de packager des composants formant une unité de déploiement, et s'intègrent avec SDO (Service Data Object) qui complètent l'architecture SCA par une solution commune d'accès à différents types de données. Cette initiative est intéressante car :

- d'une part elle illustre la complémentarité entre les approches à composants et services plutôt que de les opposer, comme souvent résumé dans les débats sur le couplage fort et faible

- d'autre part, elle propose une réponse à des besoins réels du marché et des utilisateurs en matière de plates-formes d'exécution et de déploiement de SOA.

1.2.4 Le modèle de référence de OASIS

OASIS travaille sur un modèle de référence pour les architectures orientées services [2]**.** Cet effort a pour volonté de définir un cadre conceptuel commun qui pourra être utilisé de façon consistante à travers les différentes implémentations. Le but est donc : d'établir les définitions qui devraient s'appliquer à toutes les SOAs, d'unifier les concepts pour expliquer les patrons de conception génériques sous-jacents et de donner une sémantique qui pourra être utilisée de façon non ambiguë lors de la modélisation de solutions données.

OASIS définit une SOA comme étant un *"paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains"*. Plus précisément, le modèle de référence de OASIS repose sur les notions de besoins (needs) et de capacités (capabilities). Ainsi, des personnes ou des organisations créent des capacités afin de résoudre

et d'apporter une solution à leurs problèmes. Les besoins d'une personne peuvent être adressés par les capacités offertes par une autre personne. Il n'y a pas de relation une à une entre les besoins et les capacités. De plus, la granularité des besoins et des capacités est guidée par le métier. Un besoin donné peut nécessiter de combiner plusieurs capacités, et une capacité peut répondre à plus qu'un seul besoin. Une architecture orientée service offre donc un cadre pour faire correspondre des besoins à des capacités et de combiner des capacités pour répondre à ces besoins.

Service et dynamicité

Dans le modèle de OASIS le concept central est évidemment le service. Le modèle de référence souligne que cette notion de service communément correspond implicitement soit à la capacité d'effectuer une tâche pour un tiers, soit à la spécification de la tâche qui est offerte par un tiers, soit enfin à l'offre d'effectuer une tâche pour un tiers. Plus précisément, ils définissent un service comme étant un mécanisme qui permet l'accès à un ensemble constitué d'une ou de plusieurs capacités, où l'accès est rendu possible à travers une interface définie et est effectué de façon cohérente avec les contraintes et politiques spécifiées par la description de service. La personne qui offre un service est un fournisseur de service et celui qui l'utilise est un consommateur. Un service est considéré comme étant opaque dans le sens où son implémentation est cachée au consommateur de service. Autour de la notion de service, les concepts clés pour décrire le paradigme SOA et qui sont introduits dans une perspective de dynamicité sont la visibilité, l'interaction et l'effet. La visibilité fait référence à la possibilité pour les fournisseurs et les consommateurs de se "voir" mutuellement en vue d'interagir. Cela implique notamment la mise en place de mécanismes de découverte de service ainsi que la mise à disposition de toutes les informations nécessaires pour qu'un consommateur potentiel puisse prendre connaissance de l'existence et des capacités d'un service donné. L'interaction correspond à l'activité d'utilisation d'une capacité. Cela s'effectue typiquement à travers l'échange de messages. Enfin le but principal lors de l'utilisation d'une capacité est de réaliser un ou plusieurs effets. En plus de ces concepts liés au support dynamique des interactions avec des services, le modèle OASIS identifie des concepts liés aux services eux-mêmes. Ces concepts regroupent notamment la description de service, ainsi que les contrats et les politiques qui sont liés aux services et aux fournisseurs et consommateurs de services.

Description de service

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

Ainsi le consommateur a besoin de savoir :

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

Politiques et contrats

Une politique représente une contrainte ou une condition définie par un consommateur ou un producteur sur l'utilisation, le déploiement ou la description d'une entité qu'il possède. Ainsi, une politique peut par exemple spécifier que tous les messages reçus sont cryptés. Plus largement, une politique peut être appliquée sur différents aspects d'une SOA tels que la sécurité, la confidentialité, la qualité de service, ou même des propriétés métier. Garantir qu'une politique est respectée peut nécessiter d'empêcher certaines actions non autorisées ou le passage dans un état donné. Il peut aussi être nécessaire de mettre en oeuvre des actions pour compenser lorsqu'une violation de politique a été détectée. Détecter une violation de politique peut être réaliser sous la forme d'assertions qui doivent être compréhensibles et traitables de préférence de façon automatique. Un contrat représente, quant à lui, un accord entre au moins deux parties. Tout comme les politiques, les contrats portent sur les conditions d'utilisation des services. Un contrat doit lui aussi être exprimé sous une forme interprétable, et cela afin de faciliter la composition automatique de services. Le respect d'un contrat,

contrairement aux politiques où le bon respect d'une politique est la responsabilité du propriétaire de la politique, peut nécessiter la résolution de désaccords qui éventuellement est à la charge d'une autre entité faisant autorité.

1.3 Les services Web

Les services Web constituent une approche pour mettre en oeuvre le paradigme de service. Les services Web [17] ont été proposés initialement par IBM et Microsoft, puis en partie standardisés sous l'égide du W3C (W3 Consortium). Le groupe de travail du W3C a produit un glossaire [16] dans lequel il définit un Web service comme suit :

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL[2]). Other systems interact with the Web service in a manner prescribed by its description using SOAP[3] messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Webrelated standards."

IBM donne dans un tutoriel[4] la définition suivante des Web services :

"Les web services sont la nouvelle vague des applications web. Ce sont des applications modulaires, auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le web. Les Web services effectuent des actions allant de simples requêtes à des processus métiers complexes. Une fois qu'un Web service est déployé, d'autres applications (y compris des Web services) peuvent le découvrir et l'invoquer."

D'autres définitions similaires adoptent un point de vue plus général en ne prescrivant pas l'usage exclusif de WSDL pour la description des interfaces de service ou celui de SOAP pour le traitement des messages. Ainsi par exemple, certains auteurs proposent l'utilisation de messages XML (sans les extensions apportées par SOAP) échangés directement sur le protocole HTTP. De façon similaire, la définition officielle du consortium W3C fait

[2] Web Service Description Language.
[3] Simple Object Access Protocol.
[4] http://webservices.xml.com/pub/a/2001/04/04/webservices/index.html

spécifiquement référence au protocole HTTP, mais en pratique, d'autres protocoles sont aussi utilisés.

Une étude plus approfondie des points communs partagés par les différentes définitions et par les usages qui sont faits des services Web, permet de dégager au moins deux principes fondamentaux :

- Les services Web interagissent a travers l'échanges de messages encodés en XML. Dans certains cas, les messages sont plutôt transmis dans un encodage binaire, même s'ils sont généralement produits et consommés en XML ou en utilisant des interfaces de programmation basées sur les concepts d'*élément* et d'*attribut* codéfinis par le modèle dit « XML Infoset ».

- Les interactions dans lesquelles les services Web peuvent s'engager sont décrites au sein d'interfaces. La définition W3C restreint la portée des interfaces des services Web aux aspects fonctionnels et structurels, en utilisant le langage WSDL qui, essentiellement, ne permet de décrire que des noms d'opérations et des types de messages. Cependant, d'autres auteurs proposent d'aller plus loin et de considérer aussi les aspects comportementaux et non fonctionnels.

Par rapport à d'autres plates-formes pour le développement d'applications distribuées telles que CORBA et Java RMI, l'une des différences primordiales est que les services Web n'imposent pas de modèles de programmation spécifiques. En d'autres termes, les services Web ne sont pas concernés par la façon dont les messages sont produits ou consommés par des programmes. Ceci permet aux vendeurs d'outils de développement d'offrir différentes méthodes et interfaces de programmation au-dessus de n'importe quel langage de programmation, sans être contraints par des standards comme c'est le cas de CORBA ; qui définit des ponts spécifiques entre le langage de définition IDL[5] et différents langages de programmation. Ainsi, les fournisseurs d'outils de développement peuvent facilement différencier leurs produits avec ceux de leurs concurrents en offrant différents niveaux de sophistication. Par exemple, il est possible d'offrir des environnements pour des méthodes de programmation de services Web minimalistes au-dessus de plates-formes relativement légères comme c'est le cas de NuSOAP (pour le langage PHP) ou d'Axis. En revanche, des plates-

[5] *Interface Definition Language*

formes plus lourdes telles que BEA WebLogic, IBM Websphere ou .Net (voir le chapitre) offrent un environnement pour des méthodes de développement très sophistiquées.

Les principes à la base des services Web, bien que simples, expliquent leur adoption rapide : le nombre d'outils associés au services Web a connu un essor considérable dans une période de temps réduite, assurant ainsi que les technologies associées aux services Web seront utilisées pour de nombreuses années à venir. La description explicite des interactions entre les services Web est aussi un point fort des services Web. Cependant, des efforts de recherche et de développement plus approfondis sont nécessaires afin d'exploiter réellement ce principe dans des outils ou des méthodes de développement. En particulier, l'usage des descriptions comportementales et non fonctionnelles des services requièrent davantage d'attention de la part de la communauté de recherche.

1.4 Architecture en couches

Une architecture services Web est une architecture en couche qui repose sur plusieurs protocoles [13]. De nombreuses organisations telles que WebServices.Org, The Stencil group, IBMconceptualWeb services stack (WSCA),W3CWeb Services,Microsoft, Sun One (webservice architectures), Oracle, Hewlett-Packard, BEA Systems ou Borland proposent leur propre modèle [15]. A titre d'exemple, la *figure 1.2* illustre la pile de protocoles proposée par le W3C, appelée la "Web Service Architecture" (WSA).

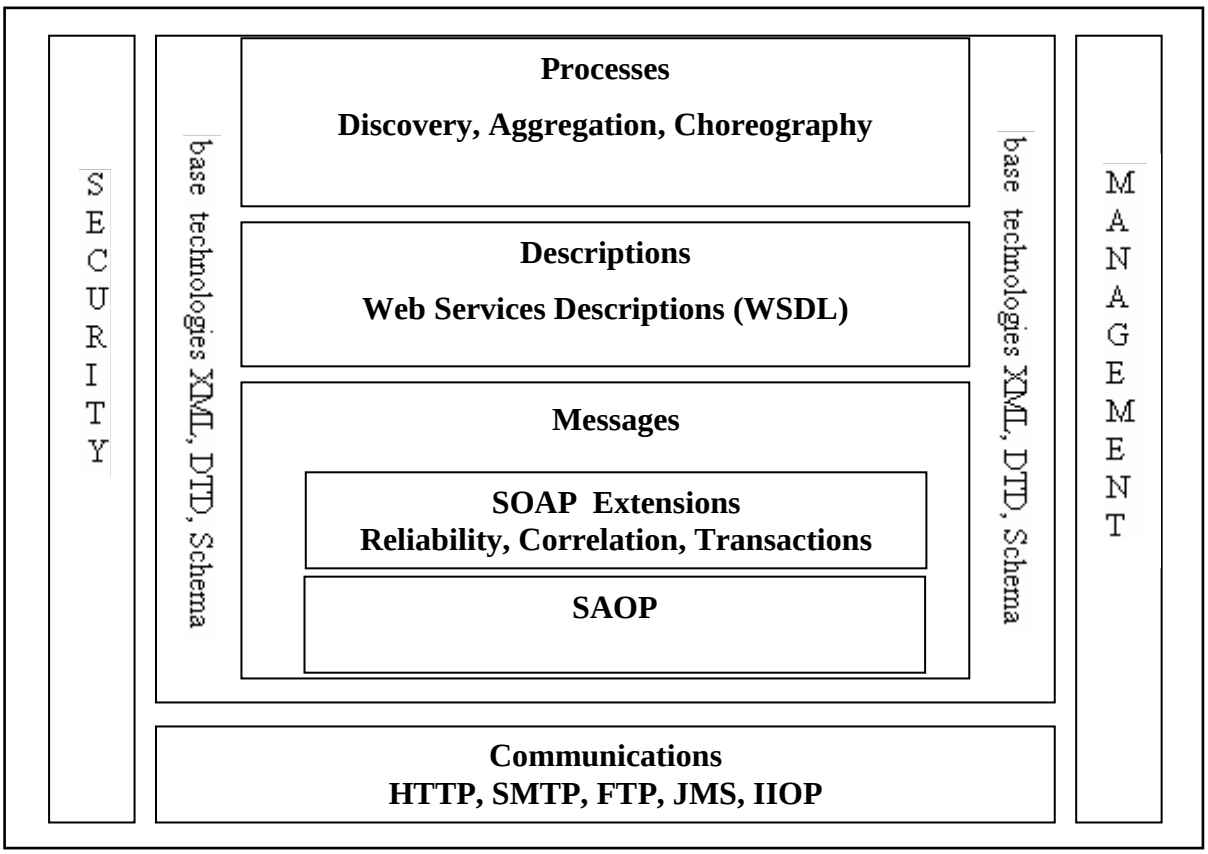

Figure 1.2 – Architecture des services Web du W3C

Il y a trois couches dans la WSA : communication, message et gestion des processus. Chacune de ces couches se subdivise en domaine de granularité plus fine (*e.g.*, les normes de sécurité font partie d'un domaine de la couche message). La couche communication adresse les aspects liés au transport des messages. Il est souvent possible de spécifier le style, le mode et l'encodage d'une communication. On distingue au moins deux styles de communication : le style RPC (Remote Procedure Call) et le style Document. En RPC, le Web service est similaire à l'approche distribuée traditionnelle. Cependant, de nombreuses variantes existent. Le style Document correspond à une communication où le client interagit avec le serveur non plus sous la forme de procédures mais de documents XML auto-descriptifs qui sont échangés. Trois mode de communication peuvent être envisagés : RPC ou mode requête-réponse, one-way messaging et asynchronous callback. Enfin l'encodage, qui spécifie comment les types sont représentés en XML peut être Literal (*i.e.*, suit littéralement les définitions de XML Schema) ou SOAP encoded (*i.e.*, suit la spécification de SOAP). Enfin le protocole de communication support le plus souvent utilisé est HTTP mais il peut être aussi SMTP, FTP, etc. La couche message décrit la sémantique des messages SOAP. Composées

fondamentalement par le trio SOAP, WSDL et UDDI, les spécifications de cette couche ont également pour objectif d'offrir aux applications distribuées des mécanismes complexes de sécurité, de garantie de livraison, de transaction, de description, etc. Ces spécifications évoluent régulièrement. Des organismes, tels que le WS-I, sont chargé de prendre en compte toutes les propositions et de valider l'interopérabilité.

La couche gestion des processus regroupe tout ce qui est notamment lié à la composition et au management des services. Le management des Web services peut être défini comme un ensemble de capacités telles que la découverte de l'existence, la disponibilité, la santé, les performances, le contrôle et la configuration des ressources à l'intérieur des architectures à base de Web services. C'est le Web Services Distributed Management (WSDM) Technical Committee qui est en charge des normes de la gestion standardisée des Web services. La composition de services peut être simplement définie comme étant "le procédé consistant à combiner des services existants pour former de nouveaux services". Combiner des services peut se résoudre de deux façons distinctes : soit par le biais d'une chorégraphie, soit via une orchestration. Une chorégraphie décrit le flux de messages échangés par un Web service lors de son interaction avec d'autres services. Il s'agit donc d'une manière décentralisée de gérer une composition puisque chaque Web service est responsable d'une partie du workflow. La norme "Web Services Choregraphy Interface" (WSCI) [18] permet de spécifier le comportement du Web service par rapport au reste de la composition. Une orchestration comprend un élément central responsable de la composition dans son ensemble. Le processus devient alors la somme de ses sous processus et l'orchestrateur gère, seul, les échanges de messages. De nombreux travaux se sont intéressés aux langages dédiés à l'exécution du processus métier, parmi lesquels WSFL d'IBM et XLANG de Microsoft. Ces efforts ont ensuite été fusionnés pour former une spécification commune nommée "Business Process Execution Language for Web Services" (BPEL4WS) [14].

1.5 Standards industriels

Les Web services reposent sur un certain nombre de standards. Les trois standards principaux sont : SOAP,WSDL et UDDI que nous détaillons ci-dessous. Notons qu'il existe de ombreuses plates-formes pour les Web services, comme par exemple le Web Services Platform de Sun ou le Web Services Toolkit d'IBM. Diverses implémentations pour standards

sont aussi disponibles, comme par exemple Apache Axis pour SOAP, jUDDI pour UDDI qui sont tous les deux issus du Jakarta Apache Project, ou encore le SOAP Toolkit de Microsoft.

1.5.1 Déploiement, recherche et invocation de services Web

Le cycle de vie d'un service Web se déroule de la façon suivante : une fois créé, le service est déployé sur le réseau (local ou Internet). Puis un utilisateur ayant des besoins spécifiques va rechercher un service correspondant à ses besoins à l'aide d'un annuaire spécialisé. Enfin, une fois le service trouvé, l'utilisateur va invoquer le service : une communication va être mise en place entre l'utilisateur et le service Web. Ce cycle de vie, représenté par la *figure 1.3*, fait appel à trois grandes technologies : SOAP, WSDL, UDDI.

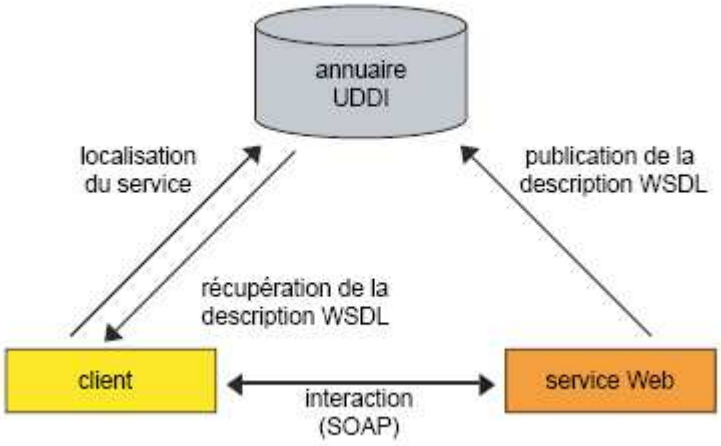

Figure 1.3 – Déploiement, découverte et invocation de services Web

1.5.2 SOAP

Originellement appelé SOAP pour "Simple Object Access Protocol", le protocole d'échange de messages entre Web services est un standard ouvert et entièrement basé sur le langage XML. Ce protocole ne requiert aucune plate-forme spécifique, ni d'exigence particulière en ce qui concerne l'implémentation de l'application. Au contraire, ce mécanisme fournit une sémantique tout à fait indépendante des contraintes techniques et, de fait, compréhensible par tous. Dans la pratique, le transfert est le plus souvent assuré via le protocole HTTP. SOAP peut cependant reposer sur d'autres protocoles de transport comme par exemple SMTP ou JMS (Java Message Service). Un message SOAP est un document

XML dont la structure est spécifiée par des schémas XML. Plus précisément tout message SOAP se compose d'un unique élément *envelope* qui englobe une partie *header* et une partie *body*. La partie header contient les méta données concernant d'éventuelles propriétés non fonctionnelles du service (jeton de sécurité, contexte de transaction, certificat de livraison, etc.) ; tandis que la partie body regroupe les éléments métier tels que les appels de méthode avec transfert des données spécifiques. Cette structure permet une séparation nette des différentes préoccupations auxquelles doit répondre le service. Généralement, les données header des messages SOAP sont traitées par des filtres (handlers) placés autour de l'implémentation fonctionnelle du service. Conceptuellement, le header offre une possibilité d'extension des messages en rajoutant des données spécifiques au contexte. Toutefois, le consommateur et le producteur du service doivent se comprendre sur la sémantique de ces données supplémentaires, d'où le rôle fondamental de normalisation des travaux de la "Web Service Architecture".

1.5.3 WSDL

Comme tout composant, les services Web sont atteignables par le biais d'interfaces. Il s'agit du contrat WSDL qui s'apparente aux Interface Definition Languages (IDL) déjà existants, puisqu'il s'intéresse à fournir une abstraction fonctionnelle du service. WSDL repose sur XML pour décrire de façon abstraite (et donc indépendante de l'implémentation du service) l'ensemble des opérations et des messages qui peuvent être échangés avec un service donné. Une description WSDL contient aussi des informations sur le protocole de communication utilisé, le format d'encodage des données et l'adresse URL à laquelle le service est accessible. Un contrat rédigé en WSDL doit permettre au client de trouver l'adresse du service et de rédiger des messages compréhensibles par ce dernier. Ainsi, le WSDL se compose en deux parties. Une première partie définit de manière abstraite les éléments, les opérations et les types de données, tandis qu'une seconde partie précise de manière concrète les adresses physiques de ces opérations ainsi que le mapping des données avec les protocoles de transport. Cette distinction est utile car elle permet de concevoir le service indépendamment de l'environnement de déploiement. Plus précisemment, le WSDL définit les services Web à travers six éléments :

- ***portType*** : combine plusieurs messages pour former une opération. Il y a quatre types d'opération : one-way, request-response, solicit-response et notification.
- ***message*** : définit les informations échangées lors d'une requête ou d'une réponse. Un message à un nom et potentiellement des *parts* qui font référence à des paramètres et des valeurs de retour.
- ***types*** : décrivent sous la forme d'une spécification XML Schema les types des données échangées entre le client et le fournisseur de services.
- ***binding*** : spécifie le protocole de communication (le plus souvent HTTP, mais aussi SMTP, FTP, etc.) et le format d'encodage des données (encodage RPC, Literal Document, etc.) pour les opérations et messages définis par un type de port donné. Il est possible grâce à des extensions internes de WSDL de définir des binding SOAP.
- ***service*** : regroupe un ensemble de ports, chaque port offre une alternative (différents protocoles, etc.) pour accéder au service.
- ***port*** : est un point d'accès au service identifié de manière unique par la combinaison d'un binding et d'une adresse internet.

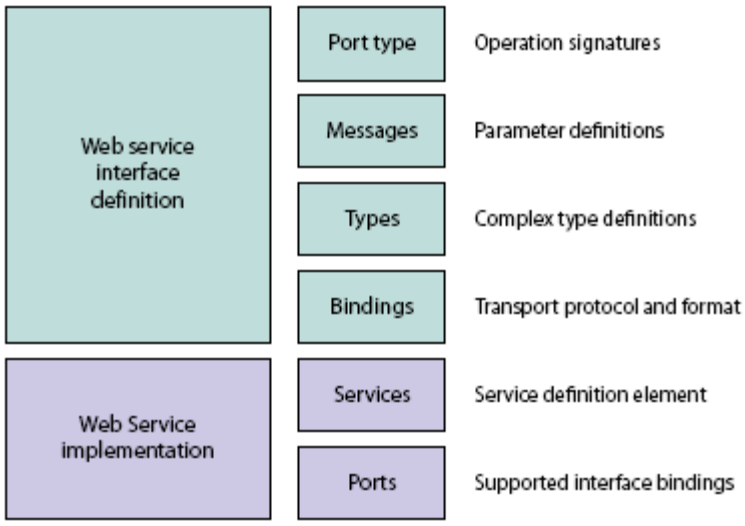

Figure 1.4 – Architecture WSDL

1.5.4 UDDI

Il existe également un mécanisme de découverte des Web services : il s'agit de "Universal Discovery Description and Integration" (UDDI) qui peut être considéré comme un annuaire deWeb services. UDDI permet aux entreprises de s'inscrire et de publier leurs services Web. UDDI se comporte lui-même comme un Web service dont les méthodes sont

appelées via le protocole SOAP. Les opérations pouvant être effectuées concernent la recherche, la navigation, l'ajout et la suppression de services. UDDI permet aux développeurs de découvrir les détails techniques d'un service Web (le WSDL) ainsi que les informations orientées métier. Par exemple, en utilisant UDDI, il devient possible de répertorier tous les services Web reliés aux informations boursières, implémentés sur HTTP, et qui sont gratuits d'utilisation. De plus, UDDI permet de fournir un niveau d'indirection permettant un couplage faible entre les services d'une composition. Concrètement un répertoire UDDI est un fichier XML qui décrit un business et les services qu'il offre.

Il existe plusieurs catégories de classification pour les services Web :

– Pages blanches : les Web services sont classés par fournisseurs de services,
– Pages jaunes : les services Web sont classés par catégories,
– Pages vertes : contiennent des informations techniques sur les services Web.

1.6 Agrégation de services Web

« Les composants sont destinés à être composés. » La réutilisation est un avantage des approches par composants. Les services Web, tels qu'ils sont définis actuellement, sont limités à des fonctionnalités relativement simples. Toutefois, pour certains types d'applications, il est nécessaire de combiner un ensemble de services Web « basiques » en un service plus « sophistiqué » afin de répondre à des exigences plus complexes : c'est l'agrégation de services web. L'agrégation de services constitue l'un des champs de recherche les plus actifs dans le domaine des services Web. Plusieurs courants sont d'actualité, comme l'orchestration et la composition en utilisant des techniques d'intelligence artificielle. La définition d'un service Web par sa spécification WSDL n'est pas assez riche pour la composition de services. En effet, WSDL permet de décrire un service du point de vue technique : les ports, les types de messages échangés... mais ne donne aucune information sur le rôle que doit jouer le service d'un point de vue fonctionnel : le service Web n'a pas de description sémantique.

1.7 Services Web et Web Sémantique

Depuis le début des années 2000, de nombreux axes de recherches concernant l'Internet se sont tournés vers le Web sémantique. L'Internet est un immense réservoir de documents en tout genre (articles, sites personnels, audios, vidéos) mais plus ces ressources s'accumulent, plus la recherche d'une information précise s'avère difficile. Tim Berners-Lee [22], pionnier dans le domaine d'Internet, propose alors d'ajouter à toutes ces ressources une sémantique qui permettrait aux systèmes informatiques d'en « comprendre » le sens en accédant à des collections structurées d'informations et à des règles d'inférence qui peuvent être utilisé pour conduire des raisonnements automatisés : c'est la naissance du Web sémantique.

Suite à l'explosion du commerce électronique, il a été nécessaire d'offrir toutes sortes d'applications et de services via Internet. La communauté des « commerces et organisations accessibles par le Web » a fait le plus gros du travail : ils ont connecté une énorme quantité de produits et services en tout genre à Internet, en les rendant accessibles à des ordinateurs a travers des protocoles de communication simples. Mais parallèlement, le domaine des agents intelligents a également envahit Internet [20]. Maintenant, le but recherché par ces agents est de pouvoir se servir des services Web afin de mieux satisfaire leurs utilisateurs (par exemple, pouvoir proposer l'organisation d'un voyage en faisant appel à différents services Web). Pour cela, il est nécessaire que ces agents puissent comprendre les fonctionnalités proposées par les services : c'est la naissance des services Web sémantiques.

1.7.1 Les services Web sémantiques

L'idée consiste à faire converger le Web sémantique et les services Web afin de proposer des services et des procédés qui prennent en compte la connaissance que l'on peut avoir d'un service ou d'un procédé afin de pouvoir profiter des travaux sur le raisonnement à partir de connaissances qui ont été mises en avant par le Web sémantique. Les premiers travaux sur les services web sémantiques exposés dans [21] proposent une approche basée sur l'utilisation de langages de descriptions de la famille de DAML. Cette approche permet de capturer les données et méta-données associées à un service. Ces données spécifient les propriétés et capacités du service ainsi que les prérequis et les conséquences de son exécution.

Le langage de description de services Web sémantiques de référence est le langage OWL-S[6], une extension du langage OWL[7], proposé par le groupe de recherche sur le Web sémantique du DARPA[8] à l'origine du langage DAML[9].

1.8 Conclusion

Les services Web sont une conséquence naturelle de l'évolution du Web dans un moyen ouvert qui facilite des applications complexe du business et des interactions des applications scientifiques. Les services Web s'occupent des problèmes d'interactions systématiques entre applications sur le Web et l'intégration ou l'utilisation de l'infrastructure réseau existante dans le Web. Les composants clés de cette partie sont concentrés sur l'interoperabilitié, le support efficace pour l'intégration d'application et la création d'une représentation uniforme d'applications dans les systèmes distribués hétérogènes. Les services Web proposent deux approches, négocier avec les exigences de l'interopérabilité des systèmes hétérogènes: permet une interopérabilité qui utilise un petit ensemble de protocoles communs, et développer une représentation uniforme d'applications réseau qui sont accessible en utilisant de multiples protocoles de communication. Le but des services Web est de fournir une structure flexible où l'interopérabilité n'empêche pas une intégration efficace.

Souvent un service web ne peut pas répondre aux besoins de l'utilisateur, alors une solution est nécessaire, ce qui a permet l'émergence de la composition de services que nous allons aborder dans le chapitre suivant.

[6] Ontology Web Language for Service
[7] Ontology Web Language
[8] Defense Advanced Research Projects Agency
[9] DARPA Agent Markup Language. http ://www.daml.org/

Chapitre 2

Composition de services Web : techniques et description.

Chapitre 2

Composition de services Web : techniques et description.

2.1 Introduction

La réutilisation joue un rôle important dans le développement de logiciel. L'une de ses principales applications dans les logiciels est la composition, qui est simplement défini comme étant le développement de logiciels en réutilisant les applications existantes. Comme l'un des plus récents exemples, *la composition des services Web* est un challenge en ingénierie des logiciels. Les services Web sont les derniers produits de la technologie de *Calcul Orienté Services "COS"* [10] et définit par W3C comme système logiciel désigné pour supporter l'interopérabilité et l'interaction machine à machine à travers un réseau. Ils ont une interface décrite dans un format traité par la machine (en WSDL). Les autres systèmes interagissent avec les services Web en utilisant des messages SOAP qui sont transmis avec le protocole HTTP. L'auteur dans [23], identifie deux types de services Web: *simples* et *composites*. Les Services simples (basiques ou élémentaires), sont des applications basées sur Internet qui ne reposent pas sur d'autres services Web pour accomplir les requêtes des utilisateurs ; par contre les services composites, sont définis comme une conglomération de services Web (appelés services participants) travaillant en collaboration pour offrir un autre service.

Dans cette partie, nous passons en revue les différentes techniques de composition de services Web ainsi les approches de la description de la composition. Nous pouvons classer les techniques de composition en deux grandes familles : Les techniques de composition dites *« statiques »*, i.e., qui sont définies à l'aide de processus métier (orchestration et chorégraphie) ; et les techniques de composition dites *« dynamiques »*, lorsque la composition de services Web tient compte des services disponibles, de leurs fonctionnalités et du but à atteindre que ce soit avant ou pendant l'exécution des services Web. Les techniques de compositions dynamiques peuvent être regroupées en deux sous familles : les techniques utilisant une approche basée sur les workflows (processus métier) et celles basées sur des techniques d'intelligence artificielle. Nous présentons après la problématique de la description des compositions de services. Plusieurs approches existent pour décrire les compositions de services Web. Nous évoquerons en particulier BPEL4WS et OWL-S.

2.2 Les techniques de composition

2.2.1 Orchestration et Chorégraphie

L'orchestration de services Web permet de définir l'enchaînement des services selon un canevas prédéfini, et de les exécuter à travers des « scripts d'orchestrations ». Ces scripts décrivent les interactions entre services en identifiant les messages échangés, les branchements logiques et les séquences d'invocation. La chorégraphie trace la séquence de messages pouvant impliquer plusieurs sources (les clients, les fournisseurs, les partenaires). Elle est associée à l'échange de messages publics entre services Web plutôt qu'un processus métier exécuté par un seul partenaire. Il existe une différence importante entre orchestration et chorégraphie de services Web [31] : L'orchestration se base sur un processus métier exécutable pouvant interagir avec des services Web internes et externes. Elle offre une vision centralisée ; le procédé est toujours contrôlé du point de vue de l'un des partenaires. La chorégraphie est de nature plus collaborative: chaque participant impliqué dans le procédé décrit le rôle qu'il joue dans cette interaction. Orchestration et chorégraphie sont des compositions statiques de services Web permettant de créer un processus métier (*workflow*).
Les principaux langages d'orchestration et/ou de chorégraphie de services Web sont répertoriés dans [44] et représentés sur la *figure 2.1* dans l'ordre chronologique d'apparition sur le marché :

- XLANG - XML Business Process Language (Microsoft) [50]
- BPML - Business Process Modeling Language (BPMI) [24]
- WSFL - Web Service Flow Langage (IBM) [35]
- WSCL - Web Service Conversation Language (Hewlett-Packard)
- WSCI - Web Service Choregraphy Interface (SUN)
- BPEL4WS - Business Process Execution Language for Web Services (IBM, Microsoft, BEA)

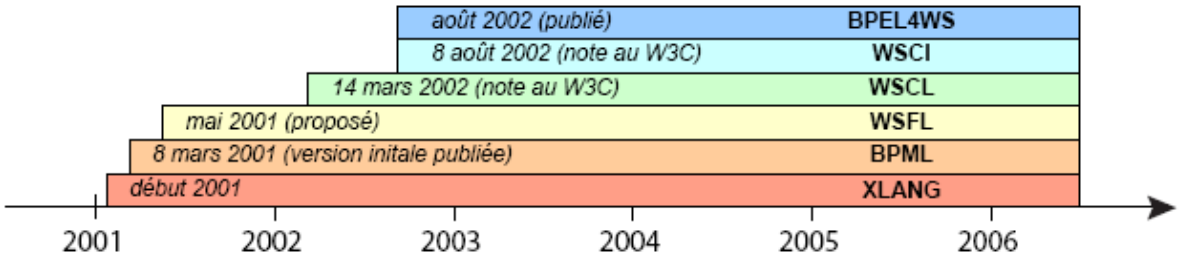

Figure 2.1 – Les langages Orchestration et/ou Chorégraphie

Le but de BPML est de construire un procédé centralisé se chargeant de coordonner les services Web ; c'est un langage d'orchestration. WSCL se charge uniquement de décrire les interactions entre les couples de services Web sans spécifier comment est créé le contenu des messages échangés ; c'est un langage de chorégraphie. Les langages XLANG et WSCI offrent une description du comportement de chaque service Web vis à vis des évènements qu'il pourra rencontrer en décrivant ses opérations et ses messages. La collaboration entre les différents services s'effectue de manière décentralisée. Ce sont donc des langages de chorégraphie. Les langages, comme BPEL4WS, permettent de décrire deux types de collaboration : la coordination centralisée pour décrire l'enchaînement des opérations d'un service Web et la collaboration décentralisée pour définir le contrat d'interaction entre deux services Web (appelé également conversation). Ces langages font de l'orchestration et de la chorégraphie. Qu'il s'agit d'orchestration ou de chorégraphie, l'agrégation des différents services Web se fait de manière statique avant exécution.

Les principaux reproches faits aux langages d'orchestration et de chorégraphie sont les suivants : procédés et services Web sont trop fortement couplés, l'agrégation obtenue est

rigide et donc difficilement modifiable, et les langages sont de trop bas niveau ; c'est-`a-dire trop proches des langages de programmation.

D'autres approches existent, comme par exemple le système Melusine [30 , 31], qui propose l'utilisation d'une couche de médiation entre le workfow et les services Web afin de permettre une meilleure interopérabilité entre services ainsi qu'une plus grande robustesse face aux changements. Par exemple, lorsqu'un service n'est pas disponible au moment de son invocation, la couche de médiation peut se connecter à un autre service. L'objectif du système Melusine est de diminuer le couplage entre procédés et services Web.

2.2.2 Composition dynamique : principes

On appelle composition dynamique l'agrégation de services Web permettant de résoudre un objectif précis soumis par un utilisateur en prenant en compte ses préférences. Cette composition peut se faire avant ou pendant l'exécution des services Web. Le service résultant de cette composition est appelé service composite. A l'heure actuelle, il n'existe pas de standard : les recherches sont principalement académiques.

La composition de services Web est une tâche complexe. Cette complexité est due principalement aux raisons suivantes [46] : le nombre de services disponibles sur le Web est potentiellement très important et en constante augmentation ; les services Web peuvent être créés et modifiés à la volée ; le système de composition doit donc détecter et gérer ces changements ; les services Web sont créés par des organisations différentes qui n'ont pas forcément les mêmes modèles de concepts pour décrire ces services. En effet, deux services qui effectuent la même action peuvent être décrits d'une façon très différente.

Les différentes approches existantes pour la composition de services Web peuvent être regroupées en deux courants.

Le premier se base sur le fait qu'un service Web composite est défini par un ensemble de services atomiques et par la façon dont ils communiquent entre eux. Cette définition est du même type que la manière dont est défini un processus métier. Ce courant propose donc d'adapter les méthodes d'orchestration et de chorégraphie afin de les rendre dynamiques.

Dans le second courant, la composition est vue comme la génération automatique d'un plan d'exécution des services Web. Les approches envisagées sont des approches du domaine de l'intelligence artificielle et plus particulièrement de la planification. Ces approches sont basées sur le fait qu'un service Web peut être spécifié par ses préconditions et ses effets.

2.2.2.1 Approches orientées workflow

D'une certaine façon, un service composite est similaire à un processus métier [27]. Un service composite inclut un ensemble de services atomiques ainsi que les contrôles et échanges de données entre ces services. De la même façon, un processus métier est composé d'un ensemble d'activités élémentaires structurées, ainsi que l'ordre d'exécution entre elles.

Ainsi Casati présente Eflow [26], une plateforme pour la spécification, la création et la gestion de services composites en utilisant les méthodes de génération de workflows statiques mais représentés en interne sous la forme d'un graphe qui peut être modifié dynamiquement lors de l'exécution. Ce graphe contient, en plus des nœuds représentant les services, des noeuds de décision et d'évènements qui permettent une plus grande robustesse du système ; en cas de non disponibilité d'un service Web, celui-ci peut être automatiquement remplacé par un service Web équivalent. Khalaf [32] propose de regarder les chorégraphies créées avec BPEL4WS en tant que services Web, c'est-`a-dire créer des patrons de composition qui pourront être composés, formant ainsi de nouveaux patrons de composition. Cette solution facilite la composition par une intégration plus simple de patrons d'assemblage prédéfinis dans un nouveau patron mais ne règle pas le problème de la dynamicité de la composition de services Web.

Laukkanen et Helin [34] identifient deux solutions possibles pour composer dynamiquement des processus métier : remplacer un service Web dans un processus métier existant par un autre service ayant des fonctionnalités similaires, ou définir un nouveau workflow à partir des services Web disponibles. Les auteurs proposent d'utiliser les descriptions sémantiques des services Web pour pouvoir comparer les fonctionnalités en utilisant les notions de préconditions et d'effets de ***OWL-S***. La solution proposée (représentée dans la figure 2.2) peut être résumée ainsi :

1. Identifier les fonctionnalités requises
2. Trouver les services adaptés grâce à leur description sémantique
3. Créer ou modifier le workflow
4. Exécuter le workflow et vérifier son exécution.

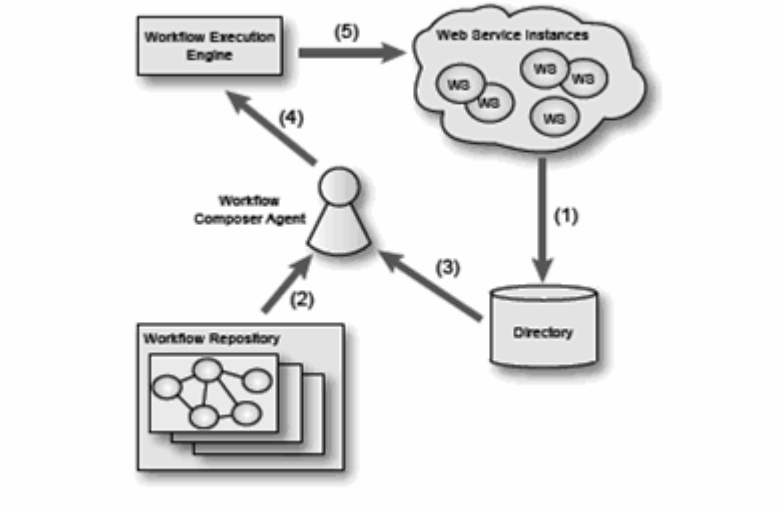

Figure 2.2 – Composition dynamique de workflows

Dans le même esprit, Osman [41] propose de limiter les points négatifs des techniques de composition de type workflow en utilisant les technologies du Web sémantique, plus particulièrement OWL ; il ne s'agit pas directement de composition automatique mais le but est de limiter le nombres de services Web disponibles afin de créer des workflows plus facilement. Pour cela, les auteurs proposent de regrouper les services Web en domaines spécifiques (e.g., domaine des services Web d'hôtels, de transports, etc.) Ainsi le choix des services à utiliser est restreint au domaine des services ayant comme fonctionnalités les objectifs du workflow.

2.2.2.2 Approches orientées intelligence artificielle

La composition dynamique de services Web par des techniques d'intelligence artificielle, et plus particulièrement par des techniques de planification, est la voie qui semble la plus prometteuse [46]. En effet, les concepts de OWL-S sont très fortement inspirés de ceux de la planification, les préconditions présentent les conditions qui doivent être satisfaites pour garantir la bonne exécution d'un service Web. Les effets sont le résultat du succès de

l'exécution d'un service. Dans ce qui suit, nous présentons quelques axes de recherche actuels concernant la composition par planification et par d'autres techniques d'intelligence artificielle.

Calcul situationnel (situation calculus) : McIlraith et al [37] proposent d'adapter et d'étendre le langage Golog pour la construction automatique de services Web. Golog est un langage de programmation logique qui permet de faire du calcul situationnel (i.e., langage logique qui sert à représenter des changements ou évolutions en terme de situations, d'actions et d'objets). Le problème de la composition de services Web est abordé de la façon suivante : la requête de l'utilisateur et les contraintes des services sont représentées en terme de prédicats du premier ordre dans le langage de calcul situationnel. Les services sont transformés en actions (primitives ou complexes) dans le même langage. Puis, à l'aide de règles de déduction et de contraintes, des modèles sont ainsi générés et sont instanciés à l'exécution à partir des préférences utilisateur. Ces recherches ont été étendues dans [40] : les prédicats nécessaires aux calculs situationnels sont tirés de DAML-S et représentés sous la forme de réseaux de Petri.

Theorem proving. : Waltinger [51] a élaboré une approche pour la composition de services par preuve de théorèmes. Cette approche est basée sur la déduction automatique et la synthèse de programmes. Les services disponibles et les requêtes utilisateur sont traduites dans un langage du premier ordre. Puis, des preuves sont produites à partir d'un prouveur de théorèmes. La composition est obtenue à partir de preuves particulières. Dans le même genre, Rao et al [45] ont introduit une méthode de composition automatique de services Web sémantiques en utilisant des preuves de théorèmes logiques linéaires (Linear Logic Theorem Proving).

Système multi-agent : Du fait de leur autonomie et leur hétérogénéité, les services Web peuvent être vus comme des agents. Ainsi Cheng et al [28] proposent le langage ASDL (Agent Service Description Language) qui a pour objectif de décrire le comportement externe des services Web. Une specification ASDL décrit les messages compris par un service, ainsi que les protocoles d'interactions utilisés. Puis, la composition est effectuée à l'aide du langage ASCL (Agent Service Composition Language) qui permet de décrire la logique avec laquelle un nouveau service est composé à partir de services existants. L'avantage de ces travaux est de mettre l'accent sur les interactions entre services au sein de la composition et d'adapter

celle-ci en fonction des contraintes résultant des interactions. Cependant cette méthode n'est pas totalement dynamique. En effet, elle consiste à assembler et adapter au contexte d'exécution des patrons de composition définis a priori par le concepteur. Plus récemment, Muller et Kowalczyk [39] travaillent sur un système multi-agent pour la composition de services basé sur la concurrence entre coalitions de services ; les agents représentants les services se contactent les uns les autres pour proposer leurs services en fonction de leurs capacités de raisonnement et ainsi former des coalitions d'agents capables de résoudre les buts fournis par l'agent utilisateur. Puis, les différentes coalitions vont faire une offre la plus compétitive possible. Chaque solution reçoit une note de l'agent utilisateur. C'est donc la solution ayant le plus haut score qui sera choisie. Ainsi, seuls les services Web correspondant le plus aux attentes de l'utilisateur seront utilisés dans la composition.

Planification : Un intérêt croissant de la communauté planification pour la composition de services Web peut être expliqué par la similarité entre les descriptions OWL-S et les représentations en planification (cf. chapitre 1). En effet, la communauté du Web sémantique s'est fortement inspirée du langage PDDL lors de la conception du langage de description OWL-S. Ainsi, une description OWL-S peut être facilement traduite en une représentation PDDL : les services Web sont représentés par des opérateurs ayant des préconditions et produisants des effets. Puis, un planificateur peut alors être utilisé pour produire un plan qui correspond àla composition des services Web. Carman et al. [25], mais également Constantinescu et al [29] ont utilisé cette correspondance pour composer un ensemble de services Web de manière intuitive :

les services Web sont choisis et composés uniquement à partir des types de données de leurs entrées et sorties. Pour avoir un état de l'art complet de la composition dynamique par planification, nous pouvons nous reporter aux travaux de Peer publiés en 2005 [43], mais nous allons donner dans un aperçu des principales pistes de la composition de services Web par planification.

2.2.2.3 Composition par planification

Planification « classique » : De nombreux travaux ont été produits ces dernières années. Les travaux de Sheshagiri et al. [47] proposent l'utilisation d'ontologies de domaines afin

d'ajouter des contraintes pour optimiser la recherche dans un espace de plans. Ceux de McDermott [36] proposent d'utiliser la planification par régression estimée (Estimated-Regression Planning) qui permet d'utiliser des heuristiques pour guider la recherche dans un espace de situations. Le principe est le suivant : un agent qui désire interagir avec un service Web va construire un plan puis l'exécuter. Si l'exécution échoue, l'agent aura appris de nouvelles informations de cette interaction, qui seront utilisées dans le prochain cycle de planification et exécution. Sirin et al. [48] présentent une méthode semi-automatique pour la composition de services Web. A chaque fois qu'un utilisateur doit sélectionner un service Web, tous les services disponibles qui correspondent au service sélectionné, sont présentés à l'utilisateur qui fait alors le choix à la place du système. L'idée de la composition semi-automatique est assez intéressante car il est très difficile de capturer le comportement des services Web dans leurs moindres détails, puis de les composer automatiquement. Bien que cette méthode soit simple, elle permet de voir comment un planificateur automatique et un humain peuvent travailler ensemble pour répondre à la requête de l'utilisateur. Enfin Peer [42] propose une approche basée sur le langage PDDL dans laquelle, après création du domaine de planification à partir de la description sémantique, un planificateur est choisi parmi plusieurs en fonction des instructions PDDL utilisées dans le domaine ou en fonction de la complexité du but à atteindre.

Planification basée sur les règles (Rule-based planning) : Medjahed [38] présente une technique pour générer des services composites à partir de descriptions déclaratives de haut niveau. Cette méthode utilise des règles de composabilité pour déterminer dans quelle mesure deux services sont composables. L'approche proposée se déroule en quatre phases :

1. La phase de spécification. Elle offre une spécification de haut niveau de la composition désirée en utilisant le langage CSSL (Composite Service Specification Langage).
2. La phase de correspondance. Elle utilise des règles de composabilité pour générer des plans conformes aux spécifications du service demandeur.
3. La phase de sélection. Si plus d'un plan est généré, la sélection est effectuée par rapport à des paramètres de qualité de la composition.
4. La phase de génération. Une description détaillée du service composite est automatiquement générée et présentée au demandeur.

La principale contribution de cette approche est la notion de règles de composabilité. Les règles de composabilité considèrent les propriétés syntaxiques et sémantiques des services Web. Les règles syntaxiques incluent des règles pour les types d'opérations possibles et pour les liaisons protocolaires entre les services (i.e., les bindings). Les règles sémantiques incluent des règles concernant la compatibilité des messages échangés et la compatibilité des domaines sémantiques des services, mais également des règles de qualité de la composition. Cette notion de règles de composabilité met en avant les attributs possibles des services Web qui peuvent être utilisés dans la composition de services, et peut ainsi jouer le rôle de directive pour d'autres méthodes de composition par planification.

Planification hiérarchique : Dans [52], Wu et al préconisent l'utilisation du planificateur SHOP2 pour la composition automatique de services Web à partir de leur description sémantique. SHOP2 est un planificateur HTN (Hierachical Task Network).

D'après les auteurs, le principe de décomposition d'une tâche en sous-tâches dans la planification hiérarchique est très similaire au concept de décomposition de processus composites dans OWL-S. Afin de rendre leur proposition réalisable, les auteurs font deux hypothèses concernant le modèle de processus :

– Les processus atomiques ont, soit des sorties, soit des effets, mais pas les deux en même temps.
– Les structures de contrôle Split et Split+Join ne sont pas utilisées dans les processus composites.
Sirin et Parsia [49] sont allés plus loin en intégrant un raisonneur sur les langages de descriptions dans le planificateur SHOP2. La planification HTN est très puissante pour les domaines où la connaissance est complète et détaillée, ce qui n'est pas, d'après Klush et al [33], le cas avec les services Web.

2.3 La description d'une composition de services Web

La description d'une composition de services Web doit tout d'abord contenir une description individuelle de chaque service, puis la description d'une composition elle même. La description de la composition va faire référence aux descriptions individuelles d'un service

qui appartiennent à cette composition. Actuellement, les services Web sont décrits en WSDL (Web Services Description Language). Historiquement, WSDL a été conçu en se basant sur IDL (Interface Description Language) de CORBA (Common Object Request Broker Architecture). En effet, IDL a pour but de décrire une fonction qui peut être publiée. A travers de cette description, les serveurs et les clients en CORBA peuvent interpréter les interfaces, puis les appeler. Dans le contexte des services Web, WSDL a comme objectif la description des services Web. WSDL permet en particulier de décrire les paramètres d'entrée et de sortie d'un service, ainsi que les manipulations des services au niveau des opérations [53]. Les descriptions de composition de services utilisent WSDL pour référencer les services qui appartiennent à la composition. Actuellement, plusieurs approches peuvent être utilisées pour décrire les services Web, cela dépend du problème qui va être traité sur la composition de services. De ce fait, nous nous sommes penchés sur deux langages de descriptions de composition. Nous avons, tout d'abord, classifié les descriptions de compositions en deux grands groupes : les approches non-sémantiques et les approches sémantiques. Les approches non-sémantiques n'utilisent pas la sémantique pour décrire les compositions. Elles restent principalement utilisées par les entreprises et correspondent à une approche plus commerciale des services Web. Les approches sémantiques restent, quant à elles, plus académiques. Ces approches utilisent le Web sémantique pour décrire les compositions de services Web. Nous commençons par étudier les approches non-sémantiques, puis les approches sémantiques.

2.3.1 L'approche non-sémantique

L'intégration de services a été proposée par des entreprises. Celles-ci ont développé plusieurs technologies telles que WSCI (Web Service Choreography Interface), WSFL (Web Service Flow Language) et plus récemment BPEL4WS (Business Process Execution Language for Web Services) [54]. Cette derniére est un langage qui combine le langage XLANG proposé par Microsoft et le langage WSFL proposé par IBM [55]. BPEL4WS apparaît donc comme la résultante des approches précédentes. D'autre part, les auteurs dans [56] montrent que BPEL4WS correspond à la combinaison des deux langages WSCI et BPML (Business Process Modelling Language). Même si d'autres approches peuvent être utilisées, telles que WSMO [57], SELF-SERV [58], WSCI [18], nous nous concentrons uniquement sur BPEL4WS dans la mesure où il semble être le plus général dans la communauté des services Web.

2.3.1.1 PEL4WS

La composition de services Web utilisant le langage BPEL4WS analyse les services comme des activités et des processus. Ces dénominations sont héritées de la famille des workflows. Tous ces langages, BPEL4WS et ses antécédents, sont considérés comme des langages de flux de services Web [60]. Le langage BPEL4WS a été mis en place pour travailler spécifiquement avec les services Web. Le fait d'utiliser les services Web a pratiquement imposé BPEL4WS à utiliser XML comme langage de base. Comme nous avons mentionné, ce langage a deux prédécesseurs qui sont XLANG et WSFL. Chacun représente des types de contrôle dans BPEL4WS. XLANG représente une structure hiérarchisée avec des contrôles de constructions spécialisées, et WSFL représente les structures graphiques avec des contrôles plus standardisés basés sur les conditions jointes (joint conditions). Donc, BPEL4WS fournit l'ensemble de ces contrôles, sans distinction. En ce qui concerne les données, le langage BPEL4WS limite la manipulation des données associées au processus. Il est fortement recommandé d'utiliser des données entre les entrées et les sorties des services, c'est-à-dire, les interactions entre services. Par rapport aux transactions de longue durée, si une erreur se produit, ce langage utilise une technique de compensation. Cette compensation permet de récupérer les actions au cas ou une erreur se produirait. La compensation est une caractéristique spécifique de BPEL4WS. Elle est utilisée pour annuler une activité qui avait déjà été exécutée mais pas encore confirmée.

2.3.1.2 Les structures dans BPEL4WS

Le but de BPEL4WS est de créer un nouveau service web composant des services déjà existants. En fait, l'interface de ce service composé est un ensemble de description WSDL dont chacune représente un service web. Les rôles qui vont appartenir à une composition sont définis comme des processus abstraits. Prenons un exemple de service d'achat, ou tout acheteur et tout vendeur possède un rôle spécifique. Ces rôles sont décrits en utilisant la notion de lien entre partenaire (partner link). Nous pouvons avoir un rôle pour chaque service qui appartient à la composition et qui exécute une activité. Les services sont donc traités comme des partenaires qui jouent chacun un rôle [61]. BPEL4WS dépend directement de WSDL. Une fois que les partenaires et les rôles sont définis, il est nécessaire de déterminer les variables, qui ont pour but de représenter les valeurs d'une donnée et de maintenir l'historique

du processus en utilisant l'échange des messages. D'un autre coté, les activités déterminent les types de communication entre les applications et le monde extérieur. Il y a trois sortes d'activités : receive, invoke et reply. Supposons que ces éléments soient exécutés en séquence. Par conséquent, la requête de l'utilisateur est reçue à travers l'activité nommée receive et les services sont appelés en utilisant l'activité nommée invoke. Les services peuvent être également exécutés parallèlement en utilisant pour cela l'activité nommée flow activity. En dernier lieu, la réponse sera capturée par l'élément reply. Les éléments receive et reply sont respectivement associés aux messages de input et output dans l'élément opération. Cet élément doit être en synchronisation avec l'opération de WSDL. La ***figure 2.3*** montre les interactions entre les activités et les services. Tous les éléments expliqués au-dessus sont des activités dans BPEL4WS. Les activités receive-reply créent une combinaison telle que request-response en WSDL [62]. En effet, l'activité d'invocation est celle qui appelle le service web. Pour cette raison, en utilisant BPEL4WS, les services Web sont normalement associés et représentés comme des activités. Le code 1.1 est un exemple des activités mentionnées pour la composition d'un service Calculatrice. Ce service composé exécute une soustraction, puis une addition afin d'illustrer le fonctionnement de BPEL4WS.

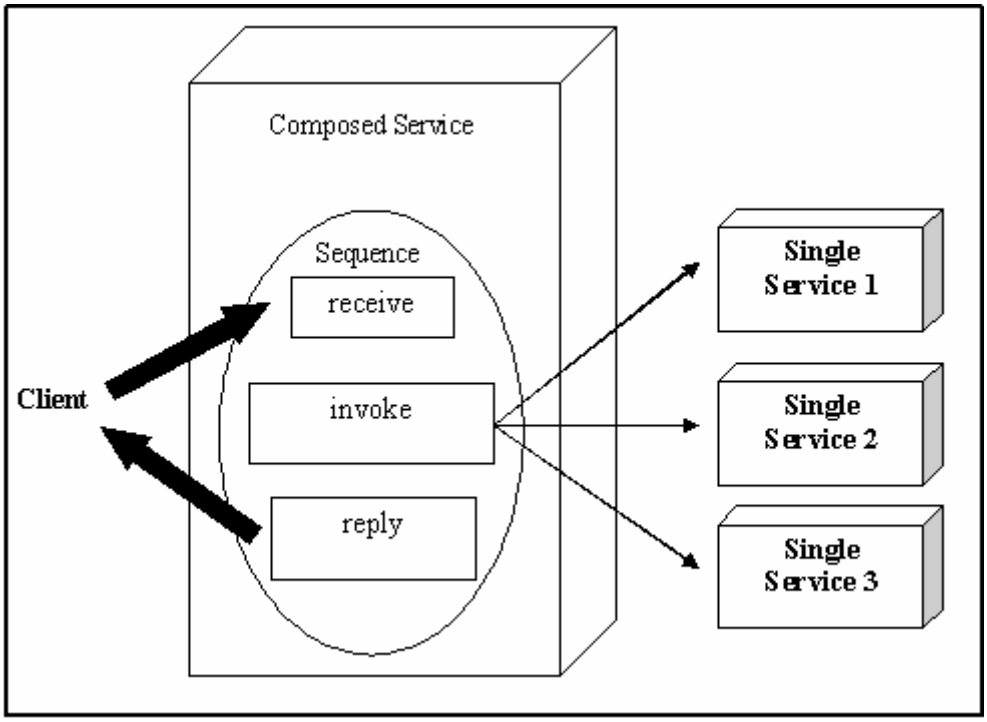

Figure 2.3 – Vision interne des activités de BPEL4WS

Code 1.1 : Un service composé en BPEL4WS

```
<partnerLinks>
      <partnerLink name="calc"
                   partnerLinkType="tns:calcPLT"
                   myRole="calculator"/>
    <partnerLink name="subt"
              partnerLinkType="ssub:subtractionPLT"
              partnerRole="subtraction"/>
     <partnerLink name="addit"
                   partnerLinkType="sadd:additionPLT"
                   partnerRole="addition"/>
</partnerLinks>
<sequence name="CalcSequence">
    <receive partnerLink="calc" portType="tns:calcPT"
              operation="calcule" variable="request"
              createInstance="yes"/>

    <invoke name="invokeSubtraction" partnerLink="subt"
            portType="ssub:Subtraction"
            operation="subtract"
            inputVariable="request"
            outputVariable="subtractReturn">
    </invoke>

    <assign name="partesParaTotal">
     <copy>
        <from variable="subtractReturn" part="subtractReturn"/>
        <to variable="request2" part="a"/>
      </copy>
    </assign>

    <invoke name="invokeAddition" partnerLink="addit"
            portType="sadd:Addition"
            operation="add"
            inputVariable="request2"
            outputVariable="addReturn">
     </invoke>

     <reply partnerLink="calc" portType="tns:calcPT"
             operation="calcule" variable="addReturn"/>
</sequence>
```

2.3.2 L'approche sémantique

L'approche sémantique garantit une description non ambiguë de la composition de services web. Cela minimise les problèmes de récupération de services incorrects, c'est-à-dire, ceux qui ne correspondent pas à la requête des utilisateurs [21]. Le Web sémantique a été proposée par Tim Berners-Lee [22]. Ce dernier a une vision d'Internet plus compréhensible et interprétable par des machines. En outre, il propose aussi qu'Internet possède un contexte coopératif [59]. Le Web sémantique est une fa¸con d'organiser, de classifier et de faire des inférences sur les données qui sont disponibles sur Internet. Grâce `a la sémantique, il est possible de manipuler des informations via un agent, comme par exemple, une machine ou une personne. Actuellement, plusieurs informations sont sur Internet et parfois il n'est pas possible de trouver des informations voulues, même si un moteur de recherche est utilisé. La principale cause est que les moteurs de recherche actuels sont basés sur les correspondances syntaxiques qui ne prennent pas en compte les significations des mots. Supposons que nous cherchions sur Internet le prix d'un produit : utilisons pour cela le mot "prix" et la valeur "12.00 ", et imaginons que le site a été décrit en utilisant le code XML suivant :
<prix>12.00</prix>

Le moteur de recherche va nous apporter tous les sites qui contiennent le mot clé "prix" et/ou la valeur "12.00 ". Cependant, un autre site qui aurait déclaré un service identique mais en utilisant le mot "coût", ne pourrait pas être considéré par le moteur de recherche.

<coût>12.00</coût>

Autant, qu'il est facile pour un être humain de voir si deux services peuvent correspondre, autant qu'il est impossible pour une machine, de faire ce type d'association. En fin de compte, une machine ne connaît pas la signification des mots sauf si elle utilise le Web sémantique. En fait, le Web sémantique peut être envisagé comme une extension d'Internet actuel, en proposant plus d'informations compréhensibles pour les êtres humains. Ces informations seront notamment plus faciles à récupérer automatiquement par des logiciels ou des machines [63].

2.3.2.1 OWL-S (Ontology Web Language for Services)

La composition de services web, en ce qui concerne la sémantique, est réalisée en utilisant une ontologie de services. Cette ontologie est appelée OWL-S, S pour services web. Elle dérive d'OWL, ce qui signifie qu'elle est également construite en utilisant des ressources et des concepts hiérarchisés. De plus, OWL-S se base sur WSDL. En particulier, OWL-S utilise WSDL comme point de communication avec le service web. Ainsi OWL-S ne connaît pas les localisations physiques de services ; ceci est géré par WSDL. Comme nous l'avons déjà expliqué, WSDL contient la description du service et utilise SOAP pour la communication. L'utilisation d'agents ou de logiciels pour trouver les services web automatique est désormais possible en utilisant cette description sémantique. Une des principales motivations de la création d'OWL-S reste la possibilité de découvrir les services web d'une manière automatique. En effet, un logiciel, basé sur les agents, interprète la description et peut ensuite découvrir les entrées et les sorties demandées par le service. Une fois les entrées et les sorties découvertes, le service peut alors être exécuté automatiquement. Donc, la découverte de services est fortement associée à ses descriptions [21]. OWL-S propose également une façon de décrire le processus de la composition de services. Grâce à ce processus, il est possible de déterminer si le service est un service composé de plusieurs autres ou si c'est un service atomique, c'est-à-dire, s'il n'a qu'une seule fonction, comme expliqué auparavant. Sur la perspective d'un utilisateur, même les services composés apparaîtront comme un unique service.

2.3.2.2 Les structures dans OWL-S

OWL-S contient trois autres ontologies appelées : Profile, Model et Grounding ; qui sont utilisées pour décrire différents aspects d'un service.

L'ontologie Profile

Le Profile décrit les services qui sont offerts par un provider et qui vont être ensuite demandés par un client. Les informations contenues dans le Profile peuvent être reparties en trois catégories :

- l'organisation qui fournit le service, c'est-à-dire, l'entreprise qui le publie et le commercialise sur Internet.

Code 1.2 : L'utilisation de l'ontologie *Profile*

```
<profile:Profile rdf:ID="CalculatorProfile">
   <service:isPresentedBy rdf:resource="#CalculatorService"/>

   <profile:serviceName>Calculator</profile:serviceName>
   <profile:textDescription>Calculate the addition, substraction
                          and multiplication of A and B.
    </profile:textDescription>

   <profile:hasInput rdf:resource="#a"/>
   <profile:hasInput rdf:resource="#b"/>
   <profile:hasOutput rdf:resource="#c"/>
</profile:Profile>
```

- la fonction calculée par le service.
- et des caractéristiques du service.

La description des fonctionnalités d'un service est indispensable au Profile. Le Profile représente deux aspects d'une fonctionnalité d'un service : la transformation d'une information (les inputs et les outputs) et les changements d'état après l'exécution de ce service. Après avoir déterminé les fonctionnalités d'un service, d'autres caractéristiques sont aussi importantes pour que le service soit découvert. Il y a en particulier, le Profile qui écrit les IOPE (Input, Output, Preconditions, Effects) d'un service. Le code 1.2 présente un exemple d'utilisation du Profile. Il s'agit d'un service correspondant a une calculatrice, service composé de trois autres services : addition, soustraction et multiplication.

L'ontologie Model

Le Model représente le comportement d'un service composé. Cette ontologie contient une classe Process qui détermine l'utilisation du service comme un processus. Elle permet également de déterminer les entrées, les sorties, les préconditions et les effets. Les entrées et les sorties représentent les transformations des données. Les entrées sont les données nécessaires pour exécuter ce service. Les sorties correspondent aux sorties de cette transformation. Les entrées peuvent être les sorties d'un service précèdent. Les préconditions et les effets représentent les changements d'état dus à l'exécution du service. Les préconditions définissent les conditions qu'il faut satisfaire avant l'exécution du service. En

ce qui concerne le type de processus, il existe trois catégories : atomique, simple et composé. Le processus atomique représente un service qui est directement appelé. Il a une unique appellation et il est directement lié avec le grounding. Le service simple est considéré comme un service atomique qui ne possède pas de liaison avec le grounding. Même si un service simple est considéré conceptuellement comme atomique, il ne peut pas être appelé directement. En effet, il aura besoin d'un service atomique pour fournir un résultat. Dans ces conditions, un service simple a toujours besoin d'un service atomique en deuxième plan pour fonctionner.

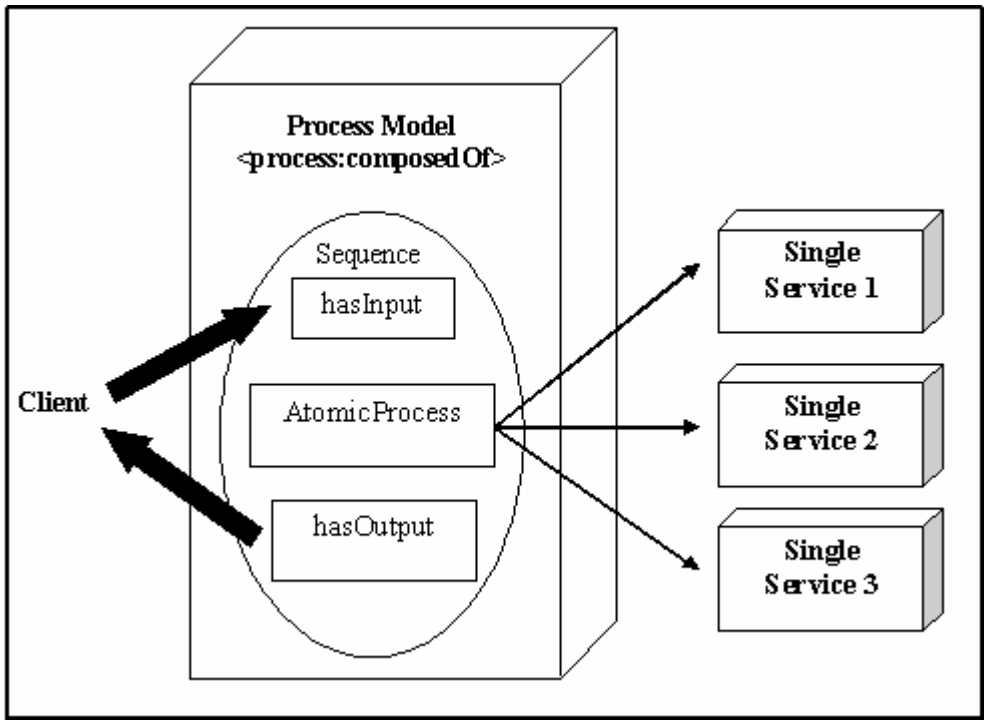

Figure 2.4 – l'ontologie de services Web

Comme le montre la ***figure 2.4***, le processus composé peut se scinder en d'autres services atomiques ou composés. Cette décomposition est spécifiée par les contrôles de construction, tels que sequence ou split (parallèle).

L'ontologie Grounding

Le Grounding permet de déterminer comment accéder au service publié. Pour cela, il fournit les détails pour connecter la spécification abstraite et la spécification concrète. Il est

effectivement nécessaire d'avoir une combinaison entre le grounding et WSDL. Le fonctionnement du grounding est assuré si et seulement si, les deux entités (WSDL et Grounding) sont présentées et se correspondent. Normalement, les concepts de grounding sont compatibles aux concepts de binding du WSDL.

2.4 Conclusion

Les services Web, de plus en plus utilisés dans le monde industriel, posent de nouveaux problèmes : comment composer ces services ? Nous avons vu qu'il existe deux grands courants de recherche. Certains proposent de composer les services en utilisant des techniques de génie logiciel : c'est l'orchestration et la chorégraphie. D'autres envisagent une approche plus dynamique : la composition de services en utilisant des techniques d'intelligence artificielle : calcul situationnel, système multi-agent ou encore planification.

Nous avons présenté en suite deux langages de description des compositions de services web. D'abord le langage BPEL4WS, dont nous avons fourni plusieurs caractéristiques, ainsi que quelques procédures d'utilisations. Nous l'avons classifié ce dernier comme un langage non sémantique pour composer les services. Nous avons ensuite, abordé le groupe de langages sémantiques, appliqués plus particulièrement dans la sémantique web. En dernier lieu, nous avons évoqué OWL-S, une ontologie appliquée aux services web.

Chapitre 3

Les langages de politiques dans les services Web.

Chapitre 3

Les langages de politiques dans les services Web

3.1 Introduction

Les politiques peuvent être spécifiées par différentes manières. De multiples approches ont été proposées dans des domaines d'application différents. Il y a, cependant, quelques exigences générales que toute représentation de politique devrait satisfaire sans se soucier de son domaine d'applicabilité **:**

- Expressivité pour manipuler un choix de conditions de politiques surgissant dans le système étant contrôlé.
- Simplicité pour faciliter les tâches de définition de politique pour des administrateurs avec les différents degrés d'expertise.
- Applicabilité pour assurer le mapping de spécifications de politique en politiques qui peuvent être implémentées pour plusieurs plates-formes.
- Scalabilté pour assurer la performance adéquate.
- Analysabilité pour autoriser le raisonnement au sujet de politiques.

L'industrie manque actuellement d'un Framework pour l'amélioration des standards de services Web en se basant sur les politiques. Deux types d'approches, *le Web sémantique et les combinaisons booléennes de prédicats de politique* pourraient aider à compenser cette lacune [67]. Les approches du Web sémantique, telles que *KAoS* [68], *Rei* [69] et *ponder* n'ont pas recueilli d'importants soutiens de l'industrie à ce jour, même si elles sont

importantes dans le traitement de certains types de politiques [67]. D'autres propositions dans le cadre de l'approche de combinaison Booléenne de prédicats de politique, paraient avoir le soutien de l'industrie [67], comme le *WS-policy framework* et le *WSPL* (Web services policy language).

3.2 Les langages basés sur l'approche Web sémantique

La politique dirigée par le Web sémantique pourrait concerner la gestion de haut niveau, à savoir, les politiques régulatrices de bas niveau d'implémentation et les politiques du niveau d'application. Une politique dirigée par le Web sémantique est aussi utile quand deux services Web doivent former un service Web composite, une fois leurs hétérogénéités sémantiques sont résolues. Les Politiques conduites par une approche du Web sémantique ne ferait que déterminer quelles règles et options que chaque service Web utilisera pour quelle transaction.

Les trois langages de l'approche Web sémantique que nous allons voir sont :

- ***KAoS*** : utilise *DAML* comme une base pour la représentation et le raisonnement au sujet de politiques dans les services Web, le calcule parallèle et dans les plates-formes du système multi agent [71, 72, 73, 74]. *KAoS* exploite aussi des ontologies pour représenter et raisonner au sujet de domaines qui décrivent des organisations d'être humain, agent, et autres acteurs informatique.

- ***Rei*** : est un nouveau langage de politique *deontic* basée sur la logique qui est fondée dans une représentation sémantique de politiques dans *RDF-S,* bien que les auteurs se progressent vers une implémentation *OWL* [75, 76].

- ***Ponder*** : est un langage de politique orienté objet pour la gestion des systèmes distribués et des réseaux [70, 77]. Les développeurs de *Ponder* innovent beaucoup de concepts de gestion de la politique utilisés dans *KAoS* et *Rei*, bien que son implémentation diffère sur plusieurs niveaux [78].

3.2.1 KAoS

KAoS est une collection de services d'agent compatible avec plusieurs framework d'agent populaires, y compris *Nomades* [106], *la DARPA CoABS Grid* [19], le *DARPA LP/Ultra*Log Cougaar* framework[1], *CORBA*[2] et *Voyager*[3]. Pendant, qu'initialement orienté aux exigences dynamiques et complexes d'applications d'agent logiciel, les services KAoS sont aussi adaptés d'un but général au Grid computing[4] et aux services Web[5] [72, 73].

- ***KAoS*** *domain services* : fournit des capacités pour les groupes de composants logiciels, personnes, ressources et autres entités pour ; être structuré dans une organisation de domaines et sous domaines pour faciliter la collaboration entre agent (agent-agent) et une administration externe de politique.

- ***KAoS*** *policy services* : tient compte de la spécification, la gestion, la résolution de conflits, et la mise en application de politiques dans les domaines. Les politiques sont représentées dans *DAML+OIL* comme des ontologies. *KAoS Policy Ontologies (KPO)* distingue entre les *autorisations* (contraintes qui autorisent ou interdisent quelques actions) et *obligations* (contraintes qui exigent que quelques actions soient exécutées, ou autrement servent pour renoncer (écarter) à une telle exigence).

Quelques caractéristiques importantes de *KAoS* [80] :

i. l'approche ne suppose pas que le système gouverné par des politiques est composé d'un ensemble homogène de composants qui ont été conçus en avance pour travailler avec les services ***KAoS***. Plutôt, le but est d'être capable d'avoir des services *KAoS* qui travaillent avec les composants arbitrairement écrits à posteriori à travers l'existence d'un support ajouté de façon transparente au niveau de la plate-forme.
ii. dans la mesure du possible, le Framework *KAoS* supporte les modifications dynamiques de politiques en temps d'exécution *(supports dynamic runtime policy changes)* et pas de configurations statiques déterminées à l'avance.

[1] http://www.cougaar.net
[2] http://www.omg.org
[3] http://www.recursionsw.com/osi.asp
[4] http://www.gridforum.org
[5] http://www.w3.org/2002/ws/

iii. Le Framework est extensible à une variété de plates-formes d'exécution qui peuvent être simultanément exécutées avec l'exécution (l'application) de différents mécanismes – en principe toute plate-forme pour laquelle les mécanismes de la mise en application de la politique peuvent être écrits.

iv. le Framework *KAoS* est déterminé à être robuste et adaptable pour continuer la gestion et la mise en application de la politique face aux attaques ou échecs de toutes combinaisons de composants.

v. *KAoS* adresse le besoin pour faciliter l'utilisation des outils d'administration, basés sur les politiques capable de contenir le domaine de connaissance et l'abstractions conceptuelles ; cela a laissé les concepteurs d'application se concentrer plus sur de haut niveau de politique que sur les détails de la mise en œuvre (implémentation).

Les outils exigent une interface graphique sophistiquée d'utilisateur pour le contrôle, en visualisant et modifier dynamiquement la politique en temps d'exécution (runtime).

3.2.1.1 Représentation de la politique

Dans *KAoS Policy Ontologies (KPO : KAoS Policy Ontologies)*, une politique est une instance d'un type de politique approprié, cela définit les valeurs associées pour ses propriétés, tel que le site de mise en application, la priorité et le *time stamp* (noter la date et l'heure) de la mise à jour. À travers plusieurs restrictions de propriété dans le type d'action, le contexte spécifique pour l'action est défini et une politique donné peut être diversement ou différemment *scoped*. La version courante de *KPO* consiste en *79* classes et *41* propriétés qui, ensemble définissent des ontologies de base pour les actions, les acteurs, les groupes, les places, les différentes entités reliées aux actions et les politiques. Pendant que l'application s'exécute, les classes et les individus qui correspondent aux nouvelles politiques et aux entités d'application sont aussi ajoutés de façon transparente et supprimés si nécessaire.

Le code 3.1 illustre un fragment d'un exemple de politique de communication qui déclare que les membres d'un domaine appelé ***A*** est autorisé à communiquer en dehors de son domaine, en utilisant une communication codée ou cryptée :

Code 3.1 : Exemple DAML de représentation de politique dans KAoS.

```
<daml:Class rdf:ID="ExampleAction">
      <rdfs:subClassOf rdf:resource="#EncryptedCommunicationAction" />
      <rdfs:subClassOf>
            <daml:Restriction>
                  <daml:onProperty rdf:resource="#performedBy" />
                  <daml:toClass rdf:resource="#MembersOfDomainA" />
            </daml:Restriction>
       </rdfs:subClassOf>
       <rdfs:subClassOf>
            <daml:Restriction>
                 <daml:onProperty rdf:resource="#hasDestination" />
                 <daml:toClass rdf:resource="#notMembersOfDomainA " />
            </daml:Restriction>
        </rdfs:subClassOf>
   </daml:Class>
<policy:PosAuthorizationPolicy rdf:ID="Example">
     <policy:controls rdf:resource="#ExampleAction" />
     <policy:hasSiteOfEnforcement rdf:resource="#ActorSite" />
     <policy:hasPriority>10</policy:hasPriority>
     <policy:hasUpdateTimeStamp>4237445645589</policy:hasUpdateTimeStamp>
</policy:NegAuthorizationPolicy>
```

3.2.1.2 Caractéristiques de gestion de politique

KAoS fournit un outil graphique appelé l'outil d'administration de politique de *KAoS (KPAT),* qui aide les utilisateurs dans la spécification de politique, la révision et l'application. Les politiques, les domaines et les entités d'application sont définis en utilisant la *KPAT GUI,* les représentations *DAML* appropriées sont générées automatiquement dans le background.

KAoS détecte des conflits potentiels entre politiques lors de la spécification, chaque fois qu'un utilisateur essaie d'ajouter une nouvelle politique au Directory service. L'algorithme *KAoS* pour la découverte des conflits de politique, est capable de détecter des conflits entre politiques en comptant sur les algorithmes implémentés comme extensions pour *Stanford's Java Theorem Prover* (JTP) qui est intégré dans *KAoS*.

Le moteur identifie les conflits de politiques en utilisant les mécanismes du subsumption entre classes et essaie de résoudre ces conflits en ordonnant les politiques en fonction de leur précédence [73, 74].

KAoS Framework est responsable de la gestion de domaines d'agents et assure la consistance de politique à tous les niveaux de la hiérarchie du domaine.

Le Directory Service est responsable de la gestion de politique globale ou générale, pendant que les protecteurs (***guards***) interprètent les politiques et les transmettent aux Exécuteurs qui sont des composants d'une plateforme spécifique, qui traite la mise en application de la politique réelle.

Avant d'exécuter leurs fonctions, pour permettre ou empêcher une action donné, les Exécuteurs de politiques d'autorisation doivent obtenir une réponse à la question suivante *" est-ce qu'une action donné est autorisée ou pas ?"* En raisons de performance et de robustesse, *KAoS* répond localement à la question de l'autorisation dans un Exécuteur ou dans un module intermédiaire dès que possible.

Les représentations basées sur des paramètres simples des politiques appropriées sont stockées localement dans le ***Guard*** et utilisé dans la réponse aux questions d'autorisation.

Une description d'action est créée par l'Exécuteur et passée au *Guard,* qui traverse son stockage de politique et contrôle pour voir si l'instance de l'action donnée est dans le domaine d'actions contrôlé par l'une de ses politiques. Si le mécanisme de l'autorisation ne trouve aucune politique applicable à la description de l'action passée vers lui, il répond à la question logique avec m'importe quelle modalité d'autorisation par défaut.

Les modalités de l'autorisation par défaut sont actuellement configurées sur chaque domaine de base. Les défauts correspondent soit à une *démocratie* où tout est autorisé et qui n'est pas interdit explicitement, ou à une *tyrannie* où tout est interdit et qui n'est pas autorisé explicitement.

3.2.2 Rei

Rei est un *Framework* de politique qui intègre le support pour la spécification de politique, l'analyse et le raisonnement dans les applications informatique en générale [75, 76]. Son langage de politique *deontic-logique-based* permet à des utilisateurs d'exprimer et

représenter les concepts *des droits (rights), prohibitions, obligations et dispenses (permissions).* Ces concepts correspondent respectivement, aux conditions de l'autorisation positive et négative et de l'engagement positif et négatif dans KAoS et Ponder. En plus, *Rei* permet à des utilisateurs de spécifier les politiques qui sont définies comme règles associant une entité d'un domaine contrôlé ou gérer avec un ensemble de droits, de prohibitions, obligation (engagements) et de dispenses.

Rei se fonde sur une ontologie d'application indépendante pour représenter les concepts des droits, des prohibitions, des obligations, des dispenses et des règles de politique. Ceci permet aux différents éléments d'un environnement répandu de comprendre et d'interpréter les politiques de *Rei* d'une manière correcte. L'ontologie de base inclut également la description des actions. Une action générale est décrite par son identificateur unique d'action, les objets de cible sur lesquels l'action peut être effectuée ou exécutée, un ensemble de conditions préalables qui doivent être vraies avant que l'action puisse être effectuée et les effets qui résultent de l'action quand elle est exécutée.

Comme *KAoS*, *Rei* permet à des utilisateurs d'enrichir l'ontologie de base avec d'autres ontologies dépendantes du domaine pour exprimer les concepts et les ressources qui sont particuliers à certains domaines.

Le langage *Rei* fournit également des modèles pour spécifier des activités de discussion et des méta-politiques. *Les premiers* sont utilisés pour échanger dynamiquement des droits et des obligations entre les entités, *alors que les derniers* sont utilisés pour résoudre les conflits de politique qui sont automatiquement détectés par le moteur de politique de *Rei*.

3.2.2.1 Spécifications de politique

Les concepts de *Rei* de droit, de permission, d'engagement, de dispense et des règles de politique sont représentés comme des prédicats de prolog. Aucun ***GUI*** équivalent de *KPAT* n'est fourni. La *figure 3.1* illustre quelques exemples des droits de ***Rei*** et des spécifications de politique. La ligne A décrit *un objet de politique* qui est utilisé pour modeler *des droits, des permissions, des obligations et des dispenses* pour effectuer une action spécifique dans certaines conditions.

A	*PolicyObject (Action , Conditions)*	right (examineStudent , professor (Var))
B	*has (Subject , Policy Object)*	has (Var , right (examineStudent, professor (Var))
C	*action (ActionName, TargetObjects, Pre-Conditions, Effects)*	action (examineStudent, Y, [student(Y)] , [])

Figure 3.1 – Exemple de spécification de politique avec *Rei*

Le Rei permet également à des utilisateurs de spécifier des politiques de contrôle d'accès basées sur le rôle ou des politiques reliant non seulement aux individus mais également aux groupes d'entités. Par exemple, dans la ligne B la *variable var* peut être résolue par un rôle ou un ensemble d'entités basé par groupe qui résolvent l'état de l'objet de politique, c.-à-d. par tous les membres *du rôle* de professeur ou groupe. Ainsi, avec la même structure de politique il est possible de spécifier les politiques pour les différentes entités, le groupe d'entités et les rôles ou n'importe quelle combinaison des trois. Cependant, le rôle n'est pas un concept spécifique du langage et il n'est pas représenté dans l'ontologie de base de *Rei*. Ceci signifie que les permissions de rôle et les devoirs ne sont pas groupées ensemble dans un seul prédicat, de ce fait compliquer la gestion de rôle.

3.2.2.2 Modèle de déploiement de politique

Le framework Rei fournit un moteur de politique qui fait le raisonnement au sujet des spécifications de politique. Le moteur accepte des spécifications de politique dans le langage *Rei* et dans *RDF-S,* conforme (cohérent) à l'ontologie de *Rei*. Spécifiquement, le moteur traduit automatiquement les spécifications *RDF* (Resource Description Framework*)* en triplet de la forme (sujet, prédicat, objet). Le moteur accepte également l'information additionnelle dépendante du domaine dans n'importe quel langage sémantique qui peut alors être converti en cette forme reconnaissable de triplet. Le moteur est conforme et complet [75] et permet des requête en fonction du langage ***prolog*** au sujet de n'importe quelles politiques, méta-politiques et la connaissance dépendante du domaine qui ont été chargées dans sa base de connaissance.

Le moteur de politique de *Rei* peut détecter des conflits de modalité entre les politiques. En particulier, le moteur repère deux politiques en conflit si elles sont des modalités

incompatibles (en conflit) et s'il y a un chevauchement dans le sujet, la cible et l'action. Afin de résoudre les conflits détectés, *Rei* fournit la possibilité de spécifier les méta-politiques qui permettent de définir des priorités sur des politiques, et/ou pour placer des relations de priorité ou de précédence entre les modalités de politique. Le *framework Rei* ne fournit pas un modèle d'application. En fait, le moteur de politique n'a pas été conçu pour exécuter les politiques mais pour raisonner seulement au sujet d'elles et pour répondre aux requêtes. Par exemple, le moteur peut indiquer si une entité a le droit ou l'obligation d'effectuer une certaine action, mais alors elle n'inclut pas des mécanismes pour assurer l'application ou l'exécution de la politique, par exemple, interdisant le sujet de politique d'effectuer des actions non autorisées ou en forçant l'entité à exécuter des actions exigées. Ainsi il ne fournit aucune protection contre les composants ou les agents malveillants ou non conformes.

3.2.3 Ponder

Ponder est un langage orienté objet déclaratif, qui supporte les spécifications de plusieurs types de politiques des systèmes d'objet répartis et fournit des techniques structurantes pour des politiques à fin de couvrir la complexité de l'administration de politique dans les grands systèmes d'information d'entreprise [70, 78, 79].

Ponder distingue les politiques de base et les politiques composites. *Une politique de base* est considérée comme une règle qui gouverne les choix dans le comportement de système et elle est spécifiée par une déclaration entre un ensemble *de subjects (sujets)* et un ensemble *de targets (cibles).* Ces ensembles sont utilisés pour définir les objets gérés sur lesquels la politique opère. *Ponder* utilise le terme subject (*sujet*) pour faire référence aux utilisateurs *principaux (managers)* ou les composants automatisés du manager (administrateur), qui ont la responsabilité de gestion, c.-à-d., ont l'autorité de lancer une décision de gestion. Un sujet (subject) peut opérer sur les objets de la cible (target) (les ressources ou les fournisseurs de service), avec l'invocation des méthodes visibles sur l'interface de la cible. Les types de politique fondamentaux dans *Ponder* sont des obligations et des autorisations. Par exemple, *les politiques d'obligations* définissent *" les actions que les sujets (subjects) de politique doivent effectuer sur des entités de cible quand les événements spécifiques appropriés se produisent ",* tandis que les politiques d'autorisation définissent *" quelles sont les opérations où un tel sujet est autorisé à faire sur les objets de la cible* " [79]. Les politiques composites

permettent aux politiques de base concernant les unités d'organisation, d'être groupées. Les exemples des politiques composites sont *des politiques* de *rôles et* de *relations*. Dans *Ponder*, les rôles sont des groupes de politiques gouvernant le comportement du même sujet en spécifiant ses droits et devoirs, tandis que les relations groupent les politiques définissant les droits et devoirs des rôles vers autres.

Les politiques de *Ponder* se fondent sur le concept principal des domaines de gestion. Les domaines fournissent des méthodes de groupement des objets auxquels les politiques peuvent être utilisées pour partitionner les objets dans un grand système, comme il est préféré. Les sujets (subjects) et les cibles (targets) de politique de *Ponder* sont définis en terme d'expressions étendue de domaine qui permettent à la combinaison des domaines de former un ensemble composite d'objets., mais d'identifier également un seul objet nommé dans les domaines.

3.2.3.1 Spécifications de politique

La figure 3.2 illustre un exemple de la politique d'autorisation de *Ponder*. La politique indique que les principaux professeurs ont accès à la lecture de tous les dossiers d'exercices de leurs étudiants seulement pendant les heures d'ouverture de l'école, c.-à-d. de 7 a.m à 7 P.m. et du lundi au vendredi. La définition de type politique introduit un nouveau type de politique défini par l'utilisateur, dont une ou plusieurs instances de politique de ce type peuvent être créés. Des types de politique peuvent également être paramétrées et les instances créées avec des paramètres spécifiques de contexte, par exemple, les ensembles de sujet et de cible peuvent être passés en tant que paramètres réels.

```
type auth+ FileAccess (subject professor,
                       target   exerciseFiles )
   {
     action   read;
     when
          Time.between(0700, 1900) and
          Time.between('mon', 'fri');
   }
inst auth+ P1 = FileAccess ("professor/Green",
                    "NodeServer/StudentFiles");
```

Figure 3.2 – La spécification de politique d'autorisation avec Ponder

La déclaration d'instance de politique crée une instance d'un type défini par l'utilisateur de politique. ***P1*** est une instance de politique qui autorise le professeur principal appelé ***Green*** à accéder aux dossiers d'exercice d'étudiant hébergés sur le ***NodeServer.*** L'action de lecture est une méthode d'interface de la cible et ne peut être ni raffinée, ni spécialisée. Quand la clause est utilisée pour nier ou refuser l'accès de la lecture aux dossiers pendant les heures de fermeture d'école. *Ponder* ne supporte pas actuellement les spécifications des règles par défaut pour les politiques. Par exemple, si une politique d'autorisation n'est pas spécifiée pour une action, la contrainte par défaut sur le comportement (i.e. si permettre ou interdire l'action) est une implémentation dépendante.

3.2.3.2 Modèle de déploiement de politique

Ponder fournit de divers outils graphiques pour l'édition (l'écriture), la mise à jour, la suppression, et la navigation. Il y a également des outils pour l'analyse syntactique et sémantique des spécifications de politique et pour transformer directement des spécifications du langage *Ponder* en code *XML* ou Java, qui peuvent être interprétés dans la phase d'exécution. En outre, les développeurs de *Ponder* ont implémenté un outil prototype de détection de conflit pour détecter des chevauchements et des conflits entre les politiques. L'outil distingue entre les *conflits de modalité* et les *conflits spécifiques* à l'application [77].

Semblable à *KAoS* et à *Rei,* d'abord par les inconsistances dans les spécifications de politique qui peuvent surgir entre les politiques avec des modalités de signes opposés qui se réfère aux mêmes sujets (subjects), cibles et actions *(par exemple, conflits entre les permissions et les prohibitions ou entre les obligations et les prohibitions).* Cependant, le manque d'une ontologie limite sa capacité de traiter des sujets, actions et cibles au niveau des variables d'abstraction. Et en fin par les inconsistances entre le contenu de politique et les critères externes *(par exemple conflit entre une obligation d'accéder à une ressource et à une limitation sur la disponibilité de ressource).*

Des travaux plus récents sur l'analyse de politique dans *Ponder* se sont concentrés sur l'utilisation du *raisonnement abductive* ensemble avec une représentation de calcul d'événement des politiques pour identifier et détecter les conflits de politiques.

Ponder fournit un modèle de déploiement complet. À la fin de la spécification de politique, la politique est compilée par le compilateur de *Ponder* dans une classe de *Java* et puis représentée dans la phase d'exécution par un *objet Java*. Les changements de la phase d'exécution à la politique ne sont pas possibles. Le modèle de distribution et d'application distingue entre les politiques d'autorisations et d'obligations. Les objets de politique d'autorisation sont distribués plus près des objets de cible de la politique où un agent de contrôleur d'accès fournit l'interprétation de leur phase d'exécution et d'application en permettant ou en rejetant des requêtes d'accès aux ressources commandées de cible. Les objets de politique d'obligation sont distribués plus près des objets de sujets de politique où un agent de gestion de politique fournit leur interprétation de la phase d'exécution et d'application dans l'occurrence d'événement de la politique appropriée.

Ponder fournit les spécifications des interfaces pour les agents d'application, c.-à-d., pour le contrôleur d'accès et l'agent de gestion de politique, mais aucune implémentation n'est donnée. Il y a quelques systèmes qui implémentent le modèle de déploiement de politique de *Ponder* dans différents domaines d'application**.**

3.2.4 Etude comparative

Exemple : Communication De Contrôle [80]

Considérons le cas d'un système multi agent modelant un institut académique où il y a la nécessité de gouverner la communication entre les professeurs et leurs étudiants, tous les deux sont représentés comme des agents. Dans cette désignation, une politique pourrait déclarer que *des professeurs sont autorisés de communiquer la catégorie ou le grade d'examen final à leurs étudiants en utilisant une communication chiffrée (cryptée) seulement après l'approbation du directeur de l'institut*

3.2.4.1 Représentation de politique

KAoS, Rei et *Ponder* représentent un exemple de politique d'une manière différente. *Le code 3.2* illustre la définition de politique *KAoS* dans la représentation *DAML*. La

politique est une instance d'un type positif de politique d'autorisation avec des propriétés associées et réglées aux valeurs désirées.

Code 3.2 : *Un fragment de la spécification de politique dans KAoS*

```
<?xml version='1.0'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
         xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
         xmlns:policy="http://ontology.coginst.uwf.edu/Policy.daml#"
         xmlns="http://ontology.coginst.uwf.edu/ExamplePolicies/PolicyExample.daml#">
<daml:Ontology rdf:about="">
    .....
<daml:Class rdf:ID="ExaminationGradePolicyAction">
    <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class
    rdf:about="http://ontology.coginst.uwf.edu/Action.daml#EncryptedCommunicationAction"/>
     <daml:Restriction>
     <daml:onProperty rdf:resource="http://ontology.coginst.uwf.edu/Action.daml#performedBy"/>
     <daml:toClass
   rdf:resource="http://ontology.coginst.uwf.edu/Examples/ActorClasses.daml#AgentProfessors"/>
      </daml:Restriction>
      <daml:Restriction>
       <daml:onProperty
            rdf:resource="http://ontology.coginst.uwf.edu/Action.daml#hasDestination"/>
       <daml:toClass
   rdf:resource="http://ontology.coginst.uwf.edu/Examples/ActorClasses.daml#AgentStudents"/>
      </daml:Restriction>
       <daml:Restriction>
        <daml:onProperty
           rdf:resource="http://ontology.coginst.uwf.edu/Examples/Action.daml#hasApproval"/>
         <daml:toClass
           rdf:resource="http://ontology.coginst.uwf.edu/Examples/ActorClasses.daml#AgentInstituteDirector"/>
         </daml:Restriction>
     </daml:intersectionOf>
  </daml:Class>
  <policy:PosAuthorizationPolicy rdf:ID="ExaminationGradePolicy">
   <policy:controls rdf:resource="#ExaminationGradePolicyAction"/>
     <policy:hasSiteOfEnforcement rdf:resource="http://ontology.coginst.uwf.edu/Policy.daml#SubjectSite"/>
     <policy:hasPriority>10</policy:hasPriority>
     <policy:hasUpdateTimeStamp>446744445544</policy:hasUpdateTimeStamp>
  </policy:PosAuthorizationPolicy >
```

Par exemple, *la propriété* de *performedBy* représente l'expéditeur, sa valeur est limitée à "*AgentProfessors*", *la valeur* du *siteOfEnforcement* est placée dans le site significatif du sujet (subject) que les mécanismes d'application seront associés aux agents de professeurs qui sont le sujet de la politique, alors que la condition de l'applicabilité est décrite par *la propriété* d'approbation.

L'avantage important pour l'utilisation du KAoS, pour représenter une politique, est que n'importe quel élément de la politique (groupes, acteurs, actions, propriétés d'action, conditions d'applicabilité) est décrit par une ontologie appropriée au niveau désiré d'abstraction. Ainsi, le système peut raisonner logiquement au sujet des conflits complexe et répond aux requêtes au sujet des entités qui peuvent être définies avec des demains considérablement variables et des niveaux de description**.**

KAoS fournit un ensemble riche d'ontologies prédéfinies qui peuvent être prolongées pour s'adapter à des exigences d'application spécifiques et cacher la complexité potentielle de la représentation de politique et les langages derrière l'interface graphique d'utilisateur ***KPAT.***
La *figure 3.3* illustre la représentation de politique de communication dans *Rei*.

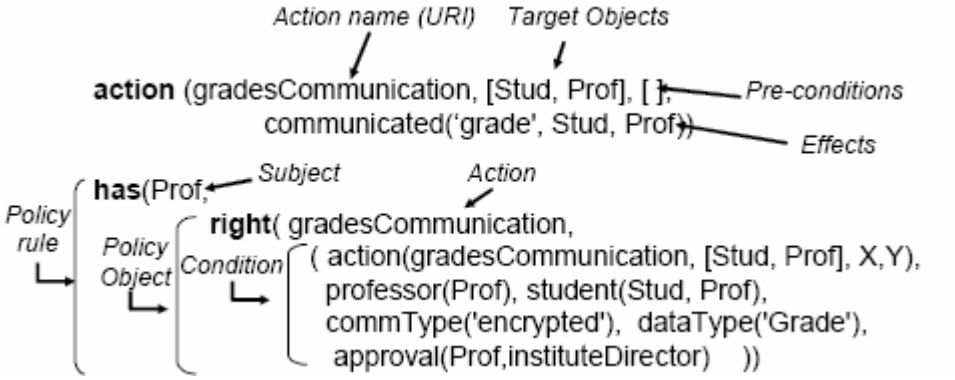

Figure 3.3 – La spécification de politique dans Rei

La politique de communication basée sur *Ponder* est illustrée sur le la *figure 3.4.* L'exploitation d'un type de politique d'autorisation de Ponder pour le mapping de la politique de communication exige l'identification de la ressource qui a besoin de protection.

```
domain prof = /SysEntities/Agents/ProfessorAgents;
domain stud = /SysEntities/Agents/StudentsAgents;

inst auth+ ExamGradeComm {
   subject  s= prof;
   target  t = /SysEntities/SysServices/CommunicationChannel;
   action t.communication ("Encrypted", data, destination) ;
   when data.getType = "Grade"
    && destination == (stud -> select (st | st.professor == s))
    && s.receivedApproval(s.getInstituteDirector()) == 'true' ;
}
```

Figure 3.4 – La spécification de politique dans Ponder

La définition de politique dans ***Ponder*** exige aux utilisateurs de savoir, à l'avance les noms spécifiques des entités à commander et de leurs interfaces.

3.2.4.2 Le raisonnement de politique

La nature déclarative de la définition de politique *KAoS* facilite l'analyse et la vérification de politique. De diverses inférences peuvent être exécutées pour une variété de buts et d'objectifs, telles que la détection et l'harmonisation des conflits de politiques, révélant les politiques relatives, ou le raisonnement au sujet de futures actions basées sur la connaissance des politiques en vigueur.
Rei se base sur le moteur de raisonnement de *prolog* pour analyser les politiques et peut raisonner au-dessus des ontologies dépendantes du domaine, en traduisant automatiquement les politiques de *RDF* dans des triplets. Dans *KAoS,* cette conversion est cachée aux utilisateurs par le théorème de Java du Stanford qui convertit la base de connaissance dans des triplets avant de raisonner sur eux et fournir seulement les résultats finaux.

Dans *Rei*, les utilisateurs ont la capacité de contrôler directement la forme intermédiaire. De plus, les utilisateurs ont les possibilités d'accéder aux instances de politique sous cette forme en soumettant quelques requêtes de *prolog* au moteur de politique.

En conclusion, *Ponder* peut détecter la modalité et les conflits spécifiques d'application dans les politiques, telles que des conflits du devoir (fonctions). Des règles de politique de ***Ponder*** sont traduites en représentation de calcul d'événement qui décrit la sémantique du langage de politique. Des techniques de raisonnement *Abductive* sont exploitées pour analyser les spécifications de politique afin d'identifier les conflits existants et pour fournir des explications sur la façon dont elles pourraient se produire.

3.2.4.3 Déploiement de politique

Dans KAoS, la représentation abstraite des actions est un avantage et un inconvénient. D'une part, cette abstraction permet à *KAoS* d'être adapté pour n'importe quelle plateforme qui a des crochets requis d'application. D'autre part, le déploiement dans n'importe quelle

plateforme exige la disponibilité d'un exécuteur (enforcer) spécifique pour la classe d'action de politique. Si un tel exécuteur n'existe pas déjà, il doit être implémenté. Sa conception est habituellement une tâche non insignifiante et exige des programmeurs ayant une compréhension profonde de la classe d'action et de son mapping pour la plateforme.

Rei, comme nous l'avons souligné, ne fournit aucun support d'application et c'est un des inconvénients principaux d'exploiter ce langage pour des applications pratiques. En outre, même si le moteur accepte les spécifications de politique basée sur l'ontologie, la gestion de politique principale est au-dessus des constructions de *Rei*.

C'est une contrainte en ce qui concerne *KAoS* qui contrôle ou gère la politique sous sa forme originale, laissant aux plateformes d'implémentation la possibilité pour choisir la représentation du temps d'exécution le plus approprié.

Dans *Ponder*, l'avantage principal de modeler des actions de politique à un niveau bas d'abstraction est qu'elles sont directement implémentables dans *Java* avec peu d'effort additionnel. C'est la raison de son adoption large dans beaucoup de systèmes existants.

3.2.4.4 Discussion

- chaque forme de représentation de politique montre le pour et le contre et le choix d'une approche devrait être conduit par les *caractéristiques du domaine d'application et par la simplicité, la lisibilité, la scalabilité d'analysabilité et les exigences d'applicabilité.*

- une ontologie simplifie la tâche d'administrer le comportement des environnements complexes.

- la possibilité de modeler des politiques à un niveau élevé d'abstraction permet à des utilisateurs de se concentrer plus sur des exigences à un niveau élevé de gestion que sur des détails d'implémentation.

- Une description basée sur l'ontologie de politique permet au système d'utiliser des concepts pour décrire les environnements et les entités étant commandées ou contrôlées.

Plusieurs possibilités peuvent bénéficier de ce dispositif puissant, tel que la détection et l'harmonisation de conflits de politiques. En outre, les approches basées sur l'ontologie simplifient l'accès à l'information de politique, avec la possibilité de calculer dynamiquement des relations entre les politiques et l'environnement, les entités ou d'autres politiques basées sur des relations d'ontologie plutôt que leurs fixation à l'avance.

Comme des bases de données, il est possible d'accéder à l'information fournie en questionnant l'ontologie selon le schéma de l'ontologie. C'est un avantage par rapport aux langages traditionnels qui fournissent seulement des requêtes prédéfinies à l'information d'accès et aux représentations statiques de la politique. Les ontologies peuvent également simplifier le partage de la connaissance de politique, augmentant de ce fait la possibilité pour les entités de négocier des politiques et de convenir à un ensemble commun de politiques. Cependant, l'adoption des ontologies pour les spécifications de politique exige l'adressage de quelques difficultés techniques.

Les interfaces graphiques peuvent aider des utilisateurs, pendant les spécifications. Par exemple, *KPAT* fournit une interface graphique pour les spécifications et la gestion de politique, mais de tels outils exigent toujours un prix dans le développement continu afin de suivre les changements cruciaux des ontologies fondamentaux.

Finalement, la spécifications de politique basée sur l'ontologie peut être difficile à implémentée à cause du niveau élevé de la spécification des politiques basées sur l'ontologie ; qui peuvent être enlevées de l'exécution concrète de l'application de politique sur les systèmes. L'espace entre les spécifications et l'exécution des politiques ne peut pas être complètement surmonté d'une façon automatisée, mais doit être résolu à un plus grand ou petit degré par les programmeurs conformés aux possibilités et aux dispositifs de chaque plateforme. Les outils et les bibliothèques de génération de codeur d'application des mécanismes d'exécution, adaptés aux plateformes spécifiques, sont parmi les dispositifs les plus importants pour que les Frameworks de gestion de politique fournissent et permettent leur implémentation répandue.

3.3 Les langages basés sur l'approche de combinaisons booléennes de prédicats de politique

Quatre langages de politique de services Web qui ont été proposés comme une base de nouveaux standards qui sont basés sur des ***combinaisons booléennes de prédicats*** ***''*** *Boolean combinations of predicates.*''. Dans le *W3C Workshop on Constraints and Capabilities for Web Services* [91], de différentes propositions d'un langage standard pour les expression de politiques dans les services Web ont été présentées. Quatre de ces langages ont présenté des variations sur les combinaisons booléennes de prédicats: le Web Services Policy Framework (WS-Policy) [81], le Web Services Description Language (*WSDL*) [82] avec l'ajout de compositeurs [83], le profil XACML pour les services Web (*WSPL*) [84] et le language outline from IONA Technologies.

Ces langages se diffèrent dans les prédicats qui sont utilisés. En WS-Policy, les prédicats sont des assertions qui renvoient un résultat booléen, mais ne sont pas autrement définis dans Policy Framework lui-même; les définitions d'assertion doivent être fournies comme une partie de chaque document spécifique du domaine qui définit les éléments à être contrôlé par une politique. Dans WSDL compositors, les prédicats sont des caractéristiques, propriétés booléennes, ou compositeur imbriqués (opérateur booléen) d'expressions *WSDL*; Les caractéristiques et les propriétés ne sont pas encore définies, Bien que certaines directives sont des données sémantiques. En *XACML WSPL*, les prédicats sont des fonctions *XACML* [85] qui retournent un résultat booléen et opèrent sur des attributs et des valeurs littérales, où un attribut peut être un treplet *name/ type/ value* ou un noeud dans un document XML identifié par une expression XPath [86]. Dans IONA outline, les prédicats sont des éléments XML simples, avec un paramètre *Yes/No* au plus, le processus de définition des éléments à être utilisés n'est pas précisé dans la proposition. Tous ces langages doivent pouvoir compter sur un mécanisme pour l'association des politiques avec les services ou avec les éléments de service. WS-Policy s'appuie sur les Web Services Policy Attachment (WS-PolicyAttachment) [87]. WSDL s'appuie sur les points de fixation définis dans le WSDL lui-même. WSPL s'appuie sur une convention pour l'utilisation de l'élément cible XACML. IONA outline ne décrit pas son mécanisme. Cette étude examinera les raisons pour lesquelles ces langages sont d'intérêt pour l'industrie et propose une abstraction pour la stratification des fonctionnalités impliquées dans ces langages.

Dans ce qui suit, nous examinons pourquoi les deux langages de politiques WSPL et WS-policy sont d'intérêt pour l'industrie et nous comparons les formes de prédicats utilisées par ces deux langages.

3.3.1 WS-Policy (IBM, Microsoft et all. 2003)

WS-Policy (*Bajaj S et al., 2004*) est un cadre de travail qui définit les structures de base pour la description et la communication des politiques des services Web. Il forme une grammaire flexible et extensible pour exprimer les besoins, les capacités et les préférences d'un service Web dans un format XML. Les politiques sont définies dans *WS-Policy* comme une collection d'« alternatives ». L'alternative est une collection d'assertions. L'assertion est l'unité de base des politiques dans *WS-Policy*. Chaque assertion appartient à un domaine spécifique. *WS-Policy* définit les domaines d'application des politiques suivants : *Security*, *Privacy*, *Application priorities*, *User account priorities* et *Traffic control*. Les assertions peuvent être regroupées ou combinées grâce aux opérateurs *<wsp:All>*, *<wsp:ExactlyOne>* et l'attribut de type *« Optional »*.

WS-Policy ne définit pas la manière dont les politiques seront interfacées avec le service Web, tandis qu'elle laisse le champ libre à d'autres spécifications pour définir la manière dont sont attachées les politiques avec les mécanismes et les ressources du service Web pour une meilleure adaptabilité. Par exemple, *WSPolicyAttachement* est une spécification qui définit l'association de *WS-Policy* avec WSDL et UDDI. L'idée principale de *WS-Policy* est d'offrir une grammaire pour décrire les politiques (préférences, capacités…) de chaque service Web et, ensuite, de pouvoir échanger ces informations et de comprendre leur sémantique. *WS-Policy* s'intègre dans des messages SOAP, celui-ci étant extensible pour supporter des nouveaux types de messages. *WS-Policy* est la base de plusieurs cadres de travail (frameworks) : *Web Services Policy Attachment*, *Web Services Reliable Messaging*, *Web Services Metadata Exchange*, *Web Services Security Policy*, etc., qui forment ensemble la métaspécification des politiques des services Web. Enfin, *WS-Policy* définit trois opérations pour le traitement des politiques : *Normalize*, *Merge* et *Intersect*. Ces opérations définissent un modèle pour compiler les politiques en normalisant, d'abord, chaque politique et ensuite les faire combiner selon un ensemble de règles.

3.3.1.1 WS-Policy Predicate Layer

Dans WS-Policy, chaque spécification d'un service Web doit définir un ensemble d' assertions de politique à être utilisé dans l'expression de prédicats de politique relative au vocabulaire défini dans la spécification. Par exemple, si la spécification du vocabulaire fondamental définit un élément du schéma XML ***<v:A>*** qui doit être contrôlé par les politiques de service Web, alors il faut un ou plusieurs éléments supplémentaires définis pour une utilisation dans l'expression de prédicats de politique relative à ***<v:A>.*** WS-SecurityPolicy [88], qui définit les prédicats d'Assertion pour être utilisés avec le vocabulaire de WS-Security [89], est l'exemple utilisé dans la spécification de WS-Policy. Chaque nouveau vocabulaire du domaine, il lui faudra son propre ensemble de prédicats d'Assertion, bien que les auteurs de WS-Policy suggèrent que, dans l'avenir, de telles Assertions seront définies comme une partie dans les spécifications du vocabulaire fondamental. WS-Security et WS-Reliability sont des exemples de spécifications d'héritage pour des assertions externes qui doivent être définies. En comparant les politiques, conceptuellement chaque politique est d'abord convertie en forme normale disjonctive.

L'intersection de deux politiques comprend des alternatives de politiques compatibles inclus à la fois les politiques du demandeur et du fournisseur. L'intersection est une fonction commutative et associative, qui prend deux politiques et retourne une politique ». Si l'intersection est vide, les deux politiques sont incompatibles. Un ensemble d'assertions dans une politique est compatible avec un ensemble d'assertions dans une autre politique, si chaque instance d'un type d'assertion d'une politique est compatible avec chaque instance de ce type d'assertion dans l'autre Politique.

Si une instance d'une assertion se produit dans un seul ensemble, "le comportement associé à ce type d'assertion est interdit à l'intersection de ces politiques », bien que cette interprétation ne semble pas sémantiquement cohérente : si une politique exige du cryptage, et l'autre ne dit rien sur le cryptage, puis l'interdiction de chiffrement n'est pas compatible avec la première politique.

3.3.1.2. Le traitement des prédicats dans WS-Policy

La spécification qui définit l'Assertion <vp:A ...> doit être déterminé par la sémantique définie dans la spécification d'Assertion du domaine spécifique si les deux instances d'un type d'assertion donnée sont compatibles. Les auteurs du WS-Policy sont déterminés à donner des directives aux développeurs d'Assertion sur la façon d'écrire des assertions qu'ils peuvent comparer facilement [90].

Une assertion peut être un type complexe de XML. Par exemple :

```
<vp:A attrB="..." attrC="...">
<vp:D>example1</vp:D>
<vp:E>25</vp:E>
<vp:F attrG="..." />
</vp:/A>
```

La spécification qui définit l'Assertion `<vp:A ...>` doit définir toutes les variantes possibles de cet élément qu'un consommateur de service peut demander, c'est quoi l'intersection de deux instances de cette assertion, les combinaisons qui ne sont pas autorisés et comment les différentes formes d'assertions acceptable se relient aux instances acceptables du vocabulaire fondamentale spécifique au domaine qui fait l'objet de politique. Tout processeur de politique, qui doit vérifier un message ou comparer les instances de `<vp:A ...>`, doit incorporer un module de code qui implémente la sémantique spécifiée pour `<vp:A ...>`.

3.3.2 WSPL

3.3.2.1 WSPL Overview (vu generale)

La syntaxe de WSPL est un strict sous-ensemble du standard *OASIS eXtensible Access Control Markup Language (XACML)*. Une politique WSPL est une séquence d'une ou plusieurs règles, où chaque règle représente une alternative acceptable. Une règle est une séquence de prédicats, qui doivent tous être satisfaits pour que la règle soit satisfaite. Les règles sont listées dans l'ordre de préférence; le choix le plus préféré est cité en premier. Une politique WSPL est en forme normale disjonctive, où les règles sont logiquement liées avec le

«OU» et les prédicats à l'intérieur de chaque règle sont liées logiquement avec le «ET». Une description plus complète des WSPL est contenue dans [64].

3.3.2.2 la couche prédicat de WSPL

WSPL définit un langage standard pour une utilisation dans la spécification de prédicats qui contraignent les items du vocabulaire spécifié d'un domaine. Les prédicats de WSPL sont des fonctions XACML qui retournent des valeurs booléennes. Les paramètres des fonctions sont des attributs XACML et des valeurs littérales. Un attribut correspond à un item du vocabulaire qui définit le domaine. Les attributs sont référencés de deux façons, selon la manière dont le domaine définit ces derniers.

Un *AttributeDesignator* fait référence à un item du vocabulaire, en utilisant une URI (Uniform Resource Identifier) qui définit le domaine et un type de données standard.

Un *AttributeSelector* spécifie un item de vocabulaire, en utilisant une expression ***Xpath*** qui sélectionne un item d'un vocabulaire ; à partir du document XML qui définit le domaine. Ce document est, généralement, une instance de ce schéma qui définit le domaine du vocabulaire.

Chaque prédicat de WSPL met une contrainte sur la valeur d'un attribut. Les opérateurs de la contrainte sont : égal, supérieur à, supérieur ou égal à, inférieur, inférieur ou égal à, setequals et sous-ensemble. Tous les opérateurs de comparaison sont fortement typés et doivent être en accord avec les types de données spécifiées pour les paramètres de la fonction. WSPL supporte un ensemble riche de types de données utilisées dans XACML : chaîne, entier, nombre à virgule flottante (double), la date, l'heure, booléens, les URI, hexBinary, base64Binary, dayTimeDuration, yearMonthDuration, x500Name et rfc822Name. Ces types de données sont toutes tirées du XML schéma [65], à l'exception de deux types de la durée qui sont pris dans ***XQuer opérators*** [66] et les deux types de nom qui sont pris dans *XACML*.

3.3.2.3 Le traitement des prédicats dans le WSPL

Afin de trouver l'intersection de deux politiques WSPL, plusieurs étapes sont

effectuées. Premièrement, les cibles *(target*s) des deux politiques doivent se correspondre *(match)* (les *target*s sont décrits plus complètement dans [64]). Si les cibles ne se correspondent pas *(do not match),* alors les deux politiques ne sont pas compatibles. Deuxièmement, une nouvelle politique est créée, dans laquelle il existe une seule règle pour chaque paire de règles de politiques originales, la nouvelle règle contient tous les prédicats de ces deux règles originales. Pour tout ensemble donné de valeurs d'items du vocabulaire, cette nouvelle politique retournera "vrai" si et seulement si les deux politiques originales retourneront vrai, puisque la nouvelle politique conserve toutes les contraintes de ces deux politiques originales.

Les règles de *WSPL* sont listées dans l'ordre de préférence dans une politique: si une règle précède une autre, alors la politique propriétaire préfère la combinaison des valeurs d'item du vocabulaire spécifié par la première règle de la combinaison spécifiée par la deuxième règle. Par défaut, l'entité qui traite une intersection de politique préserve complètement les préférences d'une politique et les préférences de la deuxième politique, dans la mesure où celles-ci sont consistantes (compatibles) avec les préférences de la première. Une préférence plus complexe, combinant des algorithmes, peut être utilisée, mais il y a toujours la possibilité des conflits de préférence. L'algorithme combiné doit avoir un mécanisme pour résoudre ces conflits.

L'étape suivante consiste à fusionner les prédicats dans chacune de ces nouvelles règles telles que, pour chaque item du vocabulaire référencé dans la nouvelle règle ; il n'y a qu'un seul prédicat (ou deux prédicats dans le cas d'un champ valeurs d'item du vocabulaire délimitée à chaque extrémité) qui sera vrai si et seulement si tous les prédicats dans la règle dont les items du vocabulaire sont vraies. WSPL spécifie le calcul de ces prédicats, fondée sur les lois de l'arithmétique et de la logique pour chaque opérateur de fonction et type de données. *Par exemple, les deux prédicats "Attribute A > Value B" and "Attribute A = Value C" sont à la fois vrai si et seulement si "Value B > Value C" and "Attribute A = Value C". Si "Valeur B" n'est pas plus grand que "Valeur C»,* puis les deux prédicats sont incompatibles et, par conséquent, la nouvelle règle ne peut jamais être vrai et elle est éliminée de la nouvelle politique. Après cette étape, chaque règle restante est intérieurement consistante : il n'y a pas de conflit de prédicats sur le même item de vocabulaire. Les deux politiques originales sont incompatibles si et seulement si cet ensemble de règles résultant est vide. L'intersection de deux politiques spécifiées en utilisant le langage de prédicats *WSPL* peut être calculée.

Calculer cette intersection ne requiert aucune connaissance de la sémantique des références du vocabulaire spécifique au domaine référencié, mais ne dépend que de la sémantique des items de l'ensemble des fonctions standards et de types de données. La politique résultante est dans une forme telle qu'un utilisateur de politique peut sélectionner n'importe quelle règle, sélectionner les valeurs pour chaque item du vocabulaire compatibles (consistantes) avec les prédicats dans cette règle et que l'ensemble des valeurs résultant sera acceptable pour les deux politiques originales.

3.3.3. Comparaison des formes de prédicats

Ces deux styles de spécification des prédicats ont leurs avantages et leurs désavantages. Un seul prédicat de *WS-Policy* peut contrôler plusieurs items relatifs dans le vocabulaire fondamental; chaque prédicat de WSPL s'applique à un seul item.

Un prédicat WS-Policy peut être abstrait. Par exemple, une assertion peut déclarer que la signature numérique est nécessaire, sans en spécifier les détails de la syntaxe de cette signature. Cette même assertion pourrait être utilisé avec de multiples syntaxes de signature numérique. D'autre part un prédicat WSPL, s'il utilise des expressions Xpath pour mettre en référence des nœuds réels dans une instance de schéma du vocabulaire fondamentale, doit dépendre d'une valeur d'un nœud réel qui sera présent en particulier des instances de schéma. Cela peut faire des politiques complexes, s'il existe de multiples façons qu'une exigence particulière pourrait être satisfaite dans une instance de schéma (par exemple, il existe de multiples façons de mettre en référence un objet qui doit être signé dans un message en utilisant le standard XML Digital Signature). Les attributs *name / type / value* de XACML peuvent être définis, toutefois, pour accomplir les mêmes fonctions d'abstraction comme les *Assertions de WS-Policy*. Les assertions de WS-Policy, qui doivent être comparé entre deux politiques peuvent être facilement déterminé, parce que les assertions ont habituellement le même nom. Il peut y avoir des cas où deux assertions déférentes doivent être comparées, par exemple, lorsqu'un consommateur revendique une Assertion "MaximumBuyingPrice", tandis qu'un fournisseur affirme une Assertion «MinimumSellingPrice».

Le *Comparable WSPL AttributeDesignators* peut toujours être matché, parce qu'ils doivent porter le même nom; les sémantiques similaires "maximum" et "minimum" sont capturées

dans l'opérateur de la fonction plutôt que dans l'attribut lui-même. Si *AttributeSelectors,* en utilisant des expressions XPath, sont utilisées, il peut y avoir plusieurs expressions qui pointent vers le même nœud d'une instance de schéma.

Un attribut XACML pourrait être défini pour exprimer de telles sémantiques, mais il ne peut pas être fait avec l'expression *XPath ;* car il n'y a rien dans le document qui indique l'ordre dans lequel les noeuds ont été ajoutés. Noter que ce type de prédicat ne peut pas être vérifié contre un message donné. Il doit simplement être affirmé comme une exigence sur un processeur de document. Afin d'utiliser une Assertion WS-Policy pour la vérification de message, le moteur de vérification doit inclure le code spécial qui sait relier une assertion donnée à un type particulier de message. Un prédicat WSPL, qui utilise des expressions XPath, peut être utilisé directement pour vérifier que le prédicat est satisfait dans un message. Les Assertions de WS-Policy peuvent être définies dans des spécifications propriétaires. Même si la spécification est finalement standardisée, il peut y avoir une longue période au cours de laquelle la spécification est en cours de développement et n'est pas accessible à tous les développeurs de processeurs de politiques. En particulier, pour les politiques relatives aux vocabulaires spécifiques à l'application, il peut y avoir peu d'incitation à la ruée vers une politique de standardisation. Les prédicats de WSPL, cependant, peuvent se référer directement à la spécification du vocabulaire fondamental. La sémantique de ces prédicats est standardisée et ne dépend pas de la spécification fondamentale. Alternativement, un XSLT peut être utilisé pour traduire les informations de l'instance d'un schéma propriétaire dans un format non propriétaire, tels que des attributs XACML pour une utilisation dans la spécification de politiques. Les opérateurs booléens définis dans *WS-Policy* peuvent être imbriqués, ce qui se traduit dans un Format compact de politique. Afin de traiter une politique, elle doit être au moins converti en Forme Normale Disjonctive.

En WSPL, les politiques sont toujours en Forme Normale Disjonctive. Avec le fait que des fonctions sont utilisées pour spécifier la sémantique, plutôt que d'avoir de la sémantique qui est implicite dans le prédicat lui-même, signifie qu'une politique donnée exprimée en *WSPL* exige presque toujours plus d'octets pour son expression que la politique correspondante de WS-Policy. La plus grande différence entre les Assertions de *WS-Policy* et les prédicats de WSPL est que chaque assertion a une unique sémantique définie du domaine qui doit être capturé dans un module de code incorporé dans toute entité qui doit traiter l'assertion, que ce soit de le comparer ou de le vérifier. Chaque nouvel ensemble d'assertions définies du

domaine exige que les processeurs de politiques doivent être mis à jour pour supporter tout changement des Assertions existantes, de même, exige des mises à jour pour le processeur.

Tout les processeurs qui n'ont pas été mis à jour ne seront pas capables de traiter des assertions nouvelles ou modifiés, ce qui les rend moins probables que les politiques seront interopérables entre les différentes plates-formes. Comme de plus en plus les Assertions sont définies, la complexité de l'entretien de chaque processeur de politique augmente. Les prédicats de WSPL, utilisent un ensemble fini de fonctions standards qui ne dépendent pas de la sémantique définie du domaine. Tout processeur de WSPL peut traiter toute politique WSPL, nouvelle ou ancienne, et indépendamment du fait que le vocabulaire fondamental est défini dans la spécification propriétaire ou non.

3.3.4 Avantages et inconvénients : WS-Policy et WSPL

Le principal avantage de la forme WS-policy est que les prédicats ont tendance à être compact et facile à lire. Le principal inconvénient est que le processeur de politique doit être configuré pour supporter la syntaxe et la sémantique de chaque type de prédicat qui sera utilisé par toutes politique. Le principal avantage de la forme WSPL est qu'un processeur de politique standard est capable de calculer l'intersection de deux politiques et de vérifier n'importe quel message contre une politique. Le principal inconvénient est que les prédicats qui mettent en référence directement des nœuds dans un exemple du schéma de domaine peuvent être trop spécifique, bien que WSPL soutien également la création de nouveaux items du vocabulaire pour exprimer les exigences les plus abstraites. WS-Policy n'a, actuellement, aucun mécanisme de préférence et de sémantique des prédicats manquants qui semble être incorrect; WSPL permet aux alternatives de politique d'être classées par préférence. WSPL a besoin d'un sous ensemble *XPath* qui peut être utilisé pour identifier un item de politique.

3.4 Conclusion

Dans ce chapitre nous avons vu les deux approches de langages de spécification de politiques, à savoir les langages de l'approche *Web sémantique et les langages de l'approche de combinaisons booléennes de prédicats de politique*.

Une description basée sur l'ontologie de politique permet au système d'utiliser des concepts pour décrire les environnements et les entités étant contrôlées. Ainsi les approches basées sur l'ontologie simplifient l'accès à l'information de politique, avec la possibilité de calculer dynamiquement des relations entre les politiques et l'environnement, les entités ou d'autres politiques basées sur des relations d'ontologie plutôt que leurs fixations à l'avance. La spécifications de politique basée sur l'ontologie peut être difficile à implémentée à cause du niveau élevé de la spécification des politiques basées sur l'ontologie.

Les langages de politique de service Web, qui utilisent des combinaisons booléennes de prédicats, diffèrent essentiellement dans les formes que les prédicats prennent. Dans WS-Policy, les prédicats sont des éléments XML dont la syntaxe et la sémantique sont spécifique au domaine, bien que chaque item de politique ou groupe d'items ayant son propre ensemble de prédicats. En WSPL, les prédicats sont des fonctions standard XACML au-dessus d'une référence d'item de vocabulaire de politique et une valeur littérale.

Chapitre 4

La compatibilité de politiques.

Chapitre 4

La compatibilité de politiques.

4.1 Introduction

Les politiques sont utilisées de plus en plus pour la gestion des systèmes automatisés et le contrôle du comportement des systèmes complexes [108], en permettant aux administrateurs de modifier le comportement d'un système sans modifier son code source; ou exiger une coopération des composants qui sont administrés. Nous assistons aussi, aujourd'hui, à l'utilisation de politiques pour décrire les contraintes d'utilisation des services Web; ce qui implique la nécessité de vérification de la compatibilité de politique dés qu'il s'agit d'interagir deux services web ou une composition de services Web.

L'adoption d'une approche basée sur la politique pour contrôler un système exige une représentation de la politique appropriée et une conception avec développement d'un Framework de gestion de politiques. Les politiques seront de plus en plus importantes pour une vraie implémentation des services Web sémantique [96, 97].

Dans ce chapitre, nous évoquons quelques définitions de politiques ainsi leurs origines et quelques travaux relatifs à la compatibilité de politiques dans la composition de services Web.

4.2 Les Origines et les motivations de la gestion basée sur la politique

Les politiques qui contraignent le comportement de composants du système deviennent une approche de plus en plus populaire à adaptabilité dynamique d'applications dans le monde universitaire et industrielle [108]. Nous allons signalé quelques avantages des approches basées sur la politique, à savoir; la réutilisabilité, l'efficacité, l'extensibilité, la sensibilité au contexte, la vérifiabilité, le support pour les composants simples et sophistiqués, se protégé contre les composants mal conçus ou malveillant et raisonner au sujet de comportement de composant. les politiques sont souvent appliquées pour automatiser des tâches d'administration de réseau, telles que la configuration, la sécurité et la qualité de service **(QoS).**

Dans le domaine de la gestion réseau, les politiques sont exprimées comme un ensemble de règles gouvernant le choix dans le comportement du réseau. Les approches multiples pour la spécification de politique ont été proposées ; cette gamme de langages de politique peut être traitée et interprétée facilement et directement par un ordinateur, pour la notation basée sur les règles de politique en utilisant le *if-then-else*, et pour la représentation de politiques comme entrées dans une table qui consiste en attributs multiples [104, 105, 79]. Il y a aussi des efforts de la standardisation progressive vers un modèle d'information et un Framework de politique commun.

The Internet Engineering Task Force, par exemple, a enquêté sur des politiques comme un moyen pour la gestion du *réseaux IP-Multiservice* en se concentrant sur la spécification de protocoles et les modèles orienté objet pour représenter les politiques [107]. Le domaine de gestion de politique dépasse de plus en plus ces applications traditionnelles d'une manière considérables. Les nouveaux défis pour la gestion de politique incluent :

- La protection des sources et des méthodes, gestion des droits numériques, filtration et transformation de l'information et l'accès basé sur la capacité.
- Les réseaux actifs, le calcul rapide et agile, les systèmes répondus et mobiles.
- Le modelage d'organisation, formation de la coalition, formaliser des accords cross organisationnels.
- Les modèles de confiance, gestion de confiance, ascendances de l'information.
- interaction homme machine efficace : gestion de l'interruption/notification, gestion de la présence, autonomie ajustable, facilitation de la collaboration, sécurité.

- Le recouvrement intelligent de toutes les politiques appropriées à quelques situations.

Par leurs capacités d'opérer indépendamment sans surveillance humaine constante, les agents peuvent exécuter des tâches qui seraient impraticables ou impossibles en utilisant des applications logicielles traditionnelles. D'autre part, cette autonomie additionnelle, si elle est non vérifiée, offre également de la possibilité intéressante d'effectuer des dommages graves si des agents sont mal conçus. Le contrôle du comportement d'un agent est une tâche complexe parce que le comportement d'agent ne peut pas être programmé pour faire face à toutes les situations, mais exige des ajustements dynamiques et continus pour permettre à des agents d'agir dans tous les contextes d'exécution de la manière la plus appropriée [95].

4.3 Définitions

Les politiques sont définies comme '' des *informations qui peuvent être utilisées pour modifier le comportement d'un système* '' [96].

La *définition* de *M. Zakaria et all* va au-delà de la modification du comportement et considère les politiques comme règles et des paramètres externes, dynamiquement modifiables qui sont utilisés comme entrée d'un système. L'utilisation prévue des politiques doit indiquer des actions à exécuter après des détections d'événements et des changements du contexte. Les politiques sont en général regardées comme un ensemble de règles, contraintes, ou ordres qui contrôlent le comportement d'une entité. Dans le domaine des services Web, les politiques sont destinées à spécifier des aspects différents du comportement d'un service Web. Cela aide un service Web à aligner ses capacités et contraintes aux exigences de l'utilisateur [99].

Les politiques sont un moyen pour contrôler, dynamiquement, le comportement de composants du système sans modifier ou changer son code source, ou exiger le consentement, ou une coopération des composants qui sont gouvernés [70, 95].

Les politiques sont vues comme information que les développeurs peuvent utiliser pour modifier le comportement d'un système. [98].

4.4 pourquoi les politiques?

Les politiques sont vues comme des règles et des paramètres externes, dynamiquement modifiables qui sont des entrés d'un système. Le système peut s'ajuster alors aux changements dans l'environnent d'exécution. Ainsi, les politiques spécifient les actions qui doivent être exécutées selon les événements détectés et les changements du contexte.

Les politiques sont ainsi définis particulièrement pour la composition de services Web. Les développeurs peuvent les utiliser pour gérer des services Web à hauts et bas niveaux. Les politiques de haut niveau adresseraient la composition, en incluant quelles préférences de l'utilisateur à intégrer dans les services Web, ce qui est nécessaire pour satisfaire ces préférences et quand la progression de l'exécution de services Web est continuée. Les politiques de bas niveau incluraient comment échanger de l'information compréhensible entre les services Web, comment traiter le manque de fiabilité d'un service Web, comment substituer un service Web avec un autre pair sans interrompre l'exécution et comment suspendre une exécution d'un service Web en raison du risque que le comportement pourrait être changer ou l'information pourrait être interceptée.

L'adoption de politiques introduit aussi la possibilité de changer le comportement des services Web sans modifier la spécification de la composition. En utilisant les politiques, les deployeurs de services Web peuvent ajuster de façon continue des aspects multiples tels que les mécanismes de la résolution de conflit de services Web pour accommoder des variations dans l'environnement. Finalement, les politiques seraient utiles dans la poursuite des actions qui causent des exceptions inattendues et en indiquant comment ces exceptions seront réparées sans avoir le besoin de changer la spécification de la composition.

4.5 Association de politiques avec les services

Il y a plusieurs de façons d'associer des politiques avec les services Web. *La figure 4.1* illustre six scénarios possibles présentés par Z. Maamar et al [112].

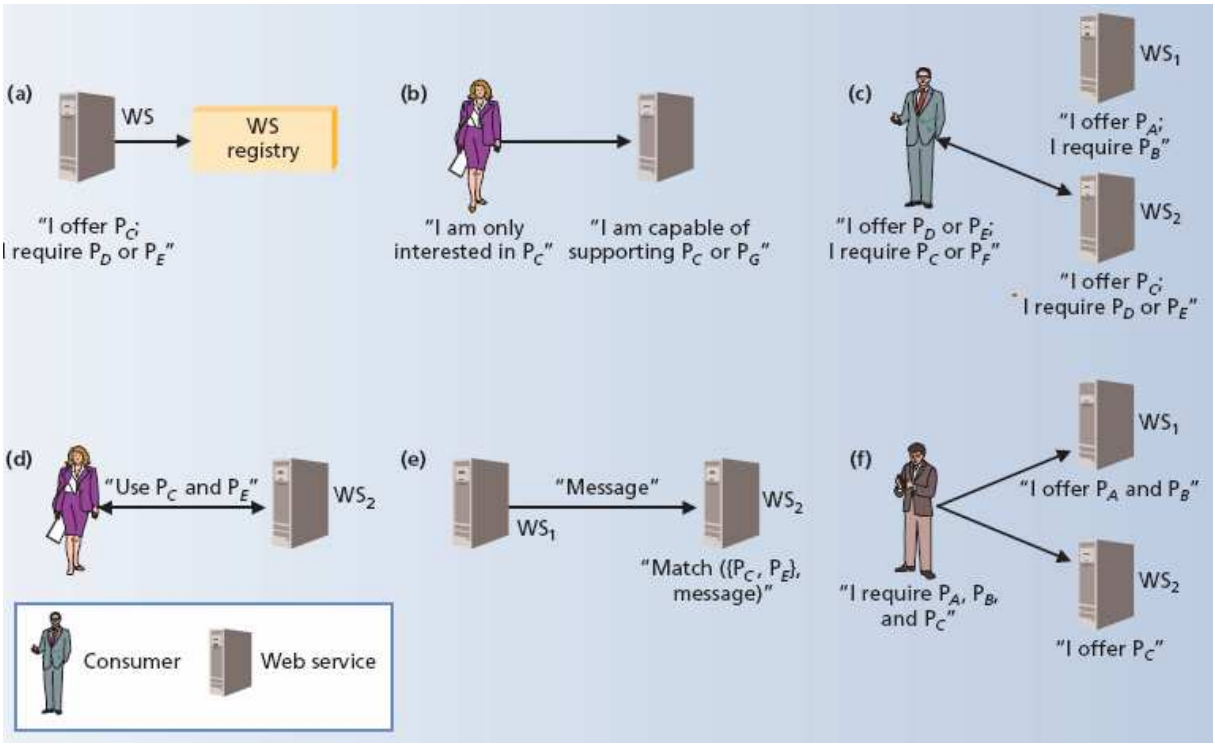

Figure 4.1 − les six scénarios qui illustrent les cas potentiels d'utilisation des politiques dans les services Web, publication (a), sélection (b), compatibilité (c), accord (d), vérification (e) et composition (f).

Le scénario de publication *(figure 4.1a)* illustre comment offrir et exiger des politiques. Quant aux règles, elles peuvent être une partie de descriptions de services Web dans le WSDL inscrit sur un UDDI-Like registry. Ces politiques dictent comment un service Web agit en tant que composant dans un service Web composite. Ensuite, le scénario de la sélection *(figure 4.1b)* illustre comment utiliser des politiques pour identifier un service Web ou une communauté de services Web pour satisfaire la requête d'un consommateur. Le consommateur pourrait être un utilisateur ou un autre service Web. Ici, le consommateur s'intéresse seulement aux services Web qui offrent la politique P_c. Le scénario de compatibilité *(figure 4.1c)* illustre comment les interactions entre les services Web peuvent également se produire au niveau de la politique qui demande de garantir la conformité des politiques. Cette conformité s'étend aux données que les politiques gèrent et échangent. Ensuite, le scénario de l'accord (*figure 4.1d*) illustre comment un consommateur, tel qu'un service Web, et un autre service Web différent conviennent sur des politiques à utiliser, à savoir P_C et P_E. Les actions de négociation et de compatibilité peuvent précéder l'accord.

Le scénario de la vérification *(figure 4.1e)* étend le scénario de l'accord. Un accord est principalement une promesse pour se conformer à quelques politiques, par exemple P_C et P_E. Cette promesse doit être accomplies (satisfaite) au moment de l'exécution; sinon le service Web pourrait faire l'objet de sanctions, comme l'abaissement de sa réputation. Dans la *figure 4.1e*, l'interaction entre les services Web est un message de transaction de business soumis aux politiques P_C et P_G. Le service Web récepteur doit vérifier que le contenu du message et d'autres comportements associés à cette transaction sont conformes aux politiques convenues P_C et P_E. Finalement, le scénario de la composition *(figure 4.1f)* illustre la composition de multiples services Web pour satisfaire les exigences d'un consommateur. Par exemple, supposons qu'un consommateur exige trois politiques, P_A, P_B, et P_C, et qu'aucun service Web élémentaire ne leur offre tout. Deux services Web, un qui offre P_A et P_B et un autre qui offre P_C, Forment un ensemble de services Web qui offre conjointement P_A, P_B, et P_C.

4.6 La composition basée sur la compatibilité de politiques.

Les services Web sont, de nos jours, une *solution* pour développer des applications ***B2B*** *(Business to Business)*. Ces applications appellent la composition de services Web, qui a comme résultat le développement des *services* Web composites parce que, en général, il est peu probable qu'un corps simple fournirait toutes sortes de services Web et développerait toutes sortes de fonctionnalités. L'hétérogénéité de services Web et l'incompatibilité de leurs politiques ne devrait pas être ignoré comme obstacle qui pourrait être produit et entraverait le progrès d'une exécution de composition. Lors de la composition de services, une vérification de compatibilité de ces politiques de services Web composant est indispensable. Le problème de la compatibilité entre ces politiques pourra empêcher la composition automatique de ces services Web. Les politiques règlent / contraignent le fonctionnement de services Web à plusieurs niveaux, par exemple le business, le comportement et la confidentialité… etc. Puisque les services Web indépendants collaborent entre eux afin de satisfaire les besoins des utilisateurs, leurs politiques respectives pourraient être en contradiction. Ceci peut mener aux conflits et aux exceptions au temps d'exécution.

Dans ce qui suit, nous examinons quelques travaux, qui traitent l'hétérogénéité *des politiques* qui cadre le développement et le déploiement de la composition de services Web.

4.6.1 Utilisation de la sémantique pour la composition basée sur la politique

En cet article [100], les auteurs ont classifié les diverses règles et contraintes du processus réels du business en syntactique, sémantique et politique qui servent à déterminer trois niveaux de compatibilité syntactique, sémantique et politique.

Règles (opérationnelles) syntactiques : La composabilité des services dépend des conditions préalables d'entrée et sorties. Les règles syntaxiques assurent que la composition des services produit un service composite qui est opérationnel ou syntaxiquement possible.

Règles sémantiques : Les règles sémantiques de composition exigent souvent le domaine de la connaissance experte, telle que les lois du commerce, les législations d'état, les règlements environnementaux fédéraux, les politiques formelles de compagnie, les procédures expérimentales, etc. L'application des règles sémantiques joue un rôle significatif dans la composabilité de service pour assurer que les services composés sont compatibles au niveau sémantique.

Politiques : plusieurs services possèdent le même profil et fournissent les mêmes fonctionnalités. Le choix automatisé d'un service parmi ces services, fonctionnellement équivalents peut exiger la compatibilité de niveau de politique.

Les règles syntactiques et sémantiques et politiques de business permettent à un agent de limiter le choix et la composabilité des services. Ces règles permettent aussi à la composition de service d'être mieux adaptée aux besoins du client.

Les trois types de politiques qui ont été définis :

Les politiques de service qui sont définies par les différents organismes offrant des services.

Les politiques de service flow qui sont définies par les organismes offrant un service composé de Web comportant des services composants.

L*es politiques d'utilisateur* ***se sont les*** politiques locales et globales d'utilisateur qui sont définies par les consommateurs des services.

Ces règles jouent un rôle important en (i) découverte, (ii) sélection et (iii) composition des services Web. Ces règles incorporent la connaissance qui est nécessaire pour choisir et composer des services Web dans un service flow cohérent. Ils ont présenté un modèle formel pour la composition de service Web, appelé le modèle de composition ***Service Flow.***

Ce modèle inclut : (i) l'ontologie de service, (ii) les types de service et leurs descripteurs et (iii) les expressions de politiques de service imposés par des fournisseurs de service. Le modèle considère également la politique d'utilisateur. Les services non seulement doivent avoir des politiques compatibles avec les politiques d'utilisateur afin d'être choisis dans la composition, mais ils doivent également satisfaire la compatibilité de niveau syntactique, sémantique et de politique. La compatibilité de niveau sémantique envisage les rapports sémantiques compatibles entre les services. La compatibilité de niveau de politique considère des compatibilités spatiales, temporelles et la valeur de la gamme (le champ).

Les auteurs fourni un algorithme pour choisir un service à partir des services fonctionnellement équivalents en vérifiant ces compatibilités de politique. Ils ont également présenté une approche pour des négociations de politique, dans les cas où la politique compatible n'existe pas. Leur approche est pour implémenter la composition de service Web qui exige la connaissance (ontologie) des services Web dans le domaine, et les règles politique.

4.6.1.1 Modèle de compatibilité de service

Les protocoles ou les règles sur la manière de composer des services valides sont indiqués comme un ensemble de *règles générales de compatibilité*. Les règles de compatibilité se réfèrent aux aspects *syntactiques, sémantiques et de politique* des contrats de service. La compatibilité syntactique vérifie que les entrées, les sorties et les conditions préalables entre les services individuels de différents ensembles sont compatibles. La compatibilité sémantique assure, que dans un domaine spécifique, les relations sémantiques entre les services sont compatibles ou les exigences d'utilisateur sont satisfaites. La compatibilité de niveau politique assure que les règles imposées par une organisation permettent à son service d'être composable avec un autre service d'une autre organisation qui

possède une politique spécifique. Ces contrôles de compatibilité vérifient les relations des politiques et des règles pour filtrer tous les services incompatibles dans un service composite.

Pour s'assurer que ces compatibilités sont vérifiées, un ensemble de règles compositionnelles de vérification s présenté dans leur modèle. Ces règles sont spécifiées comme suit :

Les types de règles de composition du modèle présenté par les auteurs, sont les suivants :

La composabilité générale : c'est-à-dire qu'il existe une compatibilité syntaxique, sémantique et politique entre s_i *et* s_j.

Inclusion de sous type de service : déclare que si l'objectif (service flow) de l'utilisateur est un service s_j, donc, obligatoirement, son sous type s_i doit faire partie du service flow SF.

Satisfaction des politiques de l'utilisateur : L'une des règles de la composition déclare que si les contraintes de l'utilisateur indiquées (spécifiées) au dessus de s_i sont toutes réunis avec la politique de service fournie par un fournisseur de service, donc le service peut être ajouté (c'est-à-dire le service est candidat) dans le service flow SF.

Relations d'ordre : déclare que si $\mathbf{s_i}$ est de type sémantique (comme la commande d'un produit), et le $\mathbf{s_j}$ est du type de service qui le suit sémantiquement (comme livraison du produit), alors la composition devrait mettre une relation d'ordre, s_i avant $\mathbf{s_j}$, (e.g : la commande du produit devrait procéder à la livraison).

Les services compatibles syntaxiquement auront une relation d'ordre suivant les relation de leurs entrées et sorties.

4.6.1.2 La composition de services basés sur la politique

Dans cette section, ils décrivent le processus de composition de services en utilisant les expressions (déclarations) de politiques de services et les politiques de l'utilisateur. Ils ont développé un algorithme de composition automatique de services flow et ils ont implémenté un prototype dans le domaine du E- gouvernement**.**

Dans le domaine du commerce électronique, les services avec les mêmes fonctionnalités peuvent être fournis par beaucoup de fournisseurs de service. Le problème posé est de choisir l'un des services qui est mieux approprié à la composition.

Leur approche, est de générer automatiquement un service flow avec des ensembles de services candidat, $\mathbf{S= \{S_1,S_2,..\}}$ avec des relation d'ordre entre eux. Chaque service candidat $\mathbf{S_i \in S}$ est un ensemble de services fonctionnellement équivalents, $\mathbf{S_i = \{s_1,s_2,..\}}$ tel que chaque $\mathbf{s_j \in S_i}$ est un service fonctionnellement équivalent a $\mathbf{s_k}$, mais le $\mathbf{s_j}$ et le $\mathbf{s_k}$ sont fournis par différents fournisseurs et sont implémentés par différentes règles de politique. A fin de choisir un service $\mathbf{s_j}$ approprié à partir de chaque S_i, ils utilisent les règles de composition qui inclus la compatibility entre $\mathbf{s_k}$ dans $\mathbf{S_i}$ et $\mathbf{s_l}$ dans $\mathbf{S_j}$ au niveau syntaxique, sémantique et politique.

Le processus de composition de service Web comprend les étapes suivantes

Premièrement, identifier les services composants du service flow, assigner les politiques d'utilisateur correspondance de politiques d'utilisateurs et les politiques de service, choisir un service à partir d'une liste de candidats et vérifier si les services instancier correspondent aux politiques globales d'utilisateur pour le service flow.

4.6.1.3 Négociation de politique pour détendre les règles compositionnelles

Le processus de correspondance (compatibilité) de politique peut avoir comme conséquence une liste vide de candidat, Quand il n'y a aucun service dont la politique correspond parfaitement à la politique d'utilisateur.

Afin d'éviter l'échec dans la composition de service, les auteurs ont introduit :

- *La compatibilité flexible qui envisage un processus de négociation pour vérifier les politiques de service et celles de l'utilisateur.*

Les échecs de compatibilité peuvent provenir de diverses sources, comme les différences entre les descripteurs en raison de leurs provenances de différentes ontologies de service. Ainsi, quand il y a des différences, le système devrait pouvoir vérifier si les descripteurs sont

des synonymes. Si c'est le cas, alors l'algorithme évalue les valeurs pour une compatibilité réussie.

Selon le degré de compatibilité, ils définissent les classes suivantes des compatibilités :

Compatibilité Parfaite : Quand les descripteurs et les valeurs des expressions de politique de service sont compatibles avec les expressions de politique d'utilisateur, alors on dit que la compatibilité est parfaite lorsque les synonymes des descripteurs ou les synonymes des valeurs sont également compatibles parfaitement.

Compatibilité Approximative De Descripteurs : Quand un descripteur d'une politique d'utilisateur est une sous classe du descripteur utilisé dans une politique de service, alors la compatibilité évalue leurs valeurs correspondantes.

Compatibilité Approximative de Valeurs : Quand la valeur d'un descripteur pour une politique d'utilisateur a un domaine plus étendu que celui d'une politique de service, alors on dit qu'il y a une compatibilité approximative de valeurs.

Aucune compatibilité : Quand il n'y a aucune compatibilité ou aucune relation qui existe comme les compatibilités approximatives entre les descripteurs où entre les valeurs, on dit il n'y a aucune compatibilité. En cas il n'y a pas de compatibilité, le système ne pourrait pas trouver un exemple de service. Mais, pour la compatibilité approximative, le système devrait pouvoir la négocier. Il y a différentes stratégies qui peuvent s'appliquer :

1. L'une est d'obtenir plus d'information de l'utilisateur pour rétrécir ou limiter l'espace.
2. L'autre peut être qu'un fournisseur de service qui a un ensemble de politiques dans des classes différentes, de la politique la plus souhaitable à la moins souhaitable.

Dans le dernier cas, quand il y a une compatibilité approximative ou aucune compatibilité avec la politique la plus souhaitable, la prochaine politique souhaitable est utilisée pour la compatibilité. Les politiques sont employées comme un ensemble de stratégies de négociation pour le système. De la même manière, un utilisateur pourrait indiquer ses préférences à l'avance, de sorte que quand il n'y a aucune compatibilité, les politiques de l'utilisateur seraient successivement utilisées pour la compatibilité. Au cas où il y aurait une politique de

service négociable dont les conditions ne correspondent pas à la politique d'utilisateur, le processus de négociation de politique est lancé. Quand ce processus aura comme conséquences (résultats) des conditions mutuellement agréables, la politique est définie pour être une politique compatible. Le processus de vérification des préférences de l'utilisateur et des politiques de services est vu comme un processus de négociation avec les conditions additionnelles qui doivent être satisfaites pour la consommation du service.

4.6.2 Une approche de politique multi-type basée sur le contexte

L'objectif de ce travail de recherche mené par *M. Zakaria et all* [99], est d'examiner le rôle de *politiques et du contexte* dans le cadre de la composition de services Web. Le contexte supporte le développement de services Web adaptable et les politiques spécifient le comportement des services Web engagés dans une composition. Dans ce papier, les auteurs présentent le résultat de ce travail de recherche dans lequel la liaison entre les services Web se produit à quatre niveaux : le n*iveau composant, niveau composé, niveau sémantique et le niveau de ressource*. L'approche proposée par les auteurs utilise trois types de politiques: *les politiques de l'engagement, les politiques de la médiation et les politiques du déploiement.* Dans la *figure 4.2, M. Zakaria et all*. illustrent *l'approche de la politique multi type basée sur le contexte pour la composition de services Web*.

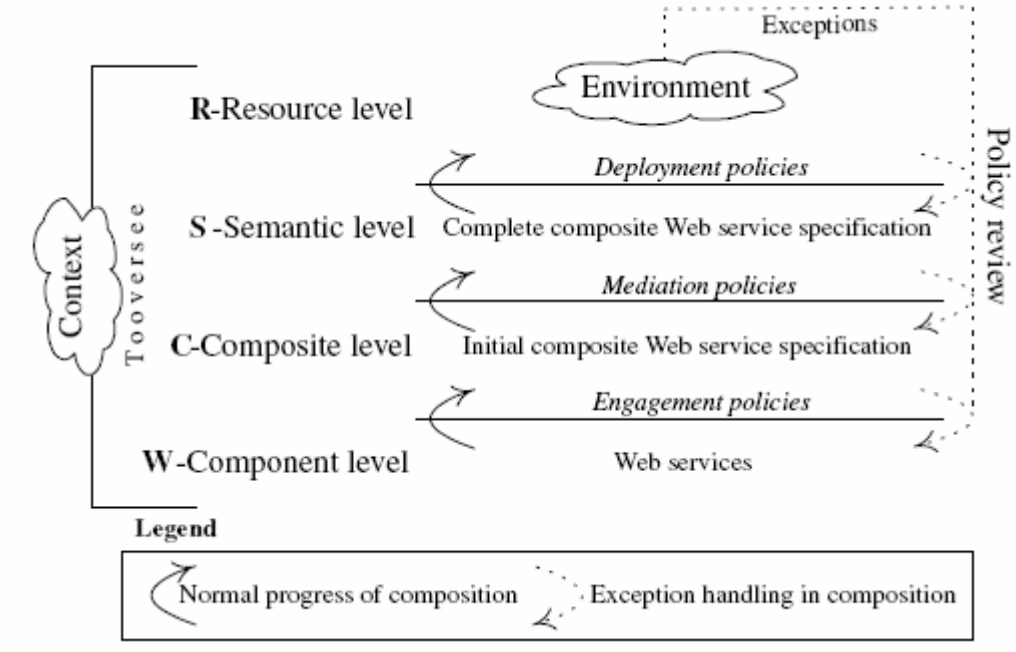

Figure 4.2 − Représentation de l'approche de politique multi-type basée sur le contexte

Avant de déclencher toute politique et exécuter toute action, l'information que le contexte fournit est validée. Un tableau a été donné, résumant les actions qui sont exécutées pendant les transitions entres les niveaux. Ces actions sont catégorisées par le type du niveau et sont présentées comme suit **:**

Au niveau W-Composant, les actions se préoccupent de chercher les services Web composant et chercher l'acceptation de ces services Web composant pour participer à la composition

Au niveau C-Composite, les actions se préoccupent de relier les services Web composant, suivant une spécification spécifique et détecter les hétérogénéités sémantiques qui pourraient survenir entre ces services Web composant.

Au niveau S-Sémantique, les actions se préoccupent de l'identification du type de la médiation (sémantique, structurel, et syntactique), cela a besoin d'être exécuté entre les arguments d'entrée/sortie des services Web composant, et assurer le succès de cette médiation.

Au niveau R-Ressource, les actions se préoccupent de l'obtention des services Web prés pour l'exécution, après chercher les ressources appropriées et satisfaire les exigences de la sécurité des services Web et des ressources.

4.6.2.1 Description de politiques

Les politiques permettent, pendant le scénario du progrès normal, de déplacer la composition d'un niveau à un niveau plus haut (direct). *figure 4.2* illustre trois types de politiques: *engagement, médiation et déploiement*. Les résultats de politiques sont soumis aussi à révision en cas d'exceptions, c.-à-d., progrès anormal.

Dans ce qui suit, ils expliquent le rôle de chaque type de politique comme suit :

I. **Les politiques d'engagement :** encadrent la participation des services Web composant dans les services Web composites. Cet encadrement met en valeur la participation d'un service Web dans plusieurs compositions**.**

Depuis que les services Web sont *contexte-aware*, ils évaluent (estimation) leurs participations et utilisent les politiques de l'engagement pour annuler leurs acceptations ou refuser la participation dans d'autres compositions supplémentaires.

II. **Les politiques de la médiation :** traitent (s'occupent) les questions de l'hétérogénéité sémantiques. Parce que les services Web proviennent des fournisseurs différents, l'ingénierie d'échange de données est cruciale**.** Cette ingénierie est de deux types.

i. Le premier type est au sujet des arguments d'entrée et sorties de services Web qui ont besoin d'être mappés (mapped) selon ce qu'ils ont, plus tôt, connu sous le nom ***d'ontologie globale.***

ii. Le deuxième type est au sujet des valeurs d'entrée et sorties de services Web. Ces valeurs ont besoin d'être converties selon ce qu'ils ont, plus tôt, connu sous le nom ***d'ontologie locale***.

Les politiques de la médiation sont responsables de supporter le mappings et conversions entre les services Web composant, une fois ce dernier décide de participer à un service Web composite, suivant un déclenchement réussi des politiques de l'engagement.

III. **Les politiques du déploiement :** visent à encadrer les interactions entre les services Web composant et le calcul des ressources. Les auteurs ont discuté plus tôt que les ressources ont limité les capacités du calcul, et l'ordonnancement de l'exécution des requêtes des composants de services Web composant est considéré juste.
Une révision (examiner) des politiques se passe, quand les exceptions surviennent pendant l'exécution de ces politiques. Une révision de la politique du déploiement de ce service Web est exigée pour que des situations semblables peuvent être évitées dans le futur.

4.6.2.2 Spécifications de politiques

Dans ce papier, ils utilisent le Langage de politique de services Web ***(WSPL)*** basé sur les arguments qu'*Anderson* avance dans [101]. Anderson suggère et propose ***WSPL,*** pour exprimer les politiques et accomplir ou atteindre l'interopérabilité des services Web.

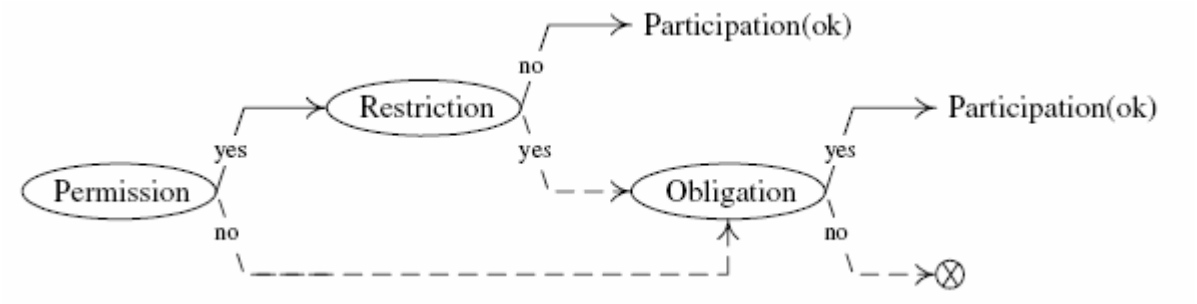

Figure 4.3 − Le comportement d'un service Web dans le scénario d'engagement.

La *figure 4.3* illustre comment les différents comportements de services Web sont reliés en se basant sur le résultat de l'exécution de la politique d'engagement par type de comportement. Les lignes tirées dans Fig. 4 représentent les cas potentiels d'exceptions.

1. Une politique d'engagement consiste à donner une ***permission*** pour qu'un service Web fasse partie d'un service Web composite. L'autorisation ou la permission est basée sur l'état du service Web qui a su utiliser son W-contexte correspondant. En cas de permission (autorisation) positive, la procédure de la participation est exécutée qui résulte la mise à jour des arguments du ***W-Contexte*** du service Web suivants : ontologie locale, nombre de participations actives, les services Web précédents, les services Web courant et les prochains services Web.

2. une politique de l'engagement pour ***restriction*** consiste en empêcher un service Web de participer à un service Web composite. Une restriction n'exécute (n'implémente pas) pas une permission négative de participation, plutôt elle suit une autorisation positive de participation *(figure 4.3)*. Les Restrictions pourraient être adaptées vers une issue de qualité (par exemple, temps de réponse, réputation, performance, débit) des services Web composant avec qui un service Web interagie.

3. les politiques d'engagement pour *l'obligation* garantie qu'un service Web fera précisément partie d'un service Web composite, malgré la permission négative ou existence de restrictions *(figure 4.3)*.

Politique de la médiation

La ***figure 4.4*** qu'ils ont donnée**,** illustre le chemin ou la manière de la progression de la réconciliation sémantique entre les services Web composants. Cette progression est basé sur le résultat de l'exécution de la politique de la médiation par type d'opération connu comme ***mapping et conversion***, respectivement.

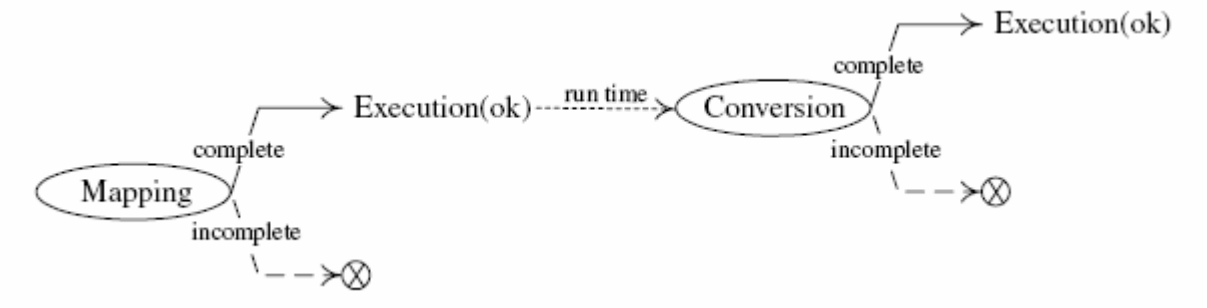

Figure 4.4 − Opération d'un service Web dans le scénario de médiation

1. *une politique de la médiation pour le mapping* concerne des opérations qui permettent d'établir des liens entre les arguments d'entrées et sorties des services Web composant d'un service Web composite. Un lien se produit selon l'ontologie globale à laquelle un service Web composite est lie. L'ontologie globale est rapportée dans le ***C-Contexte*** du service Web composite.

2. *une politique de la médiation pour la conversion* concerne des opérations qui gèrent des valeurs des arguments d'entrée et sorties des services Web composant d'un service Web composite. Cela se produit selon l'ontologie locale à laquelle les services composants sont lies. L'ontologie locale est rapportée dans le ***W -contexte*** d'un service Web composant.

Politique du déploiement

La *figure 4.5***,** qu'ils ont donnée, illustre le chemin que les différents comportements d'un service Web sont reliés en se basant sur le résultat de l'exécution de la politique du déploiement, par type de comportement.

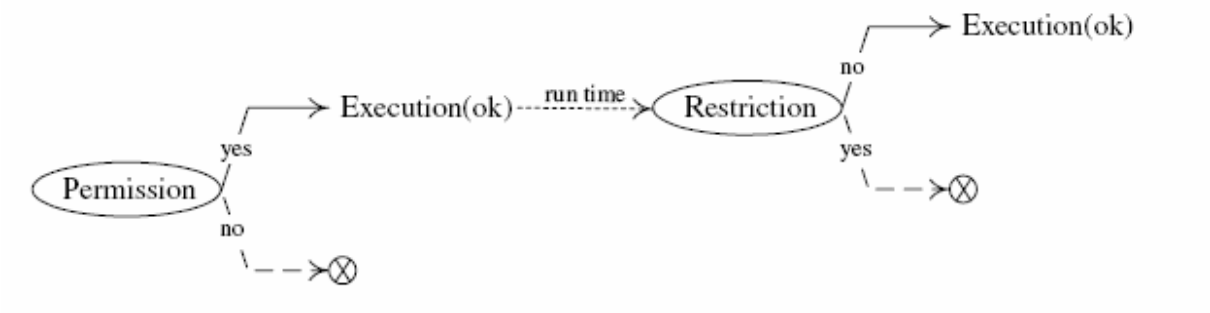

Figure 4.5 − *le comportement d'un service Web dans le scénario de déploiement*

1. *une politique du déploiement pour la permission* concerne un service Web qui reçoit les autorisations nécessaires d'une ressource à être exécuté. Les autorisations sont basées sur l'état de la ressource reflétée en utilisant son ***R-Contexte***.

2. *Une politique du déploiement pour la restriction* consiste à prévenir un service Web engagé dans une exécution sur une ressource (over a resource). Une restriction n'implémente pas une permission négative de déploiement, plutôt elle s'occupe de la permission positive de déploiement.

4.6.2.3 traitements des exceptions

Le motif d'étudier le traitement des exceptions dans l'approche de politique de la *figure 4.2* est le risque de modification de la politique. Ils considèrent les politiques comme des ressources qui peuvent être soumises aux attaques comme toute autre ressource exposées aux partis externes. Dans leurs approche de politique, les méthodes ou les moyens de traitement des exceptions examinent les résultats de politiques et prenant actions appropriées pour consulté les quatre niveaux de la *figure 4.3* de manière hiérarchique. Une exception pourrait avoir lieu à un des niveaux suivants: *niveau ressource, niveau Sémantique, niveau Composite et niveau Composant.*

4.6.3 Une approche pour la sélection d'un flux de services en considérant trois niveaux de compatibilité (syntaxiques, sémantiques et politique)

Dans cet article [102], les auteurs démontrent comment une telle composition peut être accompli pour former un ensemble cohérent de flux de service (flow), en utilisant des règles et des services exprimés comme une base de connaissances et une ontologie. La description des règles avec des concepts de sujet permet au système d'identifier facilement les règles appropriées dans certain domaine et d'identifier et sélectionner les services Web appropriés pour la composition. Ils concédèrent des types différents de règles de composition syntaxique, sémantique et pragmatique (Contextuel), qui jouent un rôle majeur dans la découverte et la sélection des services Web. Ils modélisent La connaissance des règles et le sujet d'ontologie en utilisant les standards ***OWL***, ***DAML - S***, ***RuleML*** et ***RDF***.

4.6.3.1 Vue d'ensemble de notre approche pour la composition de services

Un service composite se compose de ces services composant de chaque organisation. Cependant, la composition de service exige d'abord de choisir les services qui sont *compatibles*. Nous considérons trois niveaux de compatibilité de service à afin de faire une compositions sensible : compatibilités de niveau syntaxiques, sémantiques et de politique. Sous la compatibilité syntaxique, la découverte et la sélection de service vérifieront si un service est compatible en ce qui concerne les spécifications, comme le ***matching*** des sorties (output) d'un service avec les entrées (input) du service suivant. D'autre part, la compatibilité sémantique considère des directives spécifiques au domaine et contrôle si la composition de service suit la combinaison spécifiée (indiquée) dans une procédure d'opération standard. La compatibilité de niveau de politique exige que les politiques et les règles, que chaque compagnie impose, doivent être compatibles avec les politiques d'autres services. La compatibilité de niveau de politique considère à la fois les politiques d'organisation internes aussi bien que les règles régulatrices qui sont imposées aux procédures de gestion de niveau Business.

4.6.3.2 Les règles de politiques pour la composition

Ces règles et politiques régissent et gouvernent la composition de services, par la restriction de certaines combinaisons de services ou par l'imposition des contraintes sur le choix d'un service sur les autres services.

Ils ont défini trois niveaux des règles à considérer :

Syntactic (operational) Rules

:La composabilité de services dépend des conditions préalables, des entrée et des exigences de sorties. Par exemple, la sortie du service Web *S1* doit être compatible avec l'entrée du service *S2*.Les règles syntaxiques assurent que la composition des services rapporte un service composite opérationnel ou syntaxiquement possible.

Semantic Rules

La composition de service devrait se faire selon des règles et des standards de procédures du Business. Les règles sémantiques de composition exigent souvent la connaissance experte de domaine, telle que : les lois du commerce, les législations d'état, les règlements environnementaux fédéraux, les politiques formelles de compagnie, les procédures expérimentales, etc. Ces règles imposent la connaissance d'expertise dans un domaine. Ainsi, l'application des contraintes sémantiques (règles) qui jouent un rôle significatif dans le composabilité de service.

Policy rules

Dans la vie réelle, c'est souvent le cas, plusieurs services possèdent le même profil et fournissent les mêmes fonctionnalités. La sélection automatisée d'un service parmi ces services fonctionnellement équivalents peut exiger la compatibilité de niveau de politique. Le service et la sélection du fournisseur sont en plus limités par les exigences de la compatibilité de politiques. Plusieurs fournisseurs peuvent offrir les services, mais le système doit trouver les fournisseurs ou les politiques sont compatibles.

4.6.3.3 La sélection du fournisseur de service basée sur la politique

Les auteurs ont développé un algorithme de composition du workflow automatique et un prototype dans le domaine E-government**.** Dans le domaine E-gouvernement, les services découverts basés sur les règles sont uniques.

Dans le domaine du commerce électronique, les services avec les mêmes fonctionnalités peuvent être fournis par beaucoup de fournisseurs du service. Le problème est de sélectionner un service pour la composition. Leur approche est de générer automatiquement un service flow (flux de service) avec les ensembles du service de candidat, $S = \{S_1, S_2,\ldots\}$ et D, un ensemble de relations de dépendance entre les services dans S. Chaque service candidat S_i dans S est un ensemble de services fonctionnellement équivalents, $S_i = \{s_1, s_2,\ldots\}$ où Chaque s_j dans S_i est un service fonctionnellement équivalent à s_k, mais le s_j et s_k sont fournis par des fournisseurs différents et implémente des règles de politique différentes. Pour sélectionner un service approprié s_j de chaque S_i, ils utilisent la *compatibilité compositionnelle* au niveau syntactique, sémantique et au niveau politique entre s_k dans S_i et s_l dans S_j.

La compatibilité syntaxique contrôle les entrées sorties et les prés conditions entre les services individuels de différents ensembles. Quand ils sont compatibles, nous dirons que les services sont *syn-compatible,* noté par $\mathbf{s_i \text{ ¤ } s_j}$. La compatibilité sémantique permet de vérifier la qualité de service qui est nécessaire dans le domaine. Les services sémantiquement compatibles sont *sem-compatible,* noté par $\mathbf{s_i \, Y \, s_j}$. La compatibilité de niveau politique assure les règles imposées par un organisme, pour permettre à ses services d'être composable avec d'autres services de politique d'organisation. Les services compatibles de politique sont appelés *p-compatible*, noté $\mathbf{s_i \, Z \, s_j}$. Cette compatibilité vérifie les relations de politiques et des règles, pour filtrer les services incompatibles dans le service composite.

Les services sélectionnés dans chaque ensemble $\mathbf{S_i}$ sont considérés comme des services compatibles pour la composition.

Un services $\mathbf{s_i}$ est *compatible (composable) avec* $\mathbf{s_j}$ *, noter par* $\mathbf{s_i \sim s_j}$ **si** $\mathbf{s_i \, Y \, s_j}$, $\mathbf{s_i \text{ ¤ } s_j}$ **et** $\mathbf{s_i \, Z \, s_j}$

4.6.3.4 Architecture du Système

La figure suivante illustre les composants de l'architecture du système de leur approche pour le système de composition de services basé sur la politique.

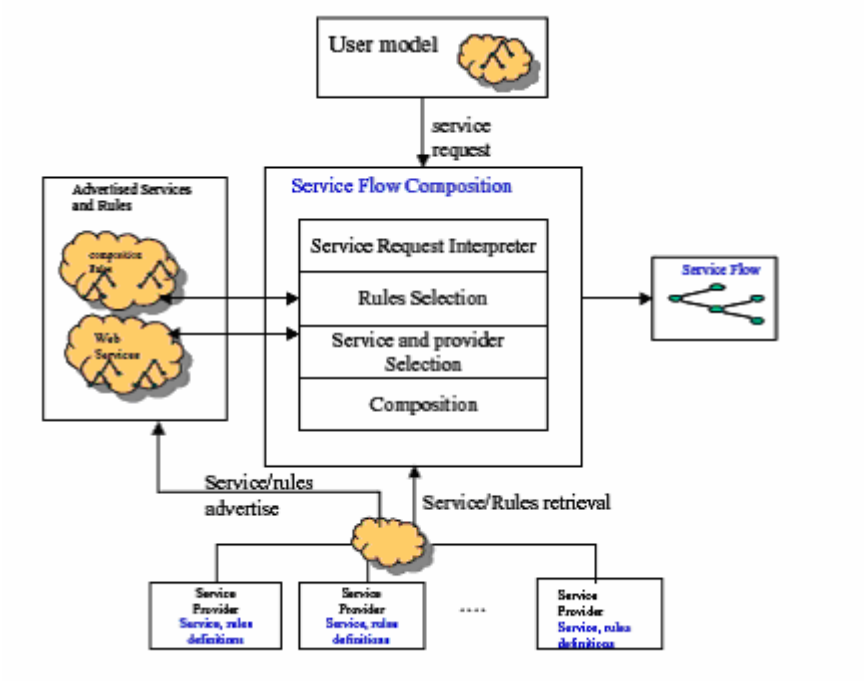

Figure 4.6 − les composants du système pour la composition automatique du service flow

4.6.4 Un formalisme pour la vérification de compatibilité de politiques

Les auteurs de cet article [110], discutent le problème de la compatibilité entre les politiques de services Web, qui pourra empêcher la composition automatique de ces services Web des difficultés de progression. Les politiques règlent et contraignent le fonctionnement de services Web à trois niveaux identifiés par le business, comportement et la confidentialité. Ils examinent un autre problème, qui est l'hétérogénéité *des politiques* qui cadre le développement et le déploiement de la composition de services Web. Brièvement, une politique peut être utilisée pour renforcer les aspects spécifiques de composition tels que le moment ou quand solliciter un service Web, quoi soumettre à un service Web, et comment substituer un service Web avec un pair. L'objectif est d'attendre une meilleure compatibilité des politiques au temps d'exécution. L'hétérogénéité concerne, non seulement les données sémantiques mais également la *consistance de politique (la cohérence)*, quand la collaboration se produit entre des services Web indépendants de différentes origines.

Dans ce travail, les auteurs se concentrent sur trois types de politiques à savoir :

1. Une politique de ***business*** qui reflète les contraintes qui vont avec la logique du business qui supporte la fonctionnalité d'un service Web (par exemple, la réservation d'hôtel doit être confirmée dans les 5 journées de travail dans la semaine).

2. Une politique de ***bihavior*** qui (comportement) reflète l'attitude d'acceptation ou de rejet d'un service Web pour participer à une composition de service.

3. Une politique ***privacy*** qui reflète la manière dont un service Web protège l'information, qu'elle reçoit (resp., envoie) des (resp.aux) utilisateurs et des pairs, contre l'abus.

4.6.4.1 L'approche de compatibilité de politique

Deux services $Web_{i,j}$ sont montrés et qui forment un service Web composite. Chaque service Web est associé à trois types de politiques: *business, Bihavior et Privacy* (l*a figure 4.7).*

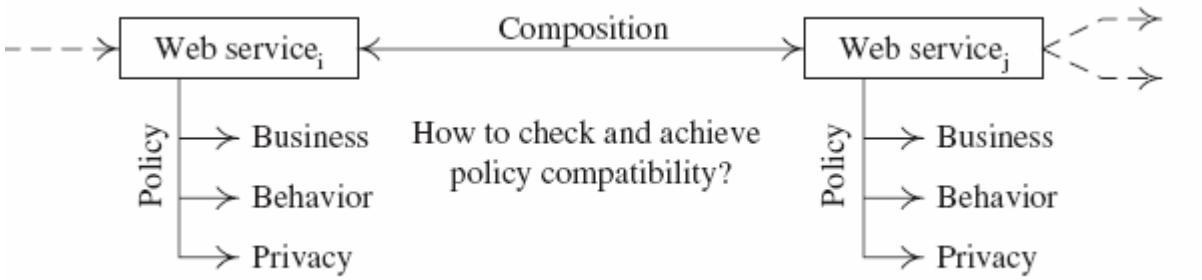

Figure 4.7 − Les types de politiques et les services Web

Business policy *(politique de Business)* est une abstraction des contraintes au-dessus d'un processus de business. Ces contraintes règlent la manière dont ce processus de business se réalise selon (i) les besoins et les exigences de l'utilisateur (ii) le règlement interne de l'organisation.

Privacy policy *(politique d'intimité)* est une abstraction d'un ensemble de mécanismes qui règlent la manière dont un service Web manipule les données qu'il reçoit, stocke et soumet. Une autre définition est suggérée par ***Kagal et autres*** qui utilisent la politique d'intimité pour

indiquer sous quelles conditions l'information peut être échangée et les utilisations légitimes de cette information [103].

Bihavior policy *(politique de comportement)* est une abstraction de l'attitude qu'un service Web expose vers des propositions qu'il reçoit concernant des participations potentielles aux compositions. Dans la recherche précédente [109], cette attitude a été illustrée avec des attributs de permission, restriction et de dispensation. *(permission, restriction and dispensation attributes.)*

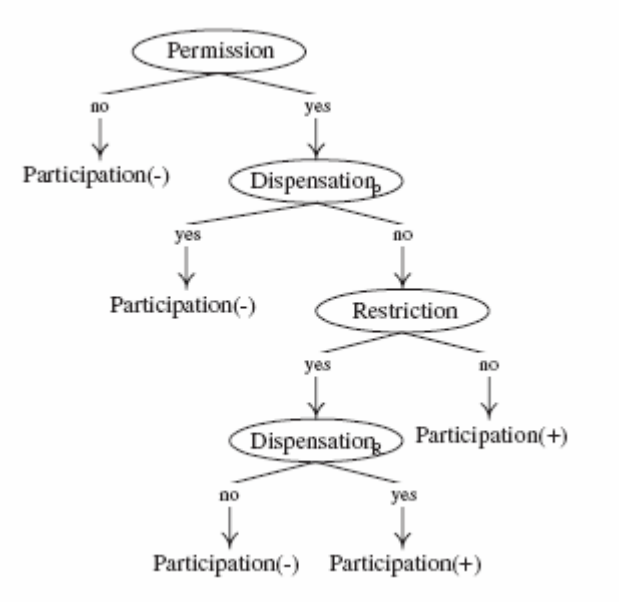

Figure 4.8 − Les attitudes du service Web

La chronologie des opérations dans la *figure 4.8* se produit comme suit. Initialement, un service Web est invité à participer à un service Web composite. La décision d'accepter ou de rejeter l'invitation est fait sur les résultats d'exécuter la politique de comportement relatif à la permission. Si on accorde une permission, la prochaine étape est de vérifier s'il y en a une dispensation qui pourrait infirmer cette permission. Si les résultats d'exécuter la politique de comportement relatif à la *dispensation*$_P$ est positive, ceci aura comme conséquence l'annulation de la permission positive de participation. La prochaine étape est de vérifier s'il y a n'importe quelle restriction qui pourrait empêcher le service Web de participer à ce service Web composite. Si les résultats de l'exécution de la politique de comportement relier à la restriction est négative, la permission positive initiale de la participation est finalement approuvée. Autrement et comme des dispensations pour les permissions, des dispensations qui annulent des restrictions sont vérifiées comme rapporté précédemment en utilisant la-politique de comportement *relatif a la dispensation*$_R$.

Spécifications des politiques

Le WSPL (Web Services Policy Language) est utilisé pour la spécification de politiques conformément aux arguments d'Anderson [101].

4.6.4.2 Compatibilité par type de politique

Dans leur approche pour traiter la compatibilité de politique entre les services Web, les auteurs se basent sur la notion de la violation, c.-à-d., les politiques de deux services Web sont considérées compatibles s'il n'y a pas de violation détectée. Ils ont proposé un formalisme pour la vérification de la compatibilité de tous les trois types de politiques considérées entre deux services.

4.6.5 Une approche en Quatre couches

Les auteurs de ce papier [111], examinent le contexte et la politique pour gérer les comportements que des services Web exposent lors de la composition et en réponse aux changements dans un environnement. À cet effet, une approche de 4 couches est conçue. Ces couches sont dénotées par : la politique, l'utilisateur, le service Web et la ressource. Dans cette approche, la gestion du comportement concerne l'exécution des politiques de types *permission, obligation, restriction et dispensation.* Un prototype qui illustre comment le contexte et la politique sont tissés dans des scénarios de composition de services Web, est présenté dans ce travail. Et ils suggèrent d'utiliser des politiques qui permettent en premier, le contrôle de la participation des services Web dans des scénarios de composition et en second lieu, la gestion du cycle de vie de ces scénarios. Ce cycle de vie s'étend sur plusieurs étapes, y compris la découverte de services Web, la sélection, le suivi, etc. Les auteurs discutent et illustrent dans le présent document leur proposition de l'approche en 4 couches pour le développement du comportement des services Web basés sur le contexte, en utilisant des politiques *(*figure 4.9*).*

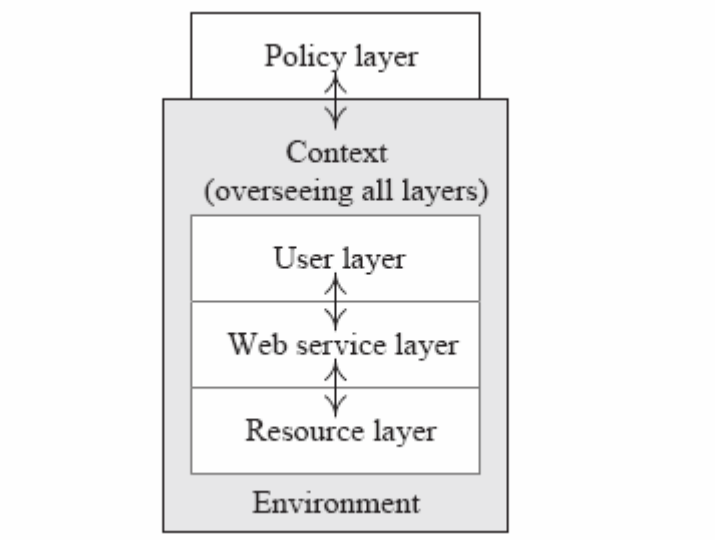

Figure 4.9 − l'approche en Quatre couches pour le comportement des services Web

Dans le haut niveau Existe la couche politique, qui se base sur les informations que le contexte renvoie à partir de reste des couches, à savoir la couche utilisateur, service Web et ressource. Les politiques sont déclenchées en fonction des changements dans les préférences des utilisateurs, les Etats des services Web et les engagements des ressources.

4.6.5.1 L'approche de 4-couche : description des quatre couches

La couche ressource : représente *the computing means* sur les quels les services Web fonctionnent. L'ordonnancement des ressources en raison d'exécution des requêtes concurrentes de services Web, qui se produit lorsque les ressources ne sont pas suffisamment disponibles pour satisfaire toutes ces requêtes à la fois. Pour appuyer et superviser cet ordonnancement, une ressource est associée à un ensemble d'arguments.

La couche service Web : représente les services Web qui sont affichés sur les différents enregistrements, comme UDDI, afin que les utilisateurs puissent les identifier en fonction de leurs besoins et les exécuter après. Cette exécution est en conformité avec la disponibilité des ressources, comme indiqué dans la couche ressource. Comme une ressource, un service Web est associé à un ensemble d'arguments.

La couche utilisateur : représente les utilisateurs à la recherche de services Web, en respectant leurs besoins qui varient de la planification du déplacement à la réservation de la commande. Les utilisateurs peuvent nécessiter une assistance dans la localisation, la sélection et la composition du service Web approprié. Comme une ressource et un service Web, un utilisateur est associé à un ensemble d'arguments.

La couche politique : représente les politiques qui définissent comment un service Web doit réagir sur la base des progrès d'une composition. Ce progrès est au sujet d'interactions d'un service Web avec les utilisateurs et les ressources. L'interaction entre les utilisateurs et les services Web concernent la manipulation et la validation des préférences des utilisateurs. L'interaction entre les services Web et les ressources concernent les mécanismes qui supportent l'exécution des services Web personnalisés sous les contraintes des ressources.

4.6.5.2 Définition des comportements de services Web

La *figure 4.10* illustre la façon dont les différents comportements d'un service Web sont reliés entre eux en se basant sur le résultat de l'exécution des politiques.

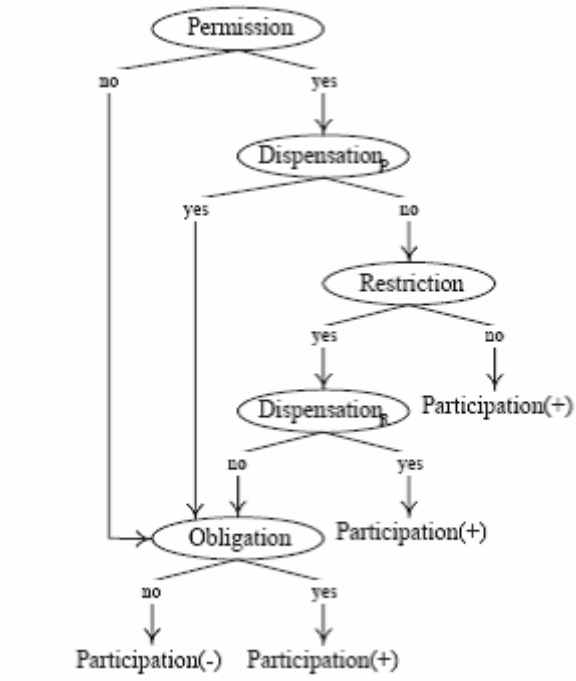

Figure 4.10 − les services Web liés aux comportement

Quatre types de comportement ont été définis: permission, restriction, dispensation et obligation. Les fournisseurs de services Web sont responsables de spécifier les politiques par type de comportement.

1. ***Permission*** : un service Web accepte l'invitation de participation à un service Web composite, sur validation de ses engagements courants dans d'autres Services Web composites.

2. ***Restriction*** : un service Web ne peut pas faire partie d'un service Web composite pour cause du non-respect de certains services Web composants dans ce service Web composite avec les exigences non fonctionnels de ce service Web.

3. ***Dispensation*** : signifie qu'un service Web interrompe, soit une autorisation ou une restriction de la participation à une composition.

4. ***Obligation*** : c'est une forte permission pour un service Web de participer à un service Web composite, en dépit d'une permission négative de la participation ou de l'existence de restrictions de la participation.

Dans cette illustration, les comportements sont représentés avec des ellipses et des liens qui sont étiquetées par oui ou non. Dans la même figure, dispensation_P et dispensation_R désigne la dispense relative à la permission et la restriction, respectivement. En outre, la participation (+) et participation (-) désigne respectivement la participation positive et négative dans une composition.

4.6.5.3 Les politiques au cours des interactions

Dans une composition de services Web, les interactions sont classées que ce soit verticale (entre le service Web composite et ces services Web composants) ou horizontale (entre les Services Web composants dans le même service Web composite).

Ici, un service Web composite a l'autorité de mener les actions suivantes sur un service Web composant *(Figure 4.11)*: *Invite ,Ping ,Trigger ,Audit et retract*

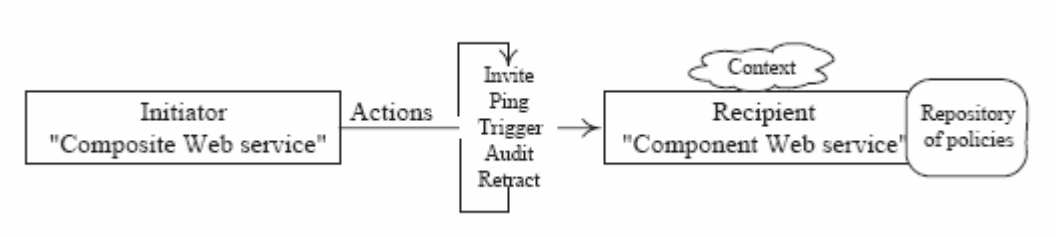

Figure 4.11 − les interactions verticales durant la composition.

Quand un service Web composite décide d'exécuter les actions ***"Invite", "Ping", "Trigger", "Audit***", ou ***"Retract"***, le service Web composant concerné vérifie son contexte avant le déclenchement des politiques et de répondre au service Web composite. Dans ce papier, les auteurs se concentrent sur les politiques liées à la participation dans une composition par le biais de l'action ***"Invite".*** Ici, un service Web composant a l'autorité de mener les actions suivantes sur un autre pair dans la même composition (*Figure 4.12*): ***Ping, Trigger***r.

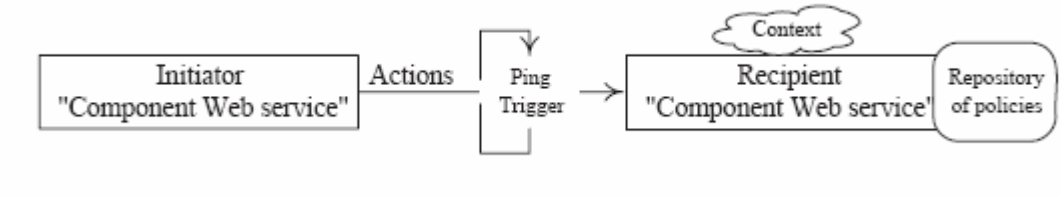

Figure 4.12 − les interactions horizontales durant la composition.

Tout comme le rôle des politiques dans les interactions verticales, les actions dans les interactions horizontales résultent par la vérification du contexte et par le déclenchement de politiques.

4.6.6 Autre proposition et travaux

Z. Maamar et al [112] Listent les six différents scénarios qui concernent les utilisations possibles des politiques dans les services Web, à savoir la publication, la sélection, la compatibilité, l'accord, la vérification et à la composition. En outre, Maamar et al, discutent de la façon dont l'absence d'un cadre normalisé pour l'amélioration de services Web, grâce à des politiques est actuellement équilibrée avec deux approches, à savoir le Web sémantique et les combinaisons booléennes de prédicats de politique.

Y. Li et al [113] discutent la vérification de la consistance formelle entre les processus *BPEL* et les politiques de confidentialité. Les auteurs arguent cela en dépit des soucis accrus de l'intimité à travers l'Internet, peu a été fait quand il vient à imposer des politiques d'intimité dans les entreprises qui collectent et consomment des données personnelles en utilisant des moyens automatiques comme les services Web. Comparé à un " code de conduite ", une politique d'intimité est souvent codée en utilisant la plateforme pour des spécifications de préférences d'intimité. Bien que le travail de Li et al n'est pas concentré, explicitement, sur la compatibilité de politique d'intimité (privacy policy compatibility), ils adressent un autre aspect de compatibilité : *combien les opérations du businesss répondent à des exigences d'intimité. (how much business' operations meet privacy requirements)*

L. Kagal et al [103] déclarent que les politiques devraient faire partie de la représentation de services Web et en particulier de services Web sémantiques. Les Politiques fournissent la spécification de qui peut utiliser un service et sous quelles conditions, comment l'information devrait être fournie au service, et comment l'information fournie sera utilisée en suite.

L. Baresi et al [114] proposent une approche basée sur la politique pour surveiller les services Web fonctionnel (par exemple, contraintes sur les données échangées) et les exigences non fonctionnelles (par exemple, sécurité, fiabilité). Dans cette approche *Baresi et al* font un rapport sur les différents types de politiques qui peuvent être définies le long du cycle de vie d'un service Web. Ces types sont des : politiques du service, politiques du serveur, politiques supportées et politiques demandées (sollicitées).

Z. Maamar et al [115] développent la consistance, faisabilité et l'inspection de politiques pour conduire la personnalisation des services Web. Pour le but de l'illustration, ils ont fait une vue d'ensemble sur la consistance de la politique qui garantie qu'un service Web fait exactement ce qu'il est supposé faire avant sa personnalisation. La personnalisation peut changer (modifier) la spécification initiale d'un service Web, quand il vient à la liste d'événements qui déclenchent ce service Web. Le temps et les paramètres d'emplacement relatif sont de nouveaux événements qui ont besoin d'être ajouté à la liste d'événements réguliers du service Web.

Foster et al [117] utilisent des représentations de machine à états finis de l'orchestration de service Web et assignent la sémantique aux interactions de processus distribuées. Mais cette méthode est principalement appropriée pour le processus de vérification. Le comportement interne de chaque service Web n'est pas pris en compte.

Bordeaux et al [118] utilisent les ***LTS*** pour formaliser le comportement des services Web et proposent quelques définitions de compatibilité. Donc, cette vérification de compatibilité peut largement être automatisées. Toutefois, leur méthode est trop simple pour faire face à des situations complexes.

Fu et al [92, 93] proposent une approche (top-down) descendante basée sur les automates pour analyser la composition des services Web ; cette approche peut effectivement garantir une composition de services Web juste, mais similaire à [117]. Le comportement de l'intérieur de chaque service Web n'est pas pris en compte.

Pour surmonter les lacunes de [117, 118, 92, 93]**,** *Y. Shi et al* [116] proposent une nouvelle approche pour vérifier la compatibilité des services Web dans des situations complexes. Le point essentiel, c'est l'idée de rôle dans l'interaction. Les auteurs estiment que l'analyse de la compatibilité est absolument nécessaire pour garantir une composition statique ou dynamique correcte de services Web. Ils proposent un formalisme du comportement des services Web en utilisant la méthode des automates. Ils proposent une définition du rôle des interactions entre les services Web. En conséquence, il y aura la vérification si deux ou plusieurs services Web sont compatibles en collaboration ou non.

4.7 Conclusion

Dans cette partie, nous avons abordé la définition, l'utilisation et les origines des politiques dans la gestion des comportements des systèmes complexes automatiques, particulièrement dans les services Web. Nous avons aussi évoqué quelques travaux qui ont discuté la compatibilité de politiques dans la composition de services Web.

Beaucoup de travaux traitent la composition en tant que processus qui vise à décomposer la tâche globale étendue par l'utilisateur, en sous tâches qui peuvent être réalisés chacune par un

service Web. Néanmoins, chacune des sous tâches peut être réalisées par plusieurs services Web ; nous disons alors, ils sont fonctionnellement équivalents.

Ainsi, un processus de sélection d'un service adéquat, par un ensemble de services fonctionnellement équivalent, doit être mis en œuvre. Ce processus est appelé communément, la composition horizontale.

Dans le chapitre suivant, nous allons présenter une nouvelle approche qui se base sur l'approche CSP pour la recherche d'un flux de service Web compatible.

Chapitre 5 :

CSP4WSC : Une approche CSP pour la mise en œuvre de la compatibilité de politique dans la composition de services web.

Contribution

Chapitre 5

CSP4WSC : Une approche CSP pour la mise en œuvre de la compatibilité de politique dans la composition de services web.

5.1 Introduction

Nous assistons de plus en plus, aujourd'hui, à l'utilisation de politique pour décrire les contraintes d'utilisation d'un service Web; ce qui implique la nécessité de vérification de la compatibilité de politique dés qu'il s'agit d'interagir deux services Web ou une composition de services Web.

Le grand succès des services Web est dû, essentiellement, à la richesse des applications rendues possible grâce à des standards communs ouverts. Néanmoins, leur prolifération a rendu les tâches de découverte, de recherche et d'utilisation de services appropriés ardues. Ces tâches sont d'autant plus difficiles lorsqu'il s'agit de services Web composites en réponse à des besoins complexes de l'utilisateur. La composition automatique de service Web implique la recherche d'une combinaison de service Web existant pour réaliser la tâche globale attendue par l'utilisateur. Cependant, chaque sous tâche peut être réalisée par plusieurs services Web, ce qui introduit l'importance d'un processus de sélection de service Web approprié en tenant compte de plusieurs contraintes : telle que la compatibilité de politiques (comportement, contrôle d'accès , confidentialité , business, sécurité….).

Dans ce chapitre, nous étudions le problème de mise en oeuvre de la compatibilité de politique dans le cadre de la composition de services web.

5.2 Vue générale de la composition basée sur les politiques

5.2.1 La composition et les politiques des services

Souvent plusieurs services possèdent le même profil et fournissent les mêmes fonctionnalités. La sélection automatisée d'un service, parmi ces services fonctionnellement équivalents, pour participer dans une composition peut exiger la compatibilité de niveau politique. Le service et la sélection du fournisseur sont en plus limités par les exigences de la compatibilité de politiques.

Donc Plusieurs fournisseurs peuvent offrir des services, mais le système doit trouver les fournisseurs ou les politiques sont compatibles.

5.2.2 Les politiques considérées

Nous rencontrons trois types de politiques dans les composants du système pour la composition automatique du service Web composite.

- ***Les politiques de service*** : sont définies par les différents organismes offrant des services.
- ***Les politiques de service Web composite*** sont définies par les organismes offrant les services Web composite comportant des services Web composants.
- ***Les politiques d'utilisateur*** sont définies par les consommateurs des services. (C'est les préférences, contraintes et buts qui sont considérés en tant que politiques spécifier par l'utilisateur).

5.2.3 Les services Web Composants

Chaque service Web s_i est composé d'un ensemble d'opérations qu'il effectue, et est associé à un ensemble de paramètres d'interface. Il est également associé à un ensemble de politiques comment et quand le service est applicable, ou quelles conditions doivent être satisfaites pour qu'il soit appelé.

Un service est une instance d'un service dans l'ontologie de service Web. Celui la peut être indiqué comme une application capable d'être défini et localisé par l'intermédiaire de Protocoles Internet, capable d'agir avec d'autres applications et peut être identifié par une (URI). Chaque service est énoncé avec un ensemble de contraintes en utilisant des descripteurs du type de service, appelés l'ensemble de politiques de service, qui sont implémentés par le fournisseur de service afin d'offrir le service.

Les services composants sont fournis par des organismes avec des politiques bien spécifiques, les besoins du service Web composite est de choisir les services composants appropriés qui sont compatible avec les politiques d'utilisateur. Le service choisi qui est conforme aux politiques s'appelle une **instance de service** (ou service instancier).

5.2.4 Les services Web composites

Le service composite est composé de multiples services Web composants et d'un ensemble de politiques qui doivent être satisfaites. Nous utilisons **P (SC)** pour dénoter l'ensemble de politiques liées à un service composite. Les politiques du service composite définies ci dessus différent de celles de service. Les politiques de service composite s'appliquent entièrement au service composite comportant de multiples services.

5.3 CSP : les problèmes de satisfaction de contraintes

Les problèmes de satisfaction de contraintes (CSP : Constraint Satisfaction Problem) sont au coeur de nombreuses applications en Intelligence Artificielle et en Recherche Opérationnelle. Un CSP est un formalisme à la fois simple et puissant permettant de représenter et de résoudre un grand nombre de problèmes. Un CSP se définit comme un ensemble de contraintes impliquant un ensemble de variables. L'objectif est de trouver une

valeur pour chaque variable afin de satisfaire l'ensemble des contraintes ou bien de satisfaire le maximum de contraintes. La recherche dans le domaine des CSP est motivée par la découverte de méthodes toujours plus efficaces en pratique pour les résoudre. Mais, comme les CSP font partie de la classe des problèmes combinatoires NP-Complets, les algorithmes connus pour les résoudre nécessitent un temps exponentiel en fonction du nombre de variables. La méthode la plus employée pour les résoudre est la recherche énumérative qui consiste à explorer systématiquement un arbre de recherche.

5.3.1 Définitions de base

Définition 1 : (Dûe à Montanari **[119]**)

Un problème de satisfaction de contraintes (***CSP***) est un quadruplet (X, D, C, R) où:

• $X = (x_1, x_2,\ldots, x_n)$: est un ensemble de n variables ;

• $D = (D_1, D_2,\ldots, D_n)$: est un ensemble de n domaines.

• $C = (C_1, C_2, \ldots, C_m)$: est un ensemble de m contraintes

• $R = (R_1, R_2, \ldots, R_m)$: est un ensemble de m relations. Chaque relation R_i définit l'ensemble des n_i-uplets sur $D_{i1} \times D_{i2} \ldots \times D_{ini}$ autorisés par la contrainte C_i.

Définition 2 : Une ***contrainte*** est une relation logique ou une propriété devant être vérifiée par un sous ensemble de variables. Une contrainte permet de restreindre les valeurs pouvant être prises simultanément par un ensemble de variables. Ainsi, une contrainte est définie par un ensemble de variable et la relation entre ces variables

Définition 3 : une ***affectation*** ou une instanciation est un ensemble de couples (variable, valeur) exprimant l'association des valeurs aux variables.

Définition 4 : Une ***solution*** d'un CSP est une affectation totale consistante, c'est à dire une assignation d'une valeur à chaque variable de X tel que toutes les contraintes soient satisfaites.

5.3.2 Les algorithmes de résolution

Le but majeur de la résolution d'un CSP est de trouver une ou toutes les solutions, c'est-à-dire, des affectations totales consistantes, ou d'affirmer l'inexistence d'une solution. Néanmoins, d'autres objectifs peuvent être attendus. Dans le cas où le CSP est **sur-contraint** (inexistence de solution), l'objectif peut être de trouver l'affectation totale qui viole le moins de contraintes possibles; dans ce cas en parle de **max-CSP.** Plusieurs algorithmes de résolution de CSP existent :

- ✓ Algorithme Backtrack (BT).
- ✓ Algorithme Back Jumping (BJ).
- ✓ Algorithme Forward Checking (FC).
- ✓ Algorithme Maintenance de consistance d'arc (MAC).
- ✓ …….

5.3.3 Algorithme Backtrack (BT)

Cet algorithme est l'algorithme de base pour la résolution des CSP. Il consiste à instancier successivement les variables et à vérifier le viol d'une contrainte, dès que toutes les variables d'une contrainte ont été instanciées. Si une contrainte est violée l'algorithme affecte une nouvelle valeur à la dernière variable jusqu'à trouver une valeur valide ou que son domaine devienne vide. Dans ce dernier cas, l'algorithme effectue un retour vers l'avant dernière variable (backtrack) et lui affecte une nouvelle valeur.

Algorithme 5.1 : Algorithme Backtrack (BT)

```
Algorithme Backtrack (A, (X, D, C, R)) : booléen   // A est une affectation, au premier appelle A = ∅
        Si non consistant(A) Alors Retourner faux Fsi
        Si A est une affectation totale alors retourner vrai   /* A est une solution */
        Sinon choisir une nouvelle variable Xi de X
          Pour toute valeur Vi de D(Xi) faire
                Si Backtrack(AU{(Xi,Vi)},(X-{Xi}),{D(X1),…,D(Xi)-{Vi},…,D(Xn)},C,R) = vrai
                alors retourner vrai Fsi
          Fin pour
        Fsi
        Retourner faux
Fin
Programme principal
        Backtrack (∅, (X, D, C, R))
Fin
```

L'inconvénient majeur de cet algorithme est le fait de faire des retours en arrière simples ou chronologiques, lors de la découverte d'une inconsistance. En effet, la variable précédente peut ne pas être la cause de l'inconsistance.

5.4 Modèle de compatibilité de service

Les protocoles ou les règles sur la façon dont se composent des services valides, sont indiqués comme un ensemble de *règles générales de compatibilité*. Les règles de compatibilité se réfèrent aux aspects *de politiques* de services. La compatibilité de niveau politique assure que les règles imposées par une organisation permettent à son service d'être composable avec un autre service d'une autre organisation qui possède une politique spécifique.

Ces contrôles de compatibilité vérifient les relations des politiques et des règles, pour filtrer tous les services incompatibles dans un service composite. Pour s'assurer que ces compatibilités de niveau politique sont vérifiées, un ensemble de règles compositionnelles de vérification sont utilisées.

Les types de règles de composition du modèle présenté, sont les suivants :

1. Satisfaction des politiques de l'utilisateur

L'une des règles de la composition déclare que si les contraintes de l'utilisateur indiquées (spécifiées) au dessus de s_i sont toutes réunies avec la politique de service fournie par un fournisseur de service, donc le service peut être ajouté (c'est-à-dire le service est candidat).

2. La compatibilité verticale

C'est la compatibilité entre un service Web composite et tous ses services Web composants. En d'autres termes, les politiques de services Web qui vont se composer doivent être compatibles avec les politiques de l'organisme offrant le service Web composite.

3. La compatibilité horizontale

C'est la compatibilité entre les services Web composants. En d'autres termes, les politiques d'un service Web composant sont compatibles avec les politiques d'un autre service si ils sont en interaction directe dans la composition.

Le processus de composition de service Web à base de politiques comprend les étapes décrites ci-dessous et illustrées dans la *figure 5.1*.

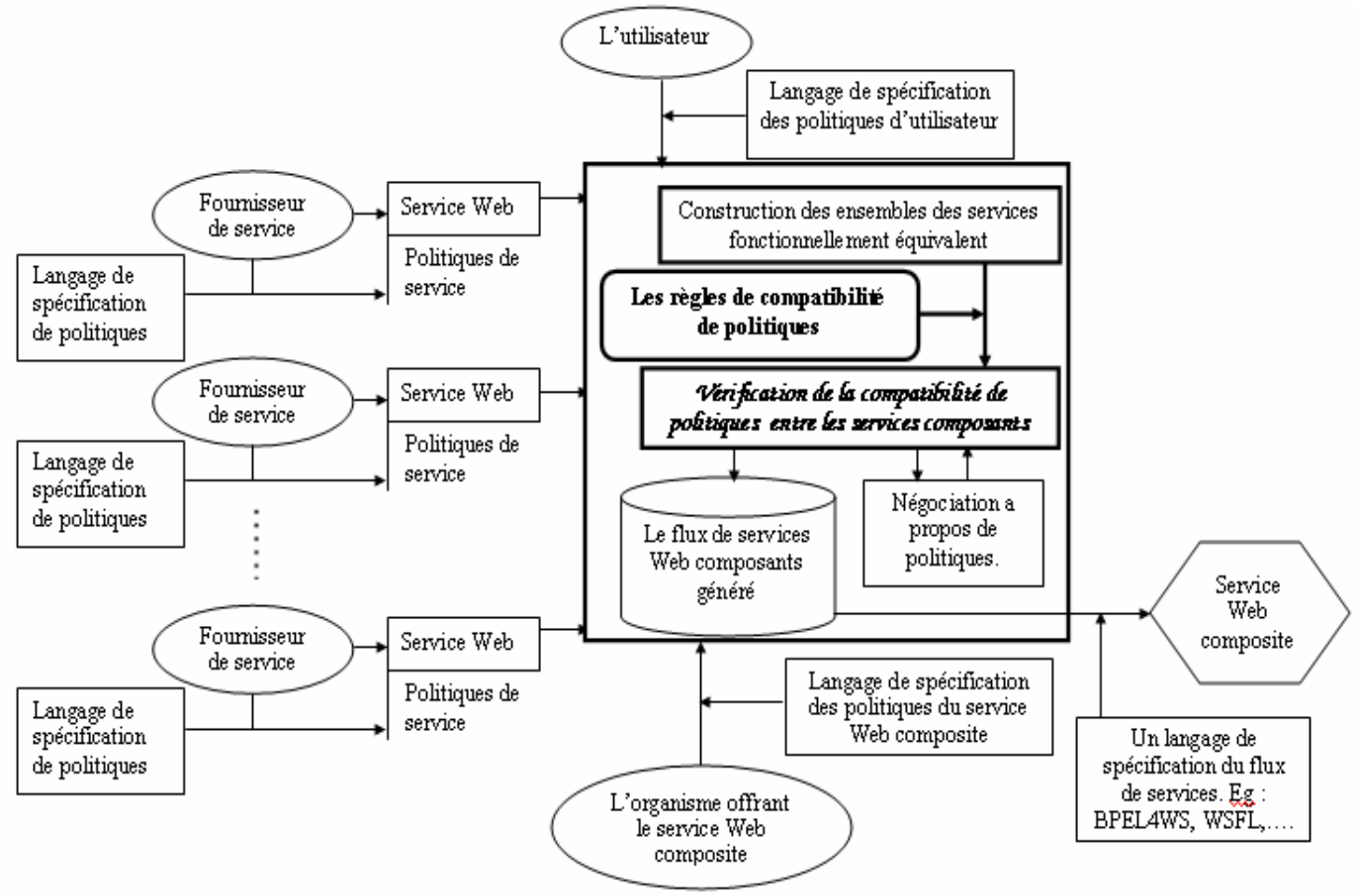

Figure 5.1 − Vue générale de la composition basé sur les politiques

La figure ci-dessus comporte les étapes suivantes :

1. *spécification des besoins de l'utilisateur : E*n utilisant un langage de spécifications de politiques, l'utilisateur exprime ces *préférences, exigences et contraintes* sur le service dont la requête est mise.

2. *spécification des politiques de l'organisme du service Web composant : C*haque service Web est associé à un ensemble de politiques qui sont définies par l'organisme offrant ce service, comment et quand ce service est applicable ? Ou quelles conditions doivent être satisfaites pour qu'il soit appelé ?

3. *Décomposition de la tâche globale* **:** la requête ou la tâche globale de l'utilisateur qui ne sera pas satisfaite par un seul services, sera automatiquement décomposée en sous tâches (service Web abstraits) ou chacune d'entre elle sera réalisée par un services Web concret.

4. *Correspondance de politiques d'utilisateurs et les politiques de service* : tous les services, dont les politiques correspondent aux politiques de l'utilisateur, sont mis dans une liste de candidat.

5. *choisir un service à partir d'une liste de candidats* **:** quand il y a des paires ou une séquence des services des différentes listes de candidat qui sont composable selon les règles compositionnelles (règles de composition), alors une de ces paires ou séquence des services sont choisis. Les règles compositionnelles éliminent les services non composables dans les listes de candidat. S'il y a seulement un service dans la liste de candidats qui correspond à la politique locale de l'utilisateur, alors le choix du service est unique.

6. *Vérification de la compatibilité des services sur tous les niveaux* : quand il y a des politiques globales, la combinaison des services choisis doit correspondre aux politiques globales de flux service afin d'être une composition réussie. Le processus du choix dans l'étape 4 devrait considérer des services différents des listes de candidats, pour rencontrer les politiques globales d'utilisateur.

7. N*égociation de politiques : u*ne fois la vérification de la compatibilité est terminée, si nous avons un flux de services a composer, la composition peut se lancer. Si non, le composant charger de la négociation est activé pour lancer un processus afin de négocier les politiques de l'utilisateur avec celles des services Web composant et celles du service Web composite.

8. *Composition du flux choisi* **:** après avoir choisi pour chaque sous tache un service Web concret qui va réaliser cette dernier et qui satisfait tout les niveaux de compatibilité (les politiques d'utilisateur, les politique du service Web composite, les politiques de services

Web composant avec lesquels il interagit), l'algorithme procède à la composition de ce flux compatible et offre au client un seul service qui inclut tout ces besoins.

5.5 Algorithme générale de composition a base de politiques

La *figure 5.1* est un schéma récapitulatif de la procédure générale de la composition de services Web à base de politique, cette approche est implantée par l'algorithme 5.2.

Ainsi, après avoir reçu la requête de l'utilisateur, l'algorithme recherche un service Web élémentaire qui peut répondre aux besoins de l'utilisateur. Dans le cas contraire l'algorithme construit un CSP formalisant le problème de la composition, puis l'algorithme ***Backtrack*** et appelé pour résoudre le CSP et générer ainsi le flux de services concret, comme nous allons le voire dans le paragraphe suivant.

Comp :la fonction qui vérifier la compatibilité.
FS : le flux de services composant compatible près pour être composé.
WS : service Web élémentaire.
P_U : l'ensemble des politiques de l'utilisateur.
P_{OSW} : l'ensemble des politiques de l'organisme offrant le service Web composite.
P_{SW} : l'ensemble des politiques d'un service Web.
S : l'ensemble des ensembles de services Web fonctionnellement équivalents.

Algorithme 5.2 : Algorithme général de composition a base de politiques

Entrées : P_U, P_{OSW}, P_{SW} : des politique.
Sorties : un service Web élémentaire WS ou un flux de services Web composants;
Début :
Si ∃ (WS) tel que ((P_U) Comp (P_{SW})) alors retourner (WS)
Sinon
Construct (S, D, C, R) ;
FS= ∅;
Si (Backtrack *(FS, (S, D, C, R))* = vrai**) alors composer** (FS)
Sinon négocier (P_U, P_{OSW}, P_{SW}) **;**
Finsi ;
Finsi ;
Fin.

Nous nous concentrons dans ce qui suit sur la composition horizontale définie ci-dessous pour la sélection d'un flux de services Web compatible en terme de politiques en exploitant l'algorithme **Backtrack** de l'approche CSP (*Constrait Satisfaction Problem*) pour formaliser et résoudre ce problème.

5.6 L'algorithme pour la sélection du flux de services Web composants

Dans ce qui suit, nous allons présenter notre approche de mise en œuvre de la compatibilité dans la composition de services Web en exploitant l'approche CSP. Nous aurons besoin de définir quelques concepts de base sur les services Web et la composition.

5.6.1 Service Web abstrait et concret

La tâche de la composition automatique de service Web consiste à trouver une combinaison appropriée de services Web existants pour atteindre un objectif global. Résoudre ce problème exige le matching et le mixage des services Web composants en fonction de certaines caractéristiques. Toutefois, il existe, généralement, un choix de nombreux services Web pour chaque sous tâche à faire (service Web) pour répondre au but principal. Nous dirons que ces services Web sont fonctionnellement équivalents. Dans la suite de ce document, comme ce qui est généralement fait dans la littérature, nous appelons chacune des sous tâches qui constituent l'objectif principal comme un service Web abstrait et Chaque service Web capable d'exécuter une sous tâche comme, un service Web concret.

5.6.2 Composition verticale et horizontale

La résolution du problème de la composition de services Web signifie le passage par deux types de processus de composition :

1. la composition verticale, qui vise à trouver la "meilleure" combinaison de services Web abstraits, c'est-à-dire, un workflow abstrait, pour atteindre le principal objectif, tout en satisfaisant toutes les restrictions interdépendantes existantes [94].

2. la composition Horizontale, le but est de trouver le "meilleur" service Web concret, parmi un ensemble services Web fonctionnellement équivalent, c'est-à-dire un Workflow exécutable, pour accomplir chaque service Web abstrait. La qualité de la réponse à la requête de l'utilisateur (la tâche de composition) est considérablement dépendante de la sélection du service Web concret. Le choix d'un service Web concret est dicté aux attributs fonctionnels et/ou non fonctionnels (c'est-à-dire, liés à la qualité des attributs de service) [94].

5.7 Formalisation du problème sous forme d'un CSP

Dans notre approche, nous avons opté pour le formalisme CSP afin de représenter et de résoudre le problème de composition horizontale ; basé sur la contrainte de compatibilité de politiques des services Web qui constituent le service Web composite afin de satisfaire les contraintes, les préférences et les exigences ou en d'autres terme les politiques. Chercher le services Web concret approprie parmi les services Web fonctionnellement équivalents signifie de passer par deux étapes pour la résolution du problème.

5.7.1 Formalisation de la composition

Nous modélisons la recherche d'une telle combinaison comme un problème de satisfaction de contraintes (CSP), dans lequel nous considérons une variable pour chaque tâche (service Web abstrait), dont le domaine est un ensemble de valeurs correspondant à l'ensemble des services potentiels pour cette tâche (service Web concret). Nous introduisons une contrainte entre chaque couple de services abstraits qui doivent interagir directement ; en suite pour chaque contrainte, nous construisons un ensemble de relations défini comme des couples de service *concret* compatible en terme de politique et qui réalise chaque un un service *abstrait* régit par la contrainte

Exemple

Soit une requête d'un utilisateur qui est décomposée en six (06) sous taches ou services Web abstraits qui représente chaque sous tache. La figure 5.2 nous montre un schéma

présentant l'ensemble de contraintes (compatibilité de politique) et relations entre chaque couple de services Web abstraits qui sont en interaction dans la composition.

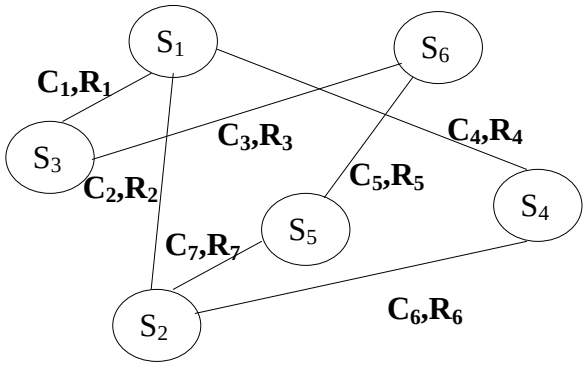

Figure 5.2 – Exemple : Schéma d'interaction entre les services Web abstrait.

Formellement, l'ensemble des variables de CSP $S= \{S_1,S_2,..S_n\}$ correspond à l'ensemble des services Web abstraits, Chaque $S_i \in S$ est un service abstrait dont le domaine est l'ensemble de services Web concrets fonctionnellement équivalents qui réalise la tâche. Ainsi $D(S_i) = \{ws_{i1},ws_{i2},.. ws_{it}\}$ tel que chaque $ws_{ij} \in D(S_i)$ est un service fonctionnellement équivalent a $ws_{ik} \in D(S_i)$, mais sont fournis par différents fournisseurs et sont implémentés par différentes règles de politique.

L'ensemble $\mathbf{C}=\{C_1,C_2,\ldots\ldots C_m\}$ correspond aux contraintes de compatibilité de politiques introduites entre chaque couple de services Web abstrait qui doivent interagir lors du processus du composition dans le plan de cette dernière.
En fin, chaque relation d'une contrainte est définie par l'ensemble des couples des services Compatibles (ws_{ij},ws_{kl}) tel que ws_{ij} est un service concret du service Web abstrait S_i, et ws_{kl} est un service concret du service Web abstrait S_k.

En résumé, le problème de composition horizontale est formalisé par un CSP P = <X,D,C,R> tel que :
$X = S = \{S_1, S_2,\ldots.....S_n\}$: l'ensemble des services Web abstraits.
$D = \{D_1, D_2..... D_n\}$: pour chaque Web services abstrait S_i, $D_i= \{ws_{i1}, ws_{i2}..... ws_{it}\}$ qui représente l'ensemble de services Web concrets qui sont fonctionnellement équivalents qui peuvent accomplir la tâche.

$C = \{C_1, C_2, \ldots\ldots\ldots C_m\}$: une contraint pour chaque couple de services Web abstrait qui interagissent.

$R = \{R_1, R_2, \ldots\ldots\ldots\ldots R_m\}$: une relation pour chaque contrainte. R_i : est l'ensemble des couples de services Web concrets compatibles.

5.7.2 Recherche du flux de services

La recherche du flux de services est effectuée à l'aide de l'algorithme ***Backtrack (BT)*** présenter ci-dessous**.** Ainsi après avoir choisi un service Web concret pour une tâche donnée, l'algorithme vérifie la consistance de ce dernier choix vis-à-vis des autres services sélectionnés pour les autres tâches. Techniquement, ce service doit être compatible en terme de politiques avec les autres services déjà sélectionnés et avec lesquels il interagie, si ce n'est pas le cas, un autre service fonctionnellement équivalent à celui-ci est choisi à sa place jusqu'à trouver un qui soit consistant ou de les avoir tous testé. Dans ce dernier cas, l'algorithme opère un retours arrière vers l'avant dernière variable (tâche) instanciée pour lui affecter une nouvelle valeur.

Algorithme 5.3 : Algorithme (BT) pour l'approche : CSP4WSC.

Algorithme Backtrack (FS, (S, D, C, R)) : booléen *// FS est une affectation, au premier appelle FS = ∅*

Si non consistant(FS) **Alors** R**etourner** faux **Fsi**

Si FS est une affectation totale **alors retourner** vrai */* FS est une solution */*

Sinon choisir une nouvelle variable Si de S

Pour toute valeur ws_{ik} de D(Si) **faire**

Si Backtrack (FS U{(Si, ws_{ik})},(S-{Si}),{D(S1),…,D(Si)-{ws_{ik}},…,D(Sn)},C,R) = vrai **alors retourner** vrai

Fsi

Fin pour

Fsi

Retourner faux

Fin

Algorithme 5.4 : Algorithme Consistant

Algorithme Consistant(FS) : ***booléen***

pour chaque **(Si, ws_{ik}), (S_j , ws_{jl})** ***∈ FS tel que (Si , Sj) ∈ C***

si **(** **non (**ws_{ik} **Comp** ws_{jl}**)** **)** ***alors*** **retourner** Faux

Fsi

Fin pour

Retourner vrai ;

Fin.

5.8 Exemple d'illustration : (compatibilité complète)

Soit une requête d'un utilisateur qui ne peut pas être satisfaite par un seul service web élémentaire, alors ça nécessite une composition de services Web. Après avoir décomposer cette tâche en sous tâche, nous avons obtenu quatre sous tâches t_1, t_2, t_3, t_4 (quatre services Web composants), S_i est un service Web abstrait qui représente la tache t_i qui contient des services Web concrets fonctionnellement équivalent qui peuvent réaliser la tâche t_i, tel que : **$S=\{S_1,S_2,S_3,S_4\}$**, **$S_1=\{s_{11},s_{12},s_{13},s_{14}\}$**, **$S_2=\{s_{21},s_{22},s_{23},s_{24}\}$**, **$S_3=\{s_{31},s_{32},s_{33},s_{34}\}$**, **$S_4=\{s_{41},s_{42},s_{43},s_{44}\}$**.

Soit le schéma d'interaction entre les services abstraits qui constituent le service Web composite.

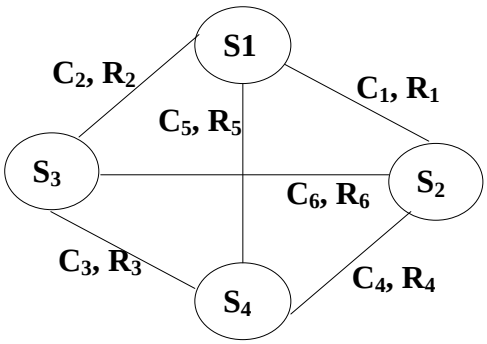

Figure. 5.3 − exemple d'interaction (compatibilité complète)

But et signification du schéma de la figure 5.3 : Chaque lien entre un service Web abstrait et un autre signifier que les deux services Web concrets qui seront choisis sont contraints a être

compatible, c'est-à-dire sont en interaction direct dans la réalisation du but attendu par le client.

Dans les 04 ensembles il n'y a que 04 services Web composants concrets qui sont politiquement compatibles d'une manière binaire et qui peuvent se composer pour donner un service Web composite qui peut répondre aux politiques de l'utilisateur. Soit cet ensemble de services composants compatibles est **{s_{12}, s_{22}, s_{34}, s_{44}}.** en appliquant l'algorithme de ***Backtrack (BT)*** de l'approche CSP, nous aurons le déroulement illustré dans la figure 3 suivante.

Soit un exemple de relation d'une compatibilité complète :

R_1 = {**(s_{12},s_{22}), (s_{13},s_{22}), (s_{13},s_{21})},** R_2 = { **(s_{11},s_{32}) , (s_{12},s_{33}), (s_{14},s_{32}), (s_{14},s_{34})},** R_3 = {**(s_{31},s_{43}), (s_{33},s_{42}), (s_{33},s_{44}),},** R_4 = {**(s_{21},s_{42}), (s_{22},s_{41}), (s_{22},s_{44}), (s_{24},s_{43})},** R_5 = {**(s_{11},s_{42}), (s_{12},s_{42}) , ,(s_{12},s_{44}), (s_{13},s_{42}), (s_{14},s_{44})},** R_6 = {**(s_{21},s_{32}), (s_{22},s_{31}), (s_{22},s_{33})}.**

Le cadre qui est sur **s_{31}** indique que s_{31} est compatible avec s_{22}, mais ne vérifie pas la consistance, c'est-à-dire s_{31} n'est pas compatible avec s_{12}, est ce qui annule la combinaison.

Le cadre qui est sur s_{42} indique que s_{42} est compatible avec s_{33}, mais ne vérifie pas la consistance, c'est-à-dire **s_{42}** n'est pas compatible avec s_{22}, est ce qui annule la combinaison.

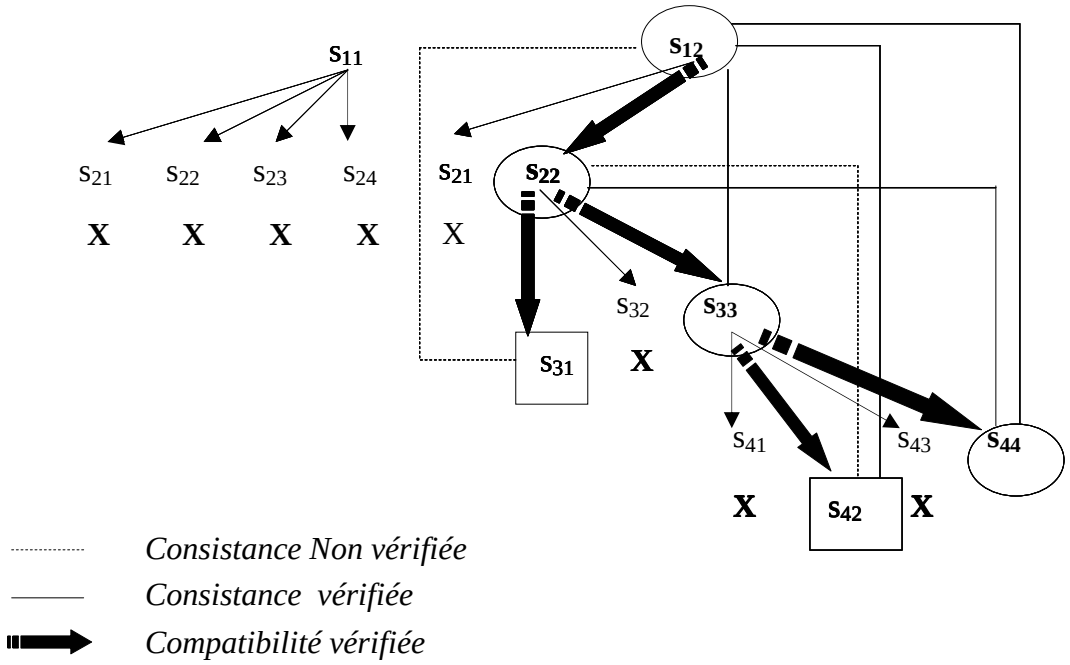

Figure 5.4 − Déroulement de l'algorithme (BT) pour l'approche : *CSP4WSC.*

5.9 Conclusion

Le problème de la compatibilité de politiques ne peut pas être ignoré dans le processus de composition de services Web, par ce la compatibilité joue un rôle majeur dans le choix d'un service Web parmi un ensemble de services fonctionnement équivalents qui sera compatible avec d'autre service choisis avec la même manière, tout ces services choisis seront en suit composés pour rependre aux contraintes, exigences et préférences de l'utilisateur.

Dans ce chapitre, nous avons proposé une nouvelle méthode pour la prise en compte de la compatibilité de politiques, dans le cadre de la composition horizontale de services Web. Ainsi, notre approche se situe sur deux niveaux : (i) la schématisation de tout le processus de composition de services Web basée sur les politiques (services Web composants, service Web composite, utilisateur), avec la proposition d'un algorithme général de la spécification des politiques jusqu'à la composition du flux sélectionné. (ii) aborde la compatibilité de politiques dans la composition horizontale des services Web. Nous nous sommes seulement concentrés sur la recherche d'un flux de service Web compatible, en utilisant l'approche CSP (*Constrait Satisfaction Problem*) pour formaliser et résoudre ce problème.

L’approche CSP s’est révélée très performante pour la résolution de nombreux problèmes ; c’est la raison pour laquelle nous avons tenté dans ce travail, de résoudre le problème de la recherche de flux de services en prenant en considération la compatibilité de politiques.

Conclusion & perspectives

La composition de services Web est un problème difficile, en raison de la formidable croissance du nombre de services Web disponibles, l'environnement dynamique et l'évolution des besoins des utilisateurs. L'existence d'un grand nombre de services qui fournissent les mêmes fonctionnalités (des services fonctionnellement équivalents), élargie cette difficulté pour la recherche d'un service approprié qui peut réaliser une tâche bien spécifiée. Une telle composition ne devrait pas seulement choisir le service disponible le plus approprié, mais devrait également respecter et adhérer aux politiques des services composants, des clients et du service Web composite. Les politiques sont en général regardées comme un ensemble de règles, contraintes, ou ordres qui contrôlent le comportement d'une entité.

Puisque des services Web indépendants se composent afin de satisfaire les besoins des utilisateurs, leurs politiques respectives pourraient être non compatibles. Ceci peut mener aux conflits et aux exceptions dans l'exécution de la composition.

Dans ce mémoire de magister, nous avons discuté comment aborder la compatibilité de politiques dans la composition horizontale des services Web. Nous avons fourni un algorithme pour choisir un service à partir des services fonctionnellement équivalents en vérifiant la compatibilité de politiques. Notre approche consiste, après avoir décomposé la tâche globale attendue par l'utilisateur en sous tâches et recherché pour chaque sous tâche les services Web capable de la réaliser, de prendre en compte la compatibilité de politiques lors de la sélection des services.

Dans notre approche nous nous sommes concentrés sur le processus de sélection d'un flux de services Web compatibles en terme de politiques en exploitant une approche CSP (*Constrait Satisfaction Problem*) pour formaliser et résoudre ce problème. Notre formalisation CSP se définit comme un ensemble de contraintes et de relations impliquant un ensemble de variables. L'objectif est de trouver une valeur pour chaque variable, afin de satisfaire l'ensemble des contraintes. Notre travail introduit une nouvelle formalisation du problème de la sélection de l'ensemble des services Web à composés, en exploitant les avantages du formalisme CSP.

En perspectives nous envisageons de :

1) Développer un formalisme (des règles) qui permet la vérification de la compatibilité de politique (Business, comportement, confidentialité, contrôle d'accès…) entre des services Web qui vont se composer. Ce formalisme nous permettra de calculer les couples de relation entre chaque deux services Web abstraits.

2) Intégrer ce formalisme de vérification dans notre approche présentée ci dessus.

3) Utiliser les *CSOP* (*Constraits Satisfaction and Optimisation Problems)* pour trouver une solution qui satisfait toutes les contraintes en optimisant une fonction objective.

4) Utiliser les *MaxCSP,* en cas ou il n'existe pas de solution, alors nous cherchons une solution qui viole le minimum de contraintes considérées.

Références

[1] H. Mahesh Dodani. From objects to services. A journey in search of component reuse nirvana. Journal of Object Technology, 3(8):49–54, 2004.

[2] M. MacKenzie, K. Laskey, F. McCabe, P. Brown, and R. Metz. Reference Model for Service Oriented Architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS, February, 2006.

[3] S. Hashimi. Service-oriented architecture explained. http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html. O'reilly on dot net, August 2003.

[4] W. Roy Schulte and V. Yefim Natis. Service oriented architectures, part 1 & 2. http ://www.gartner.com. Gartner Group, April 1996

[5] G. Wang and C. K. Fung. Architecture paradigms and their influences and impacts on component-based software systems. In 37th Hawaii International Conference on System Sciences (HICSS), 2004.

[6] N. Dokovski, I. Widya, and A. van Halteren. Paradigm. Service oriented computing. (Freeband/AWARENESS/D2.7b) TI/RS/2004/145, 2004.

[7] M.P. Papazoglou and W.J. van den Heuvel. Service-oriented design and development methodology. Int. J. of Web Engineering and Technology (IJWET), 2006.

[8] V. Muthusamy and H. Jacobsen. Small scale peer-to-peer publish/subscribe. In International Workshop on Description Logics (DL2005), Edinburgh, Scotland, 2005.

[9] M. Colan. Service-oriented architecture expands the vision of web services.IBM, developerWorks, 2004.

[10] D. Georgakopoulos and M. Papazoglu. Service-oriented computing. Communications of the ACM, Vol. 46, No 10, 2003.

[11] N. Balani. Model and build esb soa frameworks. IBM, developerWorks,

http://www-128.ibm.com/developerworks/web/library/wa-soaesb/, 2005.

[12] IBM and All. Service component architecture, building systems using a service oriented architecture, version 0.9. Joint Whitepaper by BEA, IBM, Interface21, IONA, Oracle, SAP, Siebel, Sybase. 2005.

[13] CBDI. Web services protocols summary. http://roadmap.cbdiforum.com/reports/protocols/summary.php, July 2005.

[14] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business process execution language for web services version 1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, July 2005.

[15] M. Judith Myerson. Web services architectures. January 2002. http://www.webservicesarchitect.com/content/articles/webservicesarchitectures.pdf,

[16] W3C Working Group Note. Web services glossary. http://www.w3.org/TR/2004/NOTE-ws-gloss20040211, February 2004.

[17] M. P. Papazoglou. Web services. Concepts, architectures, and applications. In Springer Verlag, 2004.

[18] A. Arkin, S. Askary, S. Fordin,W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek. Web service choreography interface. Available from http://www.w3.org/TR/wsci, August 2002

[19] M. Kahn, C. Cicalese. CoABS Grid Scalability Experiments. O.F. Rana (Ed.), 2nd International Workshop on Infrastructure for Scalable Multi-Agent Systems at the 5th International Conference on Autonomous Agents. Montreal, CA, New York: ACM Press, 2001.

[20] J. Bryson, D. Martin, S. McIlraith, and L. Stein. Agent-based composite services in daml-s : The behavior-oriented design of an intelligent semantic web. Springer Verlag, 2002.

[21] S. McIlraith, T. Son, and H. Zeng. Semantic web services. IEEE Intelligent Systems. Special issue on the Semantic Web, 46–53,16(2), 2001.

[22] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, 284(5) :34–43, 2001.

[23] B. Medjahed. Semantic Web enabled composition of Web services. Falls Church, Virginia, USA. January 19th, 2004.

[24] A. Arkin. Business Process Modeling Language, November 2002.

[25] M. Carman, L. Serafini, and P. Traverso. Web service composition as planning. In Proceedings of ICAPS'03 Workshop on Planning for Web Services (Trento, Italy), June 2003.

[26] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in EFlow. In Proceeding of the 12th Conference on Advanced Information Systems Engineering (CAISE), S. Verlag, Ed., pp. 13–31, (Stockholm, Sweden), June 2000.

[27] F. Casati, M. Sayal, and M.-C. Shan. Developping e-services for composing e-services. In Proceedings of 13th International Conference on Advanced Information Systems Engineering (CAiSE), S. Verlag, Ed, (Interlaken, Switzerland, June 2001).

[28] Z. Cheng, M. Singh, and M. Vouk. Composition constraints for semantic web services. In Proceedings of the WWW-2002 workshop, 2002.

[29] I. Constantinescu, B. Faltings, and W. Binder, Type based service composition. In Proceedings of the 13th international World Wide Web Conference on Alternate Track, ACM Press, pp. 268–269, (New York, USA), 2004.

[30] J. Establisher, and S. Sanlaville, Extensible process support environments for web services orchestration. International Journal of Web Services Practices 1, 30–39, 1-2 , 2005.

[31] S. Jamal, Environnement de procédé extensible pour l'orchestration – Application au service web. PhD thesis, Universit´e Grenoble UJF, 2005.

[32] R. Khalaf, N. Mukhi, and S. Weerawarana, Service-oriented composition in BPEL4WS. In Proceedings of the 12th International World Wide Web Conference, 2003.

[33] M. Klush, A. Gerber, and M. Schmidt, Semantic web service composition planning with OWLS-Xplan. In Proceedings of the 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, AAAI Press, (Arlington, USA), 2005.

[34] M. Laukkanen, and H. Helin, Composing workflows of semantic web services. In Proceedings of the Workshop on Web-Services and Agent-based Engineering, 2003.

[35] F. Leymann. Web Services Flow Language - WSFL 1.0. Tech. rep., IBM Software Group, May 2001.

[36] D. McDermott, Estimated-regression planning for interactions with web services. In Proceedings of the 6th International Conference on AI Planning and Scheduling, AAAI Press, (Toulouse, France), 2002.

[37] S. McIlraith, and T. Son, Adapting Golog for composition of semantic web services. In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR ’02), Morgan Kaufmann Publishers, pp. 482–493, (Toulouse, France), April 2002.

[38] B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Semantic web enabled composition of web services. The VLDB Journal 12, 4, 333–351, November 2003.

[39] I. Muller, and R. Kowalczik, Service composition through agent-based coalition formation. In Proceedings of WWW Service composition with Semantic Web Services, pp. 34–43, (Compiegne, France), Sept 2005.

[40] S. Narayanan, and S. McIlraith, Simulation, verification and automated composition of web services. In Proceedings of the 11th international conference on World Wide Web, ACM, (Honolulu, USA), July 2002.

[41] T. Osman, D. Thakker, and D. Al-Dabass. Bridging the gap between workflow and semantic-based web services composition. In Proceedings of WWW Service composition with Semantic Web Services, pp. 13–23, (Compiègne, France), Sept. 2005.

[42] J. Peer. APDDL based tool for automatic web services composition. In Proceedings of the Second Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2004) at the 20th International Conference on Logic Programming, Springer Verlag, pp. 149–163, September 2004.

[43] J. Peer. Web service composition as planning - a survey. Tech. rep., Univ. Of St.Gallen, 2005.

[44] C. Peltz. Web services orchestration - a review of emerging technologies, tools, and standards. Tech. rep., HP, January 2003. HP. [44] Rao, J., Kungas, P., and Matskin, M. Application of linear logic to web service composition. In Proceeding of the 1st International Conference on Web Services (Las Vegas, USA), June 2003.

[45] J. Rao, P. Kungas, and M. Matskin. Application of linear logic to web service composition. In Proceeding of the 1st International Conference on Web Services (Las Vegas, USA), June 2003.

[46] J. Rao, and X. Su. A survey of automated web service composition methods. In Proceedings of Semantic Web Services and Web Process Composition, First International Workshop (San Diego, CA), July 2004.

[47] M. Sheshagiri, M. Desjardins, and T. Finin. A planner for composing services described in DAML-S. In Proceedings of the AAMAS Workshop on Web Services and Agent-based Engineering, June 2003.

[48] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. In Proceedings Web Services : Modeling, Architecture and Infrastructure. Workshop in Conjunction with ICEIS2003, ICEIS Press, pp. 17–24, (Angers, France), 2002.

[49] E. Sirin, and B. Parsia. Planning for semantic web services. In Proceedings of Semantic Web Services Workshop at 3rd International Semantic Web Conference, 2004.

[50] S. Thatte, XLANG - web services for business process design, 2001.

[51] Waltinger, R. Web agents cooperating deductively. In Proceeding of FAABS 2000, Greenbelt, Ed., vol. 1871 of Lecture Notes in Computer Sciences, Springer-Verlag, pp. 250–262, April 2000.

[52] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In ISWC'03, 2003.

[53] O. Zimmermann, M. Tomlinson, and S. Peuser. Perspectives on Web Services – Applying SOAP, WSDL and UDDI to Real-Worlds Projects. Springer-Verlag, 2003.

[54] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. Communication of the ACM, 46(10) :29–34, 2003.

[55] C. Peltz. Web services orchestration and choreography. Computer Magazine, 36(10) :46–52, 2003.

[56] V. Gabor, B. Andersson, and P. Wohed. An ontology based analysis of bpel4ws and wsci. In Proceedings of the 3rd Nordic Conference on Web Services (NCWS), 2004.

[57] WSMO. Web service modeling ontology. http://www.wsmo.org/, 2006.

[58] B. Benatallah and Q. Z. Sheng. The Self-Serv Environment for Web Services Composition. IEEE Computer Society, 2003.

[59] M. C. Daconta, L.J. Obrst, and K.T. Smith. The Semantic Web. Wiley Publishing Inc., Indianopolis, Indiana, 2003.

[60] W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. IEEE Intelligent Systems, 18(1):72–76, 2003

[61] D.J. Mandell and S.A. McIlraith. Adapting bpel4ws for the semantic web bottomup approach to web service interoperation. In Proceedings of the Second International

Semantic Web Conference, volume 2870 of Lecture Notes in Computer Science, pages 227–247, Sanibel Island, Florida, 2003.

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

[63] J. Davies, D. Fensel, and F. van Harmelen. Towards the Semantic Web. John Wiley & Sons, 2003.

[64] A. Anderson. An Introduction to the Web Services Policy Language (WSPL), Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, Yorktown Heights, New York, pp. 189-192, 7-9 June 2004.

[65] W3C, XML Schema Part 2: Datatypes, W3C Recommendation, http://www.w3.org/TR/ x mlschema - 2/, 2 May 2001.

[66] W3C, XQuery 1.0 and XPath 2.0 Functions and Operators,W3C Working Draft 2002, http://www.w3.org/TR/2002/WD- x query- operators-20020816, 16 August 2002.

[67] Z. Maamar, D. Benslimane, and A. Anderson, “Using Policies to Manage Composite Web Services,” IEEE IT Professional, vol. 8, no. 5, pp. 47–51, September/October 2006.

[68] A. Uszok and colleagues. KAoS Policy and Domain Services : Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. Proc. IEEE 4th Workshop Policies for Distributed Systems and Networks, IEEE CS Press, pp. 93-98, 2003.

[69] L. Kagal, T. Finin, and A. Joshi. A Policy Language for Pervasive Computing Environment. Proc. IEEE 4th Workshop Policies for Distributed Systems and Networks, IEEE CS Press, pp. 63-76, June 2003.

[70] N. Damianou, N. Dulay, E. Lupu, M. Sloman. The Ponder Policy Specification Language. In proceedings of Workshop on Policies for Distributed Systems and Networks (POLICY 2001). Springer-Verlag, LNCS 1995, Bristol, UK, 2001.

[71] J. M. Bradshaw, P. Beautement, L. Bunch, S. V. Drakunov et al. Making agents Acceptable to People. In N. Zhong, J. Liu (eds): Handbook of Intelligent Information Technology (in press). IOS Press, Amsterdam, the Netherlands, 2003.

[72] M. Johnson, P. Chang, R. Jeffers, J. Bradshaw, et al. KAoS Semantic Policy and Domain Services: An Application of DAML to Web Services-Based Grid

Architectures. Submitted to the AAMAS 03 workshop on Web Services and Agent-Based Engineering, Melbourne, Australia, July, 2003.

[73] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri et al. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. To appear in proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Como, Italy, 2003.

[74] J. M. Bradshaw, A. Uszok, R. Jeffers, N. Suri, et al. Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads. To appear in the Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003). Melbourne, Australia, New York, NY: ACM Press, 2003.

[75] L. Kagal, T. Finin, A. Johshi. A Policy Language for Pervasive Computing Environment. To appear in proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Como, Italy, 2003.

[76] L. Kagal. Rei. A Policy Language for the Me-Centric Project. HP Labs Technical Report,HPL-2002-270, 2002.

[77] E. C. Lupu, M. Sloman. Conflicts in policy-based distributed system management. IEEE Transaction on Software Engineering, Vol. 25, No. 6, 1999.

[78] M. Sloman. Policy Driven Management for distributed Systems. Plenum Press Journal of Network and Systems Management, Vol. 2, No. 4, 333-360, 1994.

[79] N. Damianou, N. Dulay, E. C. Lupu, M. Sloman. Ponder: A Language for Specifying Security and Management Policies for Distributed Systems. Imperial College, UK, Research Report Department of Computing 2001, 2000.

[80] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, A. Uszok. Semantic Web languages for policy representation and reasoning. a comparison of KAoS, Rei, and Ponder, in: Proceedings of the 2nd International Semantic Web Conference (ISWC'2003), Sanibel Island, Florida, USA, 2003.

[81] J. Schlimmer, ed. Web Services Policy Framework (WSPolicy), http://m sdn.microsoft.com/ library/d efault.asp? url = /library/en-us/dnglobspec/h tml/ ws-policy.asp, September 2004.

[82] W3C, Web Services Description Language (WSDL) 1.1, W3C Note, http://www.w3.org/TR/ wsdl, 15 March 2001.

[83] U. Yalcinalp. Proposal for adding Compositors to WSDL 2.0,

http://lists.w3.org/Archives/Public/www-wsdesc/2004Jan/0153.html, 26 January 2004.

[84] T. Moses, ed. XACML profile for Web-services, http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf, Working draft 04, (also known as “Web Services Policy Language (WSPL)”), 29 Sept 2003.

[85] T. Moses, eds. OASIS eXtensible Access Control Markup Language (XACML), OASIS Standard 2.0,http://w ww.oasis-open.org/ committees/xacml, 1 February 2005.

[86] W3C, XML Path Language (XPath), Version 1.0, W3C Recommendation, h ttp://www.w3.org/TR/x path, 16 November 1999.

[87] C. Sharp, ed. Web Services Policy Attachment (WSPolicyAttachment), http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp, September 2004.

[88] A. Nadalin, ed. Web Services Security Policy Language(WS-SecurityPolicy), Version 1.0,http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws- s ecuritypolicy.asp, 18December 2002.

[89] A. Nadalin, et al, eds., WS-Security, OASIS Standard 1.0, http://www.oasis open.org/committees/tc_home.php? wg_abbrev= wss, 6 April 2004.

[90] J. Schlimmer, personal communication, 12 October 2004.

[91] W3C, W3C Workshop on Constraints and Capabilities for Web Services, http:// w ww.w3.org/ 2004/09/ws-cc-program., 12-13 October 2004.

[92] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In Proc. of Int. Conf. on Implementation and Application of Automata (CIAA), 2003.

[93] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of eservice composition. In Proc. of Int. World Wide Web Conf. (WWW), May 2003.

[94] A. Ben Hassine, S. Matsubara, and T. Ishida. A Constraint-based Approach to Horizontal Web Service Composition. In the proceedings of the 5th International Semantic Web Conference (ISWC), pp.130-143, Athens-USA, 2006.

[95] J. M. Bradshaw, P. Beautement, L. Bunch, S. V. Drakunov et al. Making agents Acceptable to People. In N. Zhong, J. Liu (eds): Handbook of Intelligent Information Technology (in press). IOS Press, Amsterdam, the Netherlands, 2003.

[96] M. Johnson, P. Chang, R. Jeffers, J. Bradshaw, et al. KAoS Semantic Policy and Domain Services: An Application of DAML to Web Services-Based Grid

Architectures. Submitted to the AAMAS 03 workshop on Web Services and Agent-Based Engineering, Melbourne, Australia, July 2003.

[97] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri et al. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. To appear in proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Como, Italy, 2003.

[98] E.C. Lupu and M. Sloman. ("Conflicts in Policy-Based Distributed Systems Management,", IEEE Trans. Software Eng., vol. 25, no. 6, pp. 852- 869, , Nov.-Dec. 1999.

[99] Z. Maamar, D. Benslimane, P. Thiran, C. Ghedira, S. Dustdar et S. Sattanathan. Towards a context-based multi-type policy approach for Web services composition. Knowledge and Data Engineering Journal 62(2):327-351, 6/2007.

[100] S. Ae Chun, V. Alturi, and N. Adam. Using Semantics for Policy-based Web Service Composition. Distributed and Parallel Databases, Kluwer Academic Publishers, 18(1), July 2005.

[101] H. Anne Anderson. "An Introduction to the Web Services Policy Language (WSPL)**",**in 5th IEEE International Workshop on Policies for Distributed Systems and Networks, orktown Heights,New York, 7-9 June 2004;

[102] S. Ae Chun, V. Atluri and N. R. Adam. Policy-based Web Service Composition. In Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)

[103] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara. Authorization and Privacy for Semantic Web Services. IEEE Intelligent Systems, 19(4), July/August 2004.

[104] J. Fritz Barnes, R. Pandey: CacheL: Language Support for Customizable Caching Policies.In Proceedings of 4th Interantional Web Caching Workshop, San Diego, CA, 1999.

[105] J. Hoagland. Specifying and Implementing Security Policies Using LaSCO, the Language for Security Constraints on Objects. Ph.D. dissertation, UC Davis, 2000.

[106] N. Suri, J. M. Bradshaw, M. R. Breedy, P. T. Groth, et al: NOMADS: Toward an environment for strong and safe agent mobility. Proceedings of Autonomous Agents 2000. Barcelona, Spain, New York: ACM Press, 2000.

[107] The IETF Policy Framework Working Group: Charter available at http://www.ietf.org/html.charters/policy-charter.html.

[108] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, A. Uszok. Semantic Web languages for policy representation and reasoning: a comparison of KAoS, Rei, and Ponder, in: Proceedings of the 2nd International Semantic Web Conference (ISWC'2003), Sanibel Island, Florida, USA, 2003.

[109] Z. Maamar, D. Benslimane, G. Kouadri Mostéfaoui, S. Sattanathan, and C. Ghedira. Developing Interoperable Business Processes Using Web Services and Policies. In Proceedings of The Second International Conference onInteroperability for Enterprise Software and Applications (I-ESA'2006), Bordeaux, France, 2006.

[110] Z. Maamar, Q. Z. Sheng, H. Yahyaoui, D. Benslimane, and F. Liu, "On Checking the Compatibility ofWeb Services' Policies". Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007), Adelaide, Australia. IEEE Computer Society 2007, ISBN 0-7695-3049-4, 3-6 December 2007.

[111] Z. Maamar, Djamal Benslimane, G. K. Mostefaoui S. Subramanian, and Q. H. Mahmoud. Senior Member, IEEE "Towards Behavioral Web Services Using Policies", IEEE transactions on systems, man, and cybernetics–part a: systems and humans, vol. x, no. x, month 2008.

[112] Z. Maamar, D. Benslimane, and A. Anderson. "Using Policies to Manage Composite Web Services," IEEE IT Professional, vol. 8, no. 5, pp. 47–51, September/October 2006.

[113] Y. Li, H.-Y. Paik, B. Benatallah, and S. Benbernou. Formal Consistency Verification between BPEL Process and Privacy Policy. In Proceedings of The International Conference on Privacy, Security, and Trust (PST'2006), Toronto, Canada, 2006.

[114] L. Baresi, S. Guinea, P. Plebani. WS-policy for service monitoring, in: Proceedings of the 6th VLDB Workshop on Technologies for E-Services (TES'2005) held in conjunction with the 31st International Conference on Very Large Data Bases (VLDB'2005), Trondheim, Norway, 2005.

[115] Z. Maamar, S. Kouadri Moste´faoui, Q.H. Mahmoud. On personalizing Web services using context, International Journal of Ebusiness Research, Special Issue on E-Services, The Idea Group Inc. 1 (3), 2005.

[116] Y. Shi, L. Zhang, F. Liu, L. Lin, B. Shi. Department of Computing and Information Technology, Fudan University, China "Compatibility Analysis of Web Services" in

Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05).

[117] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Compatibility verification for web service choreography. Proc. ICWS, pp. 738-741. IEEE, 2004.

[118] L. Bordeaux, G. Salaun, D. Berardi, M. Mecella. When Are Two Web Services Compatible? In Proc. of the 5th VLDB International Workshop on Technologies for e-Services (VLDB-TES), 2004.

[119] U. Montanari. Networks of constraints. Fundamental properties and applications to picture, processing. Information Sciences, 1974.