REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira de Béjaïa
Faculté des Sciences et Sciences de l'Ingénieur

Département d'Informatique
Ecole Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère en Informatique

Option

Réseaux et Systèmes Distribués

Thème

MODÉLISATION ET COMPOSITION DE SERVICES SENSIBLES AU CONTEXTE

Présenté par

Ali YACHIR

Directeur du mémoire : **M. Yacine AMIRAT, Professeur, Université Paris 12, France.**

Devant le jury composé de :

Président : M. Djamil AISSANI, Professeur, Université de Béjaïa, Algérie.

Rapporteur : M. Yacine AMIRAT, Professeur, Université Paris 12, Val-de-Marne, France.

Examinateurs : M. Abdallah BOUKERRAM, Maître de Conférences, Université de Sétif, Algérie.

Mme. Saadia TAS, Maître de Conférences, Université de Béjaïa, Algérie.

Promotion 2006 - 2007

A mes parents,

A mes frères et sœurs,

A tout mes proches,

A tout mes amis,

A la mémoire de : R.Fatima, Y.Ali, G.Hocine, M.Smail.

Table des Matières

Liste des Figures

Liste des Tableaux

Remerciements

Si ce mémoire a pu voir le jour, c'est certainement grâce au soutien et l'aide de plusieurs personnes qui m'ont permis d'accomplir ce travail dans des bonnes conditions. Je profite de cet espace pour les remercier tous.

Je tiens d'abord à exprimer ma plus grande reconnaissance et ma profonde gratitude à mon directeur de thèse, Monsieur Yacine AMIRAT, Professeur à l'Université Paris 12-Val de Marne, pour m'avoir encadré et accordé son aide précieuse tout au long de cette année, et pour ses orientations éclairées et ses encouragements qui m'ont permis de mener à bien cette étude. J'ai vraiment apprécié ses qualités humaines, son dynamisme et ses précieux conseils.

J'aimerai aussi adresser mes vifs remerciements à Monsieur Karim TARI, doctorant à l'Université Paris 12, pour sa disponibilité et ses bons conseils. J'ai apprécié les discussions enrichissantes avec lui tant du point de vue scientifique que sur le plan humain.

J'adresse mes sincères remerciements à Monsieur Djamil AISSANI, Professeur à l'Université de Béjaia, qui m'a fait l'honneur de présider le jury de thèse.

J'exprime ma profonde reconnaissance à Madame Saadia TAS, Maître de Conférences à l'Université de Béjaia, et à Monsieur Abdallah BOUKERRAM, Maître de Conférences à l'Université de Sétif, pour l'intérêt qu'ils ont bien voulu porter à ce travail en acceptant d'en être examinateurs.

Mes très vifs remerciements vont également à Monsieur Kamel TARI, chef du département informatique de l'Université de Béjaia pour son aide, sa disponibilité et ses contributions pour l'école doctorale ReSyD. Un merci bien distingué à tous les professeurs de l'école doctorale ReSyD pour m'avoir enseigné et donné des cours de très bonne qualité.

Je désire remercier tout particulièrement mes parents, ma grande mère, mes frères, mes soeurs et tous mes proches en particulier N. MOUMOU et toute sa famille. Leur soutien, leur aide et leur patience sont pour beaucoup dans l'accomplissement de cette thèse.

Enfin, un grand merci à tous mes amis qui m'ont soutenu et encouragé en particulier : Nacer, Mustapha, Salem, Slimane, Salah, Djamel, Rafik, Faouzi, Fateh, Foudil, Khaled, Malek, ainsi qu'à tous ceux qui ont contribué de près ou de loin à la réalisation de ce modeste travail.

ReSyD, Université de Béjaia Ali YACHIR
Novembre, 2007

Résumé :

Le succès du Web et la prolifération des réseaux sans fils et des équipements mobiles intelligents, permet aujourd'hui d'envisager des services à valeur ajoutée, en exploitant le concept de l'informatique ubiquitaire ou pervasive. A l'inverse de l'informatique traditionnelle qui est orientée traitement, l'informatique ubiquitaire exige une adaptation du logiciel vis-à-vis de l'environnement de l'utilisateur et des besoins de ce dernier. On parle ainsi d'une informatique orientée services (*SOC : Service Oriented Computing*). La tendance actuelle de l'approche SOC est vers la composition de services. Elle consiste à créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services existants, composés ou non en vue d'apporter une valeur ajoutée. Dans un environnement ubiquitaire, on parle de services sensibles au contexte, car ces services s'adaptent au contexte de l'utilisateur. Leur composition nécessite alors de prendre en compte le contexte et l'incertitude de l'environnement ubiquitaire. Cette thèse de magistère s'inscrit dans cette optique et vise à modéliser et composer des services sensibles au contexte. Il s'agit d'abord d'automatiser le processus de composition. Ensuite, étudier différentes approches afin de sélectionner les services pertinents et prendre en compte les perturbations qui surviennent sur ces services. Ces approches sont : les processus de décisions Markoviens (MDP), l'apprentissage de modèles Bayésiens et modèles à base d'agents. Les testes que nous avons effectués au cours de ce travail montrent clairement la performance des processus de décision markovien et de l'apprentissage Bayésien pour la composition de services sensibles au contexte.

Mots-clés : informatique ubiquitaire, système pervasifs, composition de services, contexte, sensibilité au contexte, MDP, apprentissage Bayésien, systèmes multi agents.

Abstract :

The success of the Web and the proliferation of the wireless networks and the intelligent mobile equipment, make possible today to consider added value services, by exploiting the concept of ubiquitous or pervasive computing. Contrary to the traditional computing which is treatment oriented, ubiquitous computing requires an adaptation of the software to the environment of the user and his needs. Thus, we speak about a service oriented computing (SOC). The current tendency of SOC approach is towards the composition of services. It consists in creating new functionalities by combining functionalities offered by other existing services, composed or not in order to bring an added value. In the ubiquitous environment, we speak about context aware services, because these services adapt theirs contains to the context of the user. So, their composition requires taking into account the context and the uncertainty of the ubiquitous environment. The goal of this thesis is to model and compose context aware services. It is initially a question of automating the process of composition. Then, we study various approaches in order to select the relevant services and to take into account the disturbances which occur on these services. These approaches are: Markovian decision processes (MDP), the Bayesians learning models and agent models. The test that we have done during this work show clearly the performance of the Markovian decision process and the Bayesian learning for the composition of context aware services.

Key words : ubiquitous computing, pervasive system, services composition, context, context aware, MDP, Bayesians learning, multi agent systems.

Introduction Générale

L'informatique ubiquitaire (*ubiquitous computing*) ou informatique diffuse (*pervasive computing*), également appelée informatique ambiante, représente un pas majeur vers une évolution radicale dans la manière d'interagir avec un système d'information et de communication. Le paradigme de l'informatique ubiquitaire a ouvert des nouvelles perspectives de recherche sur les futurs systèmes d'informations. Contrairement à l'approche traditionnelle orientée traitement, l'approche ubiquitaire est orientée service. Elle vise à assister l'utilisateur dans ces activités quotidiennes d'une manière transparente et intuitive. La vision est celle d'un environnement omniprésent « intelligent », qui assiste l'utilisateur dans ses tâches sans être envahissant, et capable d'exécuter les actions appropriées tout en s'adaptant aux conditions environnementales et aux besoins et préférences de l'utilisateur.

L'émergence de l'informatique ubiquitaire est rendue possible grâce aux progrès technologiques incessants vers la miniaturisation des dispositifs électroniques et informatiques. Ces progrès permettent d'ores et déjà d'intégrer une capacités de calcul et de communication dans des objets courants. Ces objets vont des équipements mobiles (PC portables, assistants PDA et PC de poche) aux dispositifs embarqués (appelé aussi enfouis) tels que les badges d'identification RFID, les téléphones portables, les capteurs, les actionneurs et tout autre équipement intelligent de contrôle/commande. L'ensemble de ces objets forme ainsi un système informatique ubiquitaire dont l'interface intuitive et flexible est constituée de l'environnement même de l'utilisateur. En favorisant la communication entre ces objets et en leurs permettant de percevoir l'environnement physique, l'informatique ambiante cherche à transformer les réseaux de capteurs et des équipements intelligents à un environnement simple, facile et proche de l'utilisateur. Cet environnement adapte son comportement d'une façon autonome aux activités de l'utilisateur et son contexte afin de lui offrir des services dits sensibles au contexte *(context-aware).*

Cependant, construire un tel environnement est un sujet défi. En effet, les environnements quotidiens de l'utilisateur sont composés des équipements hétérogènes, formant un système ouvert et dynamique, dans lequel les ressources disponibles, le contexte de l'utilisateur et ses activités, changent continuellement. Cela nécessite de construire sur une architecture type qui traite l'hétérogénéité des composants ubiquitaires et qui soit sensible au contexte d'utilisation.

L'architecture orientée services (SOA : *Service Oriented Architecture*) est une architecture type qui offre des solutions au problème de l'interopérabilité entre les applications distribuées. Dans l'approche SOA, les ressources logicielles disponibles sur le réseau sont des services. Ces derniers sont décris d'une manière indépendante de leurs implémentations ; il sont faiblement couplés, et communiquent avec des protocoles standards. Cette architecture type est la plus commode puisqu'elle permet une utilisation homogène des composants logiciels hétérogènes qui résident dans des environnements ambiants. Les services Web sont l'une des réalisations de l'architecture SOA. Ils représentent des services faiblement couplés qui communiquent en utilisant les technologies standards qui ont fait le succès du Web. L'introduction du paradigme agent dans une architecture orientée services a permis de rendre les services Web plus autonomes, plus intelligents, et capables même de percevoir leurs environnements (utilisateurs, autres agents, système). On parle alors de services d'agents. Ainsi, en représentant les composants de l'environnement ubiquitaire

comme un ensemble de services ayant la capacité de coopérer entre eux, le problème d'intégration dynamique des composants peut être traité comme un problème de composition dynamique de services.

La composition dynamique de services consiste à combiner les composants existants afin de créer des nouvelles fonctionnalités qui ne sont pas directement disponibles dans les services de bases. Il s'agit en faite, d'agréger un ensemble de fonctionnalités élémentaires dans des services de haut niveau, proches des spécifications de l'utilisateur. La composition de services est ainsi un moyen de jonction entre les fonctionnalités élémentaires et les besoins des utilisateurs. Cependant, devant la prolifération du nombre de services disponibles, le problème de combiner plusieurs services appropriés pour atteindre un certain objectif devient une tâche très délicate. Á ceci, s'ajoutent les informations du contexte qui sont un élément important à prendre en compte lors du processus de sélection et de combinaison de services, cela peut compliquer d'avantage le problème. Il est au-delà de la compétence humaine d'analyser et de générer le plan de composition manuellement. Plusieurs approches ont été proposées pour résoudre ce problème. La plupart d'entres elles sont inspirées des recherches menées conjointement sur le Workflow et les techniques de planification issues de l'intelligence artificielle (IA). Toutefois, les méthodes classiques de planification de l' IA supposent que l'environnement est statique et l'invocation des services est déterministe. De plus, trouver simplement un plan qui atteint le but est insuffisant. La composition de services nécessite de prendre en compte non seulement le coût et le temps nécessaires pour accomplir les tâches, mais aussi d'autres facteurs liés au contexte d'utilisation. Peu de travaux qui ont intégré la notion du contexte dans le processus de composition de services. Cela est dû au fait qu'il n'y a pas une formalisation standard du contexte, et la pertinence des informations du contexte diffère d'un domaine à un autre.

C'est dans cette optique que s'inscrivent les travaux de recherche présentés dans ce mémoire de magistère. Il s'agit de modéliser et composer des services sensibles au contexte. Nous proposons alors un modèle de composition de services sensibles au contexte qui comprend une planification globale afin d'atteindre le but de la composition, et un mécanisme de raisonnement basé sur les informations du contexte. L'incertitude sur le contexte et les services est prise en compte par l'introduction des processus de décision markoviens (MDP) et l'apprentissage Bayesien. Vers la fin, nous introduisons le paradigme des agents afin de prendre en compte la distribution des tâches et la coopération dans le processus de composition de services.

Afin de mener à bien notre travail, nous avons organisé ce mémoire de magistère en deux parties de la manière suivante :

La première partie : porte sur l'état de l'art du thème étudié. Elle se scinde en trois chapitres :

Le premier chapitre présente le contexte général de notre travail. Il s'agit d'abord d'introduire le paradigme de l'informatique ubiquitaire et les principales caractéristiques de son environnement. Ensuite, nous présentons quelques architectures qui servent de base pour construire des applications et des intergiciels dans un environnement ubiquitaire, avant de se focaliser sur les architectures orientées services (SOA) qui constituent la tendance actuelle des développeurs. Nous détaillons la notion de service et plus particulièrement les services Web. Pour ces derniers, nous expliquons leurs principes de fonctionnement et leurs sémantiques qui est largement adoptée afin de permettre l'automatisation du processus de découverte et de

composition de services. À la fin, nous montrons la convergence des services Web et des technologies ubiquitaires, avant de terminer ce chapitre par une synthèse.

Dans le deuxième chapitre, nous rappelons d'abord quelques définitions et classifications utilisées pour la notion du contexte, ainsi les différentes approches d'acquisition et de modélisation du contexte. Nous abordons, par la suite, la notion de sensibilité au contexte et les mécanismes d'adaptation du contexte aux applications. Enfin, nous présentons quelques architectures sensibles au contexte et nous terminons par une synthèse de ce chapitre.

Le troisième chapitre est consacré à la composition de services. Nous donnons d'abord quelques définitions et concepts liés à la composition de services et quelques exemples de son utilisation. Nous allons également mettre en valeur l'intérêt de la composition de services dans des domaines applicatifs très variés. Nous donnons par la suite un aperçu sur les récentes recherches sur la composition automatique de service Web avec le Workflow et la planification à base de l' IA. Nous présentons à la fin certains travaux qui se sont intéressés à la prise en compte des informations du contexte lors de la composition de services, avant de terminer par une synthèse qui situera la problématique à traiter dans cette thèse.

La deuxième partie : porte sur la proposition d'un modèle de composition de services sensibles au contexte. Elle se scinde en quatre chapitres :

Le quatrième chapitre présente d'abord la modélisation du contexte et des services sensibles au contexte. Il détaille ensuite l'approche proposée pour la vérification et la construction automatique d'un plan de composition de services en développant un algorithme appelé *FCoSC (Feasibility & Construction of Service Composition)*.

Le cinquième chapitre illustre en détail une approche de composition de services qui traite le non déterminisme des services, l'incertitude sur le contexte, et la nature dynamique de l'environnement. Cette approche se base sur les processus de décision markoviens (MDP) et l'apprentissage Bayésien. En effet, le formalisme MDP permet de prendre en compte le non déterminisme des actions (services) par les probabilités de transition. De plus, la fonction de récompense du MDP permet de guider la sélection vers le service qui satisfait le maximum de contexte. L'approche d'apprentissage Bayésien adapte la solution trouvée aux changements survenus dans l'environnement.

Le sixième chapitre exploite le paradigme d'agents pour la composition de services. Il s'agit de faire coopérer un ensemble d'agents afin d'atteindre l'objectif assigné à la composition de service. Ce chapitre définit les agents qui participent à la composition de service et l'organisation de leur communication, et présente les algorithmes de fonctionnement de chaque agent.

Le septième et dernier chapitre présente une étude de cas qui porte sur les services pour l'assistance et la surveillance d'une personne dépendante. Ce chapitre présente également quelques testes sur notre algorithme d'apprentissage pour la composition de services sensibles au contexte.

Ce travail de mémoire de magistère s'achève par une conclusion générale et quelques perspectives de recherche.

Partie 1

Etat de l’art

Chapitre I

Informatique ubiquitaire et services Web

I.1 Introduction

Le terme de l'informatique ubiquitaire (*ubiquitous computer*) introduit par Weiser [74] signifie une informatique omniprésente, c'est à dire, qui existe partout. Elle est aussi appelée informatique diffuse (*pervasive computer*) pour signifier qu'elle s'introduit partout. Ce terme est également utilisé pour décrire une sorte de traitements qui peuvent êtres effectués à tout moment, n'import où, et sur divers équipements informatiques [1]. Cette omniprésence est due à l'intégration de la technologie dans les objets qui entourent notre environnement de la vie quotidienne. Elle permet de relier les utilisateurs aux différentes ressources informatiques. Ainsi, ces derniers peuvent accéder à l'information et aux services souhaités, au moment voulu et dans n'importe quel endroit.

L'informatique ubiquitaire va au delà d'une simple intégration de la technologie dans l'environnement, elle vise même à la rendre invisible. C'est la notion de disparition de la technologie dans l'environnement de la vie quotidienne, comme le souligne Weiser "*the most profound technologies are those that disappear*" [74]. Selon Streitz et Nixon [67], cette disparition comprend deux aspects : le premier concerne la disparition physique, tandis que le deuxième concerne la disparition mentale.

La disparition physique est le résultat des grandes innovations technologiques qui tendent vers la miniaturisation des dispositifs électroniques et informatiques. Il est d'ores et déjà devenu possible d'enfouir des systèmes électroniques complexes dans les objets courants. C'est objets sont munis d'une capacité de calcule et de communication et deviennent ainsi des équipements intelligents. Ces derniers vont des terminaux mobiles (PC portables, assistants PDA et PC de poche) à des équipements plus spécialisés, tels que les badges d'identification, les vêtements intelligents, les téléphones portables, les appareils photo numériques, les capteurs, les actionneurs ou tout autre équipement de contrôle/commande. La communication entre ces équipements intelligents est supportée par plusieurs standards de communication développés dans le cadre des réseaux sans fil. Grâce à cette forte présence des équipements intelligents et leurs capacités de communication à l'aide des réseaux sans fil, l'environnement devient intelligent et forme une interface naturelle avec le système d'informations. A titre d'exemple, un environnement intelligent peut être composé d'un certain nombre de capteurs (pour l'image, le son, le positionnement, etc.), de capacités de calcul, stockage et communication (ordinateur) et de dispositifs de présentation de l'information (télévision, enceintes, …). Cet environnement se comporte comme un compagnon, en suivant en continu l'utilisateur et en interaction avec lui si nécessaire. Cette interaction pourra se faire d'une manière réactive (l'utilisateur exprime un besoin) ou proactive (l'environnement s'aperçoit qu'une certaine action peut être utile à l'utilisateur).

La notion de disparition mentale de la technologie est quant à elle un aspect fondamental pour rendre la vie plus confortable à l'utilisateur. En effet, la croissance du nombre des équipements intelligents et leur diversité pose un sérieux souci pour l'utilisateur, concernant le temps et l'attention nécessaire qu'il doit consacrer pour interagir avec ces derniers. Il s 'agit dans cet aspect, de rendre les communications moins intrusives, utilisables d'une manière transparente à l'utilisateur. C'est ainsi que l'interaction homme - machine disparaisse au profit de l'interaction homme - information ou directement entres des personnes à travers un système de communication. On parle alors de la communication ambiante [57] qui exploite les possibilités de l'informatique diffuse pour réduire cet environnement technologique à une interface de communication simple et intuitive.

I.2 Caractéristiques d'un environnement ubiquitaire

I.2.1 L'hétérogénéité

Les équipements utilisés dans un environnement ubiquitaire sont très variés (ordinateur portable, PDA, smart phone, etc.). Ces derniers fonctionnent avec différents systèmes d'exploitation (tels que Windows, Linux, Windows CE, PalmOS, Symbian, etc.), et éventuellement avec des configurations matérielles et logicielles différentes. De plus, ces équipements s'appuient sur diverses technologies de communication fil et sans fil. L'hétérogénéité se manifeste alors sur trois niveaux :

- **L'hétérogénéité au niveau des architectures matérielles** : L'industrie informatique a connu récemment une évolution très rapide. La figure I.1 modélise de manière cyclique ce processus et montre les transitions entre les différentes périodes d'utilisation [51].

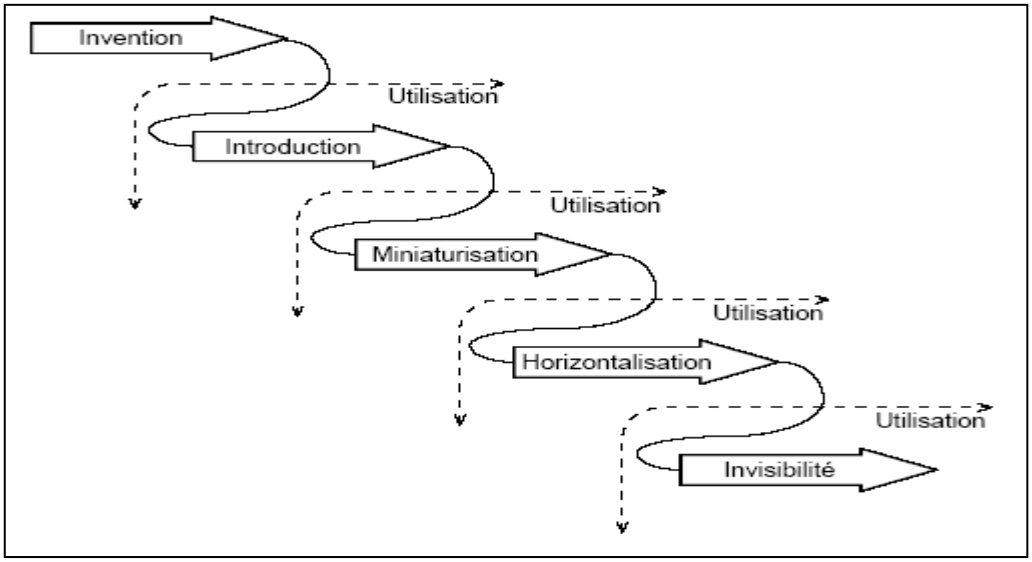


Fig. I.1 : Processus d'évolution technologique des équipements

Ce processus commence d'abord par une période d'invention et d'utilisation de la technologie, avant que cette dernière soit introduite dans notre vie de tous les jours. L'accent a été mis par la suite sur la miniaturisation afin de faciliter l'utilisation et de permettre la construction de dispositifs de plus en plus intégrés. Comme une alternative nécessaire à la complexité inhérente aux équipements conçus pour supporter n'importe quelle application informatique, L'horizontalisation propose la conception de multiples composants dédiés à une utilisation spécifique. Ces derniers effectueront leurs tâches par coopération, et non plus par intégration. Cette étape provoquera donc une croissance de l'hétérogénéité des entités informatiques car leur nombre augmentera. Comme dernière étape du processus, la technologie devienne invisible à l'utilisateur.

- **L'hétérogénéité au niveau des composants logiciels** : L'apparition des systèmes d'exploitation dans les années 60 a offert aux applications une plate-forme d'exécution indépendante de la couche matérielle sous-jacente. Avec le temps, l'hétérogénéité se pose au niveau du logiciel s'exécutant au-dessus du système d'exploitation. L'indépendance par rapport au système d'exploitation est réalisée grâce à l'utilisation des langages de programmation interprétés tels que Java et C#, ou des langages de script tels que

JavaScript et Python. Un programme écrit, par exemple en Java, est censé s'exécuter partout sur une machine virtuelle Java.

- **L'hétérogénéité au niveau de l'infrastructure de communication** : Comme le montre la figure I.2, plusieurs technologies filaires et sans fil sont cohabitées afin d'accéder au médium de communication. Il existe une large gamme de technologies d'accès développées ces dernières années. Nous citons quelques unes, sans être exhaustif. Pour les réseaux cellulaires, on trouve les technologies : GSM (*Global System for Mobile Communications*), GPRS (*General Packet Radio Service*), UMTS (*Universal Mobile Telecommunications System*) qui offrent une large couverture géographique, nationale voire internationale, à des débits limités. Pour les réseaux filaires, PSDN (*Packet-Switched Data Network*) et DSL (*Digital Subscriber Line*) offrent des débits importants. Les réseaux sans fil, tels que Wi-Fi (IEEE 802.11a/b/g) ou Bluetooth, pour des communications courtes/moyennes portée, offrent les mêmes services que l'Ethernet sans avoir besoin d'une infrastructure fixe. Par conséquent, Les terminaux seront équipés de plusieurs interfaces réseau pour pouvoir utiliser la totalité des réseaux d'accès déployés.

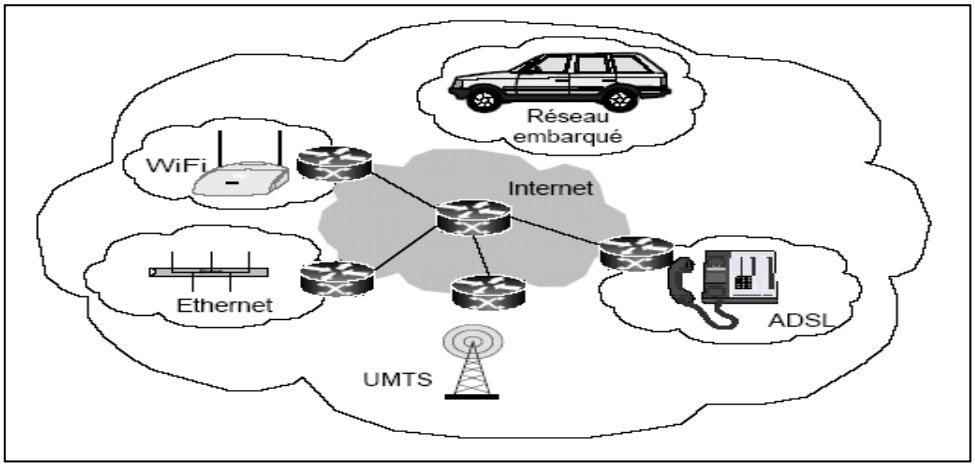


Fig. I.2 : Architecture d'accès

I.2.2 Limitation des ressources

La miniaturisation des équipements est l'un des facteurs qui a poussé vers l'informatique mobile. La plupart des équipements mobiles sont conçus d'une manière à être facilement transportables par les utilisateurs. Ils sont petits non seulement en terme du facteur forme, mais aussi en terme de puissance de traitement, mémoire, la taille d'affichage et l'énergie de la batterie. Tous ces facteurs limitent le type de traitement qui peut être effectué sur ces équipements. Malgré les progrès technologiques, les capacités de ces équipements resteront, sans doute, toujours limités par rapport aux PC de bureau.

I.2.3 Limitation des réseaux

La connexion sans fil utilisée dans les équipements mobiles est souvent limitée en bande passante et instable par rapport à la connexion filaire. A présent, les technologies standard sans fils incluent le Blutooth, Wi-Fi, GPRS, IEEE 802.11a, IEEE 802.11b, IEEE 802.11g et d'autres en cours de développement. La vitesse lente de ces technologies par rapport au réseau filaire a été toujours une barrière pour exécuter de grandes applications et déplacer des données volumineuses sur les mobiles.

I.2.4 Une mobilité très élevée

Les utilisateurs dans un environnement pervasif sont très mobiles. Sans êtres attachés à un équipement fixe, les utilisateurs peuvent se déplacer librement d'une place à une autre. Par exemple, un utilisateur dans sa voiture utilise un afficheur numérique pour regarder une conférence. En se déplaçant de la voiture vers la maison, il transfère la conférence à son PDA. Une fois qu'il est arrivé à la maison, il continue de regarder la vidéo conférence sur son PC. Cette nature élevée de mobilité provoque un changement continu de l'environnement, et du contexte qui peut affecter l'exécution de certaines applications.

I.2.5 Applications distribuées

Les environnements ubiquitaire sont de nature distribuée. En effet, les dispositifs qui se trouvent dans un tel environnement se situent dans des endroits différents et proposent leurs propres services. Ces dispositifs sont interconnectés via des réseaux filaires ou sans fil, et exploitent les infrastructures (middlewares) de communication développées dans le cadre des systèmes distribués telles que CORBA, SOA, etc. Ces infrastructures permettent de présenter les services, les exécuter, et les échanger entre les différents dispositifs si nécessaire.

I.2.6 La sensibilité au contexte

Le dynamisme de l'environnement provoque des changements dans le contexte des applications. Les premières applications sensibles au contexte, en informatique ubiquitaire, étaient sensibles à la localisation, en se basant notamment sur le système GPS *(Global Positioning System)* [73]. Le contexte peut comporter d'autres types d'informations tels que : les préférences et environnement de l'utilisateur, le terminal utilisé, les informations indiquant l'état physique ou émotionnel des personnes (la température corporelle, le rythme cardiaque, etc.)[55]. Le contexte à prendre en compte est toujours différent et dépend du but de l'application que l'on veut rendre sensible. Dans différents contextes, les utilisateurs accèdent aux même données et aux mêmes services mais reçoivent des réponses qui peuvent être différentes ou présentées différemment ou encore à des niveaux de détails différents [11]. Par exemple, un médecin consulte le dossier médical d'un patient à l'hôpital à l'aide d'un ordinateur de bureau. Dans un autre contexte, il consulte le même dossier mais chez le patient à l'aide d'un ordinateur de poche. Il peut aussi avoir une synthèse vocale du même dossier dans un autre contexte. Les informations contextuelles sont donc un élément très important à prendre en considération en informatique ubiquitaire.

I.3 Les middlewares de communication

La communication dans les environnements ubiquitaires nécessite de faire fonctionner tous les dispositifs qui y sont disséminés. Pour rappel, ces environnements sont composés des équipements hétérogènes, formant un système distribué à grande échelle, ouvert et dynamique, dans lequel les ressources disponibles et le contexte d'utilisation changent fréquemment. Dans ce cas, la gestion de la différence abstraite entre les besoins de haut niveau de l'utilisateur et les fonctionnalités fournies par les équipements nécessite une approche de communication flexible et modulaire, dans laquelle l'adaptation aux activités de l'utilisateur résulte de la construction dynamique de l'application à partir des ressources appropriées, et de la reconfiguration continuelle des applications pour s'adapter aux changements de l'environnement [62]. La mise en place d'un tel système de communication nécessite alors de prendre en compte toutes ces caractéristiques soulignées.

Le terme middleware (ou intergiciel) désigne la couche logicielle intermédiaire qui fournit un haut niveau d'abstraction de programmation permettant de masquer l'hétérogénéité des réseaux de communication, des ressources matérielles, des systèmes d'exploitation et des langages de programmation. Les middlewares s'intercalent entre les couches inférieures et les applications et services de haut niveau, comme le montre la figure suivante :

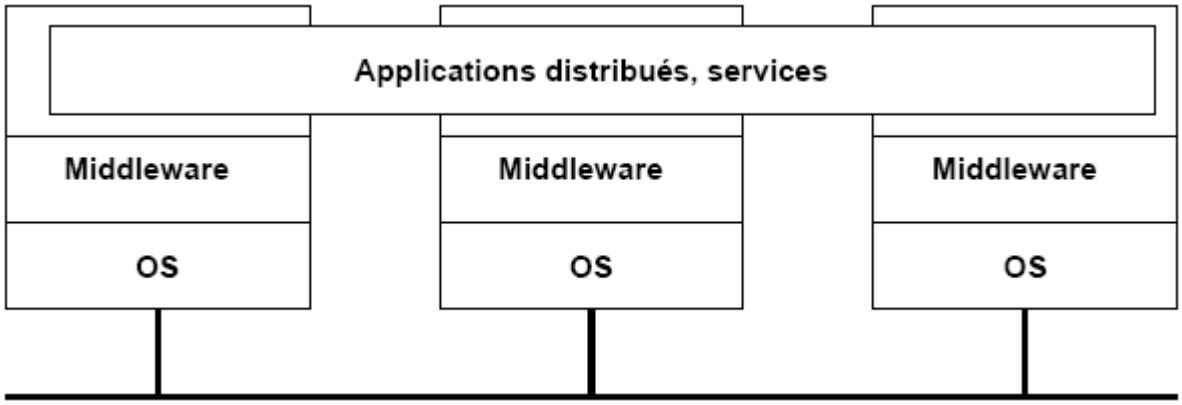


Fig. I.3 : Middlewares de communication

La problématique des systèmes distribués hétérogènes et ouverts est principalement abordée dans le cadre des architectures orienté services (SOA). Cependant SOA n'est pas un concept nouveau, l'émergence des services Web lui a donné une grande importance comme outil pour développer des applications distribuées [59]. En effet, L'idée remonte à plus d'une dizaine d'année. La création de standard comme CORBA, RMI, DCOM a déjà permis à certaines entreprises de répondre dans une certaine mesure aux problèmes d'intégration. Cependant, à cause d'utilisation des interfaces propriétaires, ces projets se sont avérés souvent coûteux et délicats. L'arrivée depuis quelques années des services Web et surtout leur large succès auprès des grands acteurs de l'informatique donne un second souffle aux SOA. Ainsi, les services vont pouvoir être implémentés sous la forme de services Web.

I.3.1 Middlewares à objets répartis

Les middlewares à objets répartis se basent sur le principe d'appel de procédures à distance afin de pouvoir invoquer des objets distants de façon transparente. Ces objets présentent des interfaces qui masquent leurs détails d'implémentation et facilitent leurs invocations. Ces middlewares sont devenus des standards. Trois d'entre eux se distinguent par le succès qu'ils ont eu : CORBA, RMI et DCOM.

I.3.1.1 CORBA

CORBA (*Common Object Request Broker Architecture*) est une architecture à objets distribués spécifiée par le groupe OMG (*Object Management Groupe*) [27]. C'est une norme de communication qui est utilisée pour l'échange entre objets logiciels hétérogènes. Un langage IDL *(Interface Definition Language)* décrit les traitements effectués et les formats de données en entrée et en sortie. Un bus applicatif, ORB *(Object Request Broker)* constitue le coeur de CORBA par lequel les requêtes sur les objets transitent.

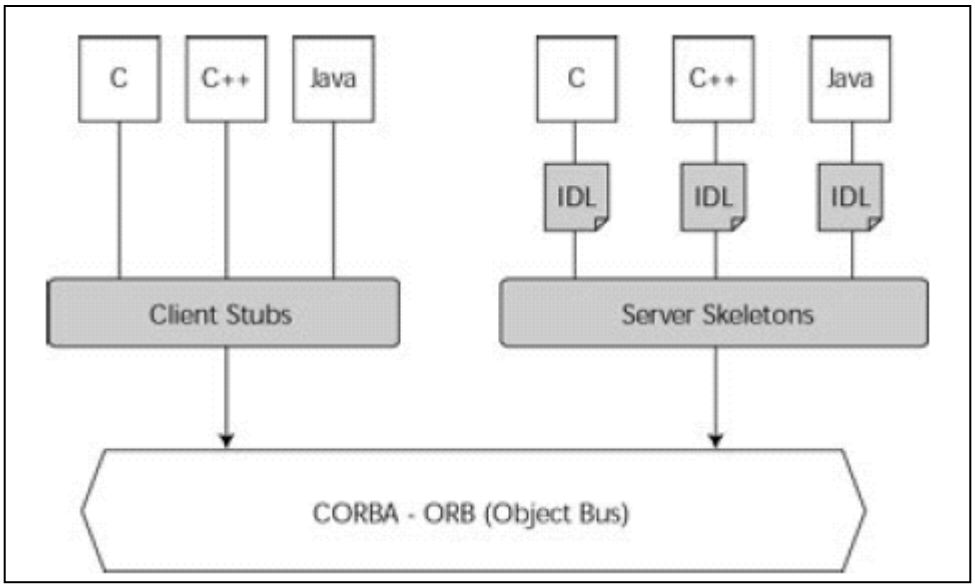


Fig. I.4 : Le modèle de l'architecture CORBA

La fonction première de l'ORB est de répondre aux requêtes en provenance d'une application ou d'un autre ORB. Il constitue donc en quelque sorte un bus par lequel transitent les objets distribués entre clients et serveurs. Cette communication reste cependant totalement transparente pour le développeur CORBA [71]. En effet, une fois l'ORB initialisé, les objets distribués s'utilisent comme s'ils étaient locaux, quel que soit le langage dans lequel ils ont été implémentés. Assurer l'interopérabilité entre objets implantés sur des plates-formes hétérogènes nécessite de disposer d'un mécanisme permettant de décrire l'interface de ces objets indépendamment de leur implémentation. C'est là le rôle du langage IDL. CORBA comprend deux types de modèles : le modèle objet et le modèle composant :

- **Le modèle objet** : Un objet distribué est une entité logicielle qui possède les mêmes caractéristiques qu'un objet classique (i.e. encapsulation des données et des méthodes, granularité, extensibilité, réutilisabilité, modularité), mais qui possède par rapport à ce dernier une certaine autonomie de fonctionnement, qui lui permet d'interopérer avec d'autres objets distribués. Lorsqu'un client invoque un objet distant, une implémentation de l'interface de l'objet, appelé souche (stubs) permet d'empaqueter les paramètres de la méthode invoquée et d'appeler l'objet distant en passant en premier par les squelettes (skeletons) qui font l'opération inverse (dépaquetage des données), et transmet l'appel à l'interface de l'objet. Au retour, les données sont empaquetées par les squelettes et transmis au client par l'intermédiaire de la souche qui dépaquette les résultats fournis.

- **Le modèle composant** : La notion de composant étend la notion d'objet et se place à un niveau plus haut de modularité. Cette notion est apparue après le constat que l'approche objet a une granularité très fine, qui la rend très peu adaptée aux systèmes complexes et aux systèmes distribués en particulier. Ces derniers sont composés de plusieurs entités coopérantes entre elles, et une fine granularité rendrait la conception et la maintenabilité difficiles, surtout si le développement de l'application est réalisé par plusieurs parties. Le modèle composant de CORBA CCM *(Corba Component Model*) qui est compatible avec les EJB *(Enterprise Java Beans*) décrit le déploiement, la configuration et la composition des composants. Deux types de composants sont définis dans ce modèle : les composants basiques qui sont une mise en forme des objets CORBA sous forme de composants et les composants étendus qui comportent plusieurs composants basiques.

I.3.1.2 RMI

Le modèle de calcul de Java c'est « écrire une fois, exécuter partout ». Ici, le modèle d'objets distribués est intégré dans le langage lui-même, en essayant de garder le plus possible la sémantique des objets locaux. Le modèle Java étant incapable de spécifier comment initier des calculs dans un espace d'adressage distant, Sun a définit un mécanisme appelé RMI [77] *(Remote Method Invocation),* qui permet au programmeur d'appeler des objets distants et le renvoi du résultat d'une méthode exécutée dans une machine virtuelle différente de celle de l'objet l'appelant.

RMI permet de créer des applications Java distribuées. Ces dernières implémentent les méthodes des objets Java distants qui peuvent êtres invoqués à partir de n'importe quelle machine qui possède la JVM (*Java Virtual Machine*).Un programme Java peut faire un appel à un objet distant une fois qu'il a obtenu sa référence, soit en cherchant dans un annuaire de services fourni par RMI, ou bien en recevant la référence comme argument en retour d'un appel. Les souches et squelettes, qui représentent respectivement les cotés client et serveur sont des classes compilées par le compilateur RMI (RMIC). Un client peut alors invoquer un objet distant qui se trouve dans un serveur. Le serveur lui même peut devenir client pour d'autres objets distants. RMI utilise la sérialisation d'objets pour empaqueter et dépaqueter les paramètres. Le protocole utilisé pour la communication est JRMP (*Java Remote Methode Protocol).*

La programmation avec RMI est facile une fois le développeur a acquis une certaine expérience avec le langage de programmation Java et les applications distribuées. Il ne contient pas un langage abstrait tel que IDL pour décrire les objets distants. De plus RMI supporte un nettoyage distribué des objets en mémoire. Cependant, RMI peut être utilisé uniquement avec le langage Java de part et d'autre de la connexion. Et la facilité d'utilisation du RMI provient du fait que ce middleware est maintenu simple et ne fournie pas des services comme CORBA. Cet aspect doit être pris en compte par le développeur.

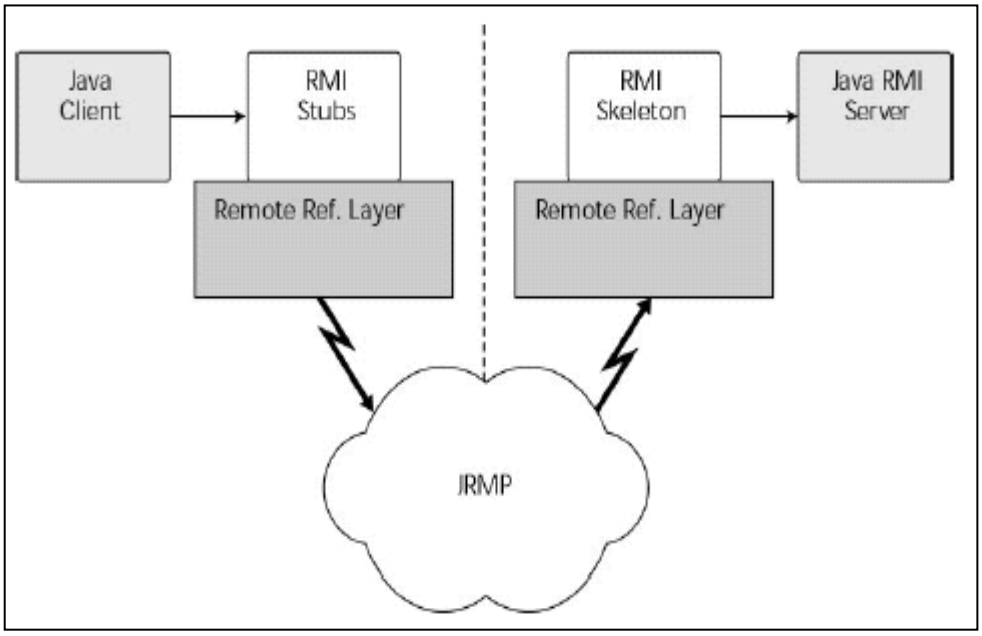


Fig. I.5 : Le modèle de l'architecture RMI

I.3.1.3 DCOM

Distributed COM (DCOM) [25] est une version distribuée du modèle de Microsoft COM (*Component Object Model)* [16], qui à été crée pour permettre la communication entre les applications Windows dans un environnement à base de composants. DCOM a étendu cette communication interprocessus à travers le réseau. DCOM permet d'appeler des objets distants en utilisant une couche de haut niveau au dessus du mécanisme DCE RPC *(Remote Procedure Call)* qui interagie avec les services de COM. Un serveur DCOM publie ses méthodes pour les clients sous forme d'interfaces. Ces dernières sont écrites en IDL, qui est similaire au langage C++. Le compilateur IDL crée des souches et des squelettes comme le compilateur IDL de CORBA. Mais la particularité de DCOM c'est que toutes les configurations de ses applications sont enregistrées dans la base de registre du système. L'utilisation de cette base de registre et le fait que DCOM soit une extension de COM, rendent exploitable les applications COM dans un environnement réparti. De plus, des librairies sont crées sous forme de fichiers qui décrivent l'objet distant qui peut être invoqué par le mécanisme COM. Un autre élément essentiel dans l'architecture DCOM est le Service Control Manager qui permet de gérer les composants du système (localisation, activation, enregistrement de référence). Le protocole utilisé est ORPC *(Object Remote Procedure Call).*

Le niveau de spécification binaire de DCOM permet l'utilisation de plusieurs langages pour coder les objets du serveur. Les interfaces binaires de DCOM peuvent être considérées comme des tables de pointeurs vers les implémentations des méthodes de l'interface. Comme RMI, DCOM supporte le nettoyage distribué des objets distants en mémoire. Cela est réalisé par un mécanisme qui vérifie si le client est encore actif. Du coté serveur, un conteur de référence est maintenu pour les clients. S'il atteint zéro, l'objet sera détruis. La plupart des utilisateurs associent DCOM au système d'exploitation Windows. Toutefois, il existe des ports pour les plateformes Unix, VMS, Apple, et Macintosh.

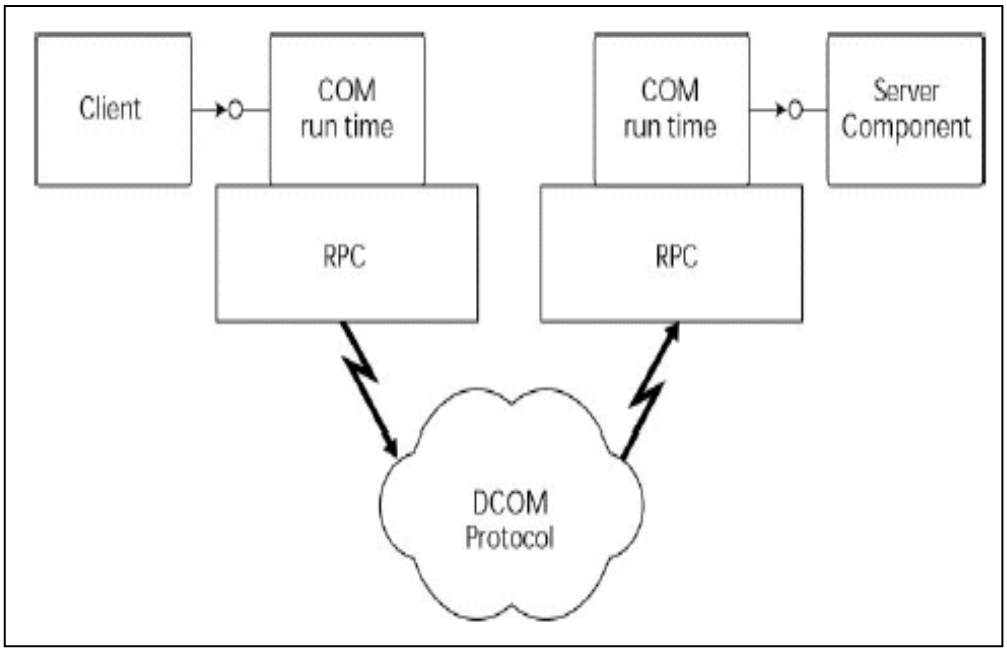


Fig. I.6 : Le modèle de l'architecture DCOM

Les approches que nous venons de présenter reposent sur des infrastructures bien particulières, qui nécessitent que les dispositifs se conforment à des contraintes spécifiques.

Cependant, les dispositifs constituant les environnements ubiquitaires peuvent être d'origine très diverses et il est nécessaire de prendre en compte leur hétérogénéité. D'autre part, une propriété essentielle à la mise en place de systèmes à grande échelle est la capacité d'ouverture, c'est-à-dire la possibilité d'intégrer facilement un dispositif qui n'a pas été conçu spécifiquement pour fonctionner dans ce système.

I.3.2 Architecture Orientée Service (SOA)

I.3.2.1 La notion de service

Un service est une brique logicielle autonome qui fournie une fonction bien définie [82] telle que l'analyse d'informations, la recherche d'informations, etc. Un service est autonome dans la mesure où il peut se suffire à lui même, toutefois il peut être publié et rendu disponible pour être utilisable par des tiers. Ainsi des services peuvent être utilisés tels quels ou bien être composés pour mener à terme un processus complexe et atteindre un objectif précis de plus haut niveau. D'un point de vue implémentation, un service peut être implémenté en utilisant différents paradigmes : objet, composant ou agent. Il est décrit et utilisé indépendamment de sa réalisation grâce aux mécanismes d'encapsulation et de liaison retardée. La liaison proprement dite n'est effectuée qu'au moment de l'exécution. Un service est décrit par son descripteur de service, sa spécification en quelque sorte. Le descripteur comporte des :

- **Informations fonctionnelles** : la sémantique des opérations, le comportement du service (pré condition, post-condition, invariant, exception, propriétés), l'interface du service.

- **Informations non fonctionnelles** : prix, politique, spécification des mesures de la qualité de service, information de déploiement, etc.

- **Informations additionnelles** : composées d'informations sur le service, non spécifiées par le fournisseur du service telles que les notes, les rapports d'utilisation, etc.

I.3.2.2 La notion de SOA

Voici la définition de SOA donnée par le site service-architecture.com [82]: « L'architecture orientée service est essentiellement une collection de services. Ces services communiquent entre eux à l'aide de données simple ou d'autres services coordonnant une certaine activité. SOA n'est pas quelque chose de nouveau, la première architecture orientée service pour beaucoup de personnes dans le passé était DCOM ou CORBA.».

Nous constatons bien que l'approche SOA prend ses origines dans les anciennes architectures. C'est une technologie d'information ou bien une stratégie par laquelle les applications utilisent les services disponibles sur le réseau tel que le *WWW (Word Wide Web)* [52]. Cette approche vient pour combler le manque de canevas architectural permettant de guider la conception, développer, intégrer et réutiliser des applications plus facilement et plus rapidement. De plus, comme le domaine de métier d'une entreprise est en constante évolution, les systèmes de cette entreprise doivent suivre cette évolution pour répondre à de nouveaux problèmes. Une architecture orientée service permettrait d'assembler des composants et des services pour construire et fournir dynamiquement des solutions à ces problèmes. Le SOA peut être à la fois une architecture au sens propre du terme et un modèle de programmation.

I.3.2.3 Principe de SOA

La figure ci-dessous illustre une vision simpliste d'une architecture orientée service et des interactions entre les différents acteurs à savoir : le fournisseur de service *(service provider)*, l'utilisateur de service *(service user)* et le registre ou l'éditeur de service *(service publisher)*. Un fournisseur de service développe le service et le publie dans un annuaire ou registre. L'utilisateur de service aussi appelé consommateur ou client cherche des services qui satisfassent la tâche qui veut accomplir. Le registre de service peut être aussi couplé avec un composant qui stocke des informations additionnelles sur chaque service tel que le coût d'utilisation du service.

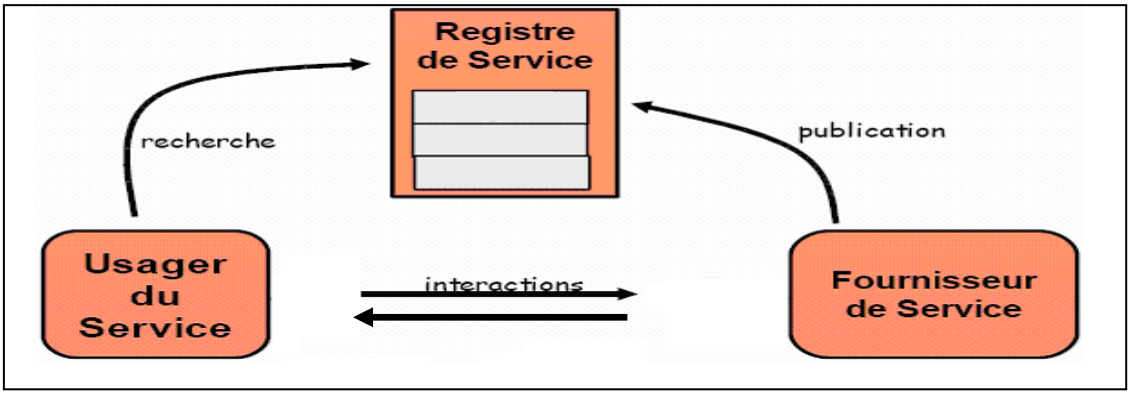


Fig. I.7 : Acteurs classiques d'une architecture orientée services

Plusieurs services peuvent être utilisés ensemble d'une manière coordonnée. Le service agrégé ou composé peut être utilisé pour satisfaire des exigences plus complexes de l'utilisateur. En fait, une façon de regarder à une SOA est comme une approche pour connecter des applications (exposées comme services) de telle sorte qu'elles puissent communiquer entre elles et chacune tire profit de l'autre. De plus, SOA est une manière de partager des fonctions (typiquement des fonctions métiers) de façon flexible et universelle.

I.3.3 Comparaison entre l'Architecture distribuée et l'Architecture Orientée Service

Une comparaison entre l'Architecture distribuée (middleware à objets réparties) et l'Architecture Orientée Service est illustrée par le tableau suivant :

Architecture distribuée	Architecture Orientée Service
Orientée fonctionnalité	Orientée processus
Élaborée pour durer	Élaborée pour gérer le changement
Processus de développement long	Processus de développement court
Centrée sur la réduction des coûts	Centrée sur le support aux métiers
Organisée au sein d'une application	Permet l'orchestration et la composition
Système fortement couplé	Faible couplage, agile
Orienté objet	Orienté message
L'implémentation doit être connue	L'implémentation reste abstraite
Technologie homogène	Technologie hétérogène

Tableau I.1 : Architecture SOA et Architecture distribuée

I.4 Les services Web

I.4.1 Définitions et caractéristiques

L'architecture SOA n'est pas un nouveau concept, l'émergence des services Web lui a donné une grande importance comme outil pour développer des applications distribuées [59]. Un service Web est un service qui communique avec des clients à travers un ensemble de protocoles et des technologies standards. Ces standards des services Web sont implémentés dans des plateformes et produits par des grands concepteurs de logiciels. Il est devenu possible pour les clients et les services de se communiquer d'une façon consistante à travers un large spectre de plateformes et d'environnements d'exploitation. Cette universalité a rendu les services Web comme l'approche dominante pour implémenter une SOA.

Le terme service est de plus en plus associé aux services Web [40] qui sont actuellement largement utilisés et devenus un standard de fait. Selon IBM [26], « Web services sont auto contenus, applications modulaires, accessibles via le Web à travers des langages standards et ouverts, qui fournissent un ensemble de fonctionnalités pour les entreprises ou les individus ». Cette définition met l'accent sur deux points. Le premier point est que un service Web est vu comme une application accessible pour les autres applications à travers le Web. Le second point, les services Web sont ouverts, ce qui signifie que les services publient des interfaces qui peuvent être invoquées par le biais de messages standards. Mais les composants dans l'architecture CORBA sont aussi modulaire et auto contenus mais ils ne sont pas des services Web.

Une autre définition qui raffine plus la précédente est celle donnée par W3C *(World Wide Web Consortium)* [3] : « un service Web est un système logiciel identifié par un URI, dont les interfaces publiques et leurs liaisons sont décrites en utilisant XML. Sa définition peut être découverte par les autres systèmes logiciels. Ces systèmes peuvent alors interagir avec le service Web avec la manière décrite dans sa définition, en utilisant des messages basés sur XML et transmis par des protocoles internet. » La définition de W3C insiste sur le fait que les services Web doivent être capable d'être définis, décrits, et découverts. Les services ne doivent pas seulement fournir des informations statiques, mais aussi des actions qui affectent l'environnement.

Les services Web développés par différentes organisations peuvent être intégrés pour satisfaire les exigences de l'utilisateur. Cette intégration est basée sur les standards communs des services Web, indépendamment des langages d'implémentation, et des plateformes d'exécution. En général, les services Web possèdent les caractéristiques suivantes qui leurs permettent une meilleure intégration dans les environnements hétérogènes :

- **Faiblement couplé** : dans le développement des logiciels, le couplage se réfère au degré de dépendance entre les différents composants et modules du logiciel. Comparativement aux composants fortement couplés tels que DCOM ou CORBA, les services Web sont autonomes et peuvent opérer indépendamment l'un de l'autre. La caractéristique de faible couplage permet aux services Web de localiser d'autres services dynamiquement, et de communiquer avec eux.

- **Accessibilité universelle** : les services Web peuvent être définis, décrits, et découverts à travers le Web qui permet une accessibilité facile. La description et l'annonce des services Web permettent aux utilisateurs du Web de localiser les services appropriés.

- **Langages standards** : quoique le noyau des services Web puisse être implémenté par différents langages de programmation, les interfaces des services Web sont décrites par un langage uniforme et standard qui est XML.

I.4.2 Langages standards des services Web

Les différentes couches qui constituent la pile des standards des services Web sont :

- XML *(Extensible Markup Language).*
- SOAP *(Simple Object Access Protocol).*
- WSDL *(Web Services Definition Language).*
- UDDI *(Universal Discovery Description Integration).*

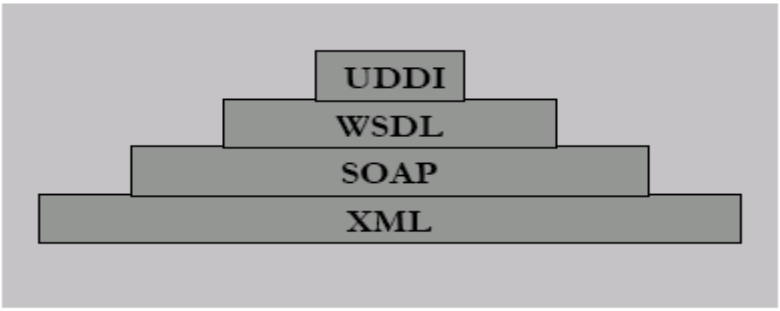


Fig. I.8 : Pile des standards des services Web

XML: qui est une norme du W3C depuis 1998, constitue un métalangage pour définir des langages de balisage. Il conserve la syntaxe de SGML et reprend ses qualités à savoir : un balisage logique hiérarchique, une description facultative du document qui peut être extérieure ou intégrée au document, une portabilité quasi universelle. C'est ainsi que les fonctionnalités les plus lourdes et les plus rarement utilisées de SGML ont été supprimées dans XML. Ce dernier permet également de créer des pages Internet sophistiquées, de structurer un document, de décrire des données. Il est aussi utilisé pour les échanges entre machines et/ou programmes, mêmes étrangers entre eux. XML facilite l'utilisation des services Web : d'abord par la séparation du contenu d'un document, de sa structure et de sa représentation qui facilite l'échange d'informations entre les différents partenaires. Ensuite, en permettant la composition de services plus complexes à travers la définition des enchaînements entre les services. Enfin, en favorisant l'émergence de standards d'industrie dans la mesure où les structures de données sont réutilisables.

SOAP : est un standard proposé par W3C pour l'échange des données. Ces dernières sont d'abord enveloppées dans des messages en format XML, puis échangées par des appels de procédures à distance (RPC) en utilisant HTTP/SMTP/POP comme protocole de communication. SOAP est la plupart du temps utilisé sur la couche de transport HTTP et il n'est pas perturbé par les pare-feux *(firewall).* Il est aussi indépendant de la plate-forme et du langage des services. Si un consommateur souhaite utiliser un service Web, il va construire avec une application cliente un message SOAP conforme au document WSDL du service. Un message SOAP correspond à une enveloppe contenant jusqu'à 2 parties : la partie *header* dans laquelle il est possible d'ajouter ou de modifier des informations contextuelles (signatures, destinataires…) et la partie *body* qui contient le message à transmettre.

WSDL : est un langage de description des services proposé par W3C. Il se base sur une grammaire XML pour décrire de manière abstraite et indépendante du langage de programmation, l'ensemble des fonctionnalités offertes par un service. Il permet de connaître les protocoles, les serveurs, les ports, le format des messages, les entrées, les sorties, les exceptions possibles et les opérations réalisées par un service Web. Ainsi, WSDL propose une double description du service : Une vue abstraite *(le quoi)* qui présente simplement les opérations et les messages des services. Une vue concrète *(le comment, et le où)* qui présente les choix d'implémentation faits par le fournisseur du service (détermination du protocole et du mode d'accès).

UDDI : c'est un registre (annuaire) qui décrit comment publier et découvrir un service Web. Les fournisseurs de services peuvent alors enregistrer et publier leurs services dans cet annuaire. Ce dernier contient des metadonnées qui peuvent être utilisées pour la recherche de services (noms, IDs, catégories, types, etc.). UDDI contient trois types de pages : blanches, jaunes et vertes. Chaque type de pages donne des informations de nature différentes. Les pages blanches contiennent des informations sur le fournisseur du service (nom, sa description, etc.), les pages jaunes indiquent ce que fait le service, et les pages vertes, plus techniques, décrivent comment utiliser un service Web et les moyens d'interaction avec un fournisseur en se basant sur la description de services fournis en WSDL.

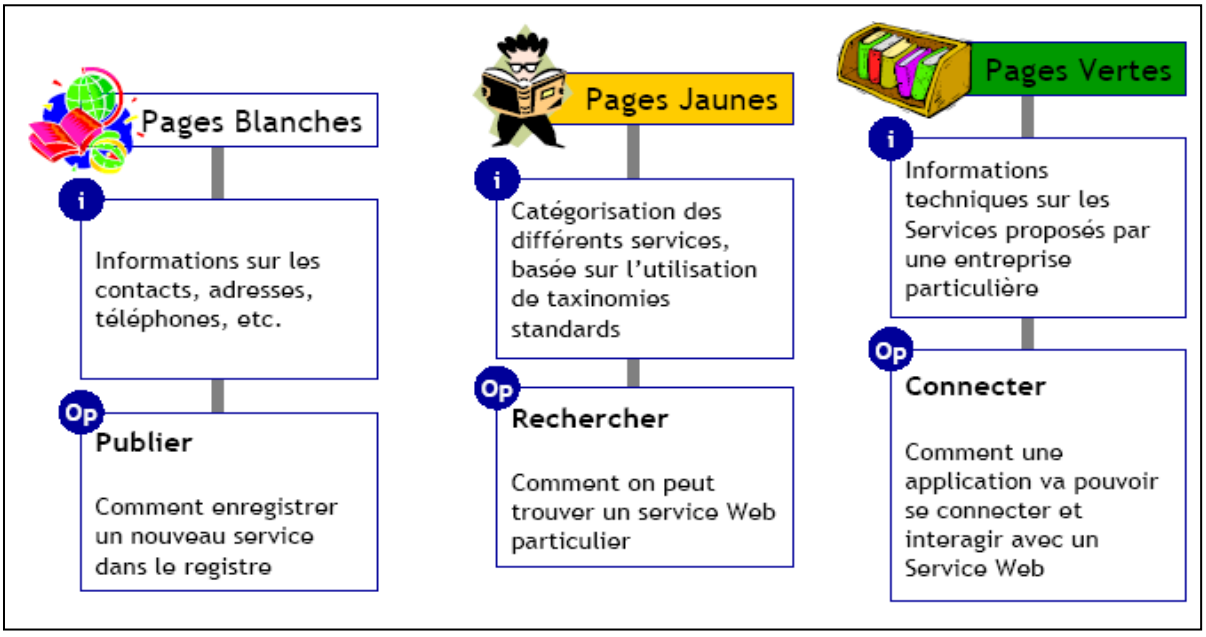


Fig. I.9 : les pages UDDI

UDDI est un registre distribué dans lequel l'information est répliquée sur plusieurs sites, en conséquence, un fournisseur de services ne doit publier sa description de service que vers un seul noeud du réseau de registres. La recherche se fait donc par interrogation du registre qui permet de trouver le service désiré et d'obtenir son URL. Des enregistrements peuvent êtres également faits auprès du registre UDDI afin de recevoir des notifications concernant des changements dans ce registre: ajout, retrait et modification au niveau des services.

I.4.3 Scénario général d'utilisation

Comme le montre la figure ci-dessus (Fig. I.10), l'utilisation d'un service Web fait intervenir trois acteurs principaux : les fournisseurs des services *(Web Services Provides),*

l'annuaire *(Directory)* ou registre de services, et les clients (utilisateurs) des services Web *(Web Service Client).* Le fournisseur doit d'abord décrire le service Web à fournir à l'aide d'un langage de description appelé WSDL qui se base sur une grammaire XML. Il doit ensuite publier cette description du service dans un registre de service appelé UDDI, afin de rendre son service disponible et connu par les clients. Ces derniers peuvent alors accéder au registre UDDI et rechercher le service Web désiré. Après l'avoir trouver, ils utilisent sa description pour créer un lien avec le fournisseur de ce service à l'aide d'un protocole de communication appelé SOAP qui enveloppe les messages dans un format XML. Dans la section suivante, nous détaillons ces différents langages standards des services Web à savoir : XML, UDDI, WSDL et SOAP :

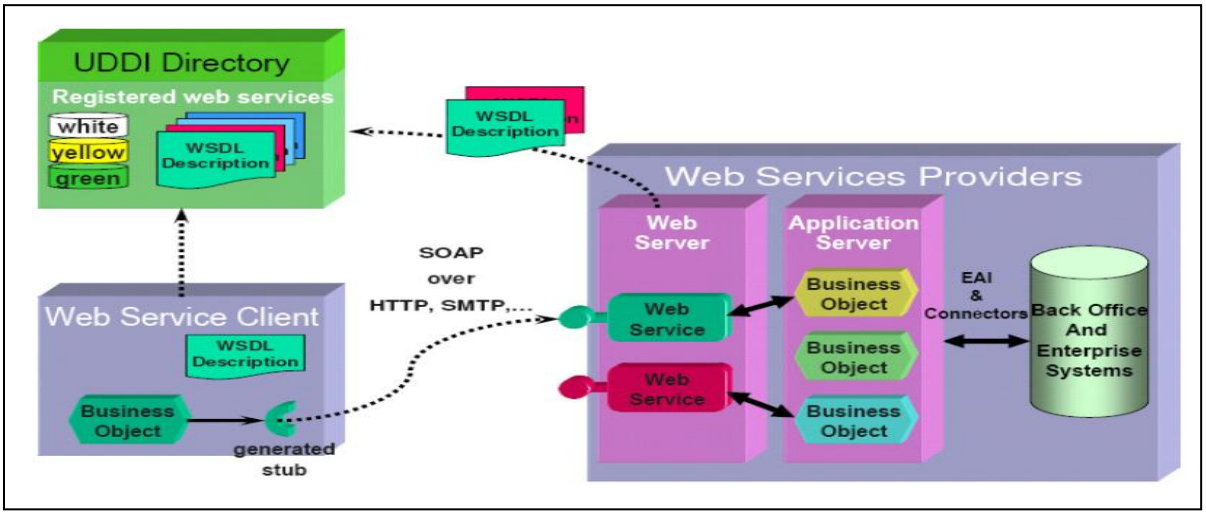


Fig. I.10 : Schéma d'interaction de services Web

I.4.4 Le Web sémantique

L'environnement du Web est ouvert, et les demandeurs de services peuvent êtres des machines plutôt que des personnes. Pour implémenter une interopérabilité fiable et à grande échelle des services Web, il faut que ces services soient interprétables par ces machines. Ceci implique une description plus riche des services Web qui se base sur les aspects sémantiques plutôt que syntaxiques. Le Web sémantique est une vision qui permet d'encoder les services Web dans une forme non ambiguë et compréhensible afin qu'elle soit utilisable par des machines pour l'automatisation, l'intégration et la réutilisation à travers des applications variées. Comme souligné dans [41], l'objectif du Web sémantique est d'utiliser une spécification sémantique pour automatiser la découverte, l'invocation, la composition, et l'exécution des services Web.

- **Découverte automatique** : nécessite la localisation automatique des services Web fournissant un service particulier qui répond aux propriétés demandées. Différentes approches ont été proposées dans la littérature pour réaliser la découverte dynamique de services. Toutes ces approches reposent sur des modèles sémantiques afin d'avoir l'information nécessaires et la façon de l'interpréter.

- **Composition automatique** : l'objectif de la composition de service est de créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services existants. Étant donnée une spécification de haut niveau des objectifs d'une tâche particulière, la composition de service implique la capacité de sélectionner, de composer

et de faire interopérer des services existants. Des travaux récents commencent à explorer des approches combinant des outils du Web sémantique et de planification à base de l' IA de manière à pouvoir composer automatiquement des services en vue d'atteindre des fonctionnalités prédéfinies.

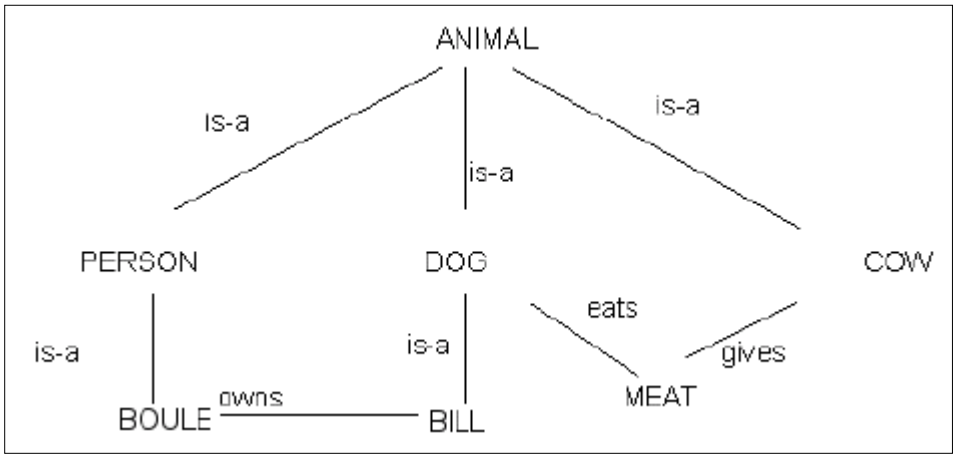
- **Exécution automatique** : nécessite un programme ou un agent qui exécute automatiquement un service Web. La sémantique Web offre des APIs interprétables par la machine pour exécuter les services, en spécifiant les données en entrée et en sortie, ainsi la façon d'exécuter le service automatiquement.

Le Web sémantique vise aussi à assurer de la manière la plus automatique qui soit, l'interopérabilité, la mise en oeuvre de traitements et de raisonnements complexes sur les connaissances tout en offrant des garanties élevées sur leur validité [15]. Ces connaissances doivent en outre s'appuyer sur des ontologies pour être partagées et garantir leurs interprétations opérationnelles.

I.4.4.1 Ontologie : définition

Une ontologie permet de fournir le sens des symboles utilisés pour construire un modèle du monde. Elle permet donc de comprendre la signification des termes utilisés dans l'environnement. Comme souligné dans [70], l'ontologie est une connaissance partagée d'un certain domaine d'intérêt et qui peut être utilisée comme un cadre commun. Elle est aussi définie comme une description formelle explicite des concepts dans le domaine du discoure (classes parfois appelées concepts) [30]. Chaque concept possède un ensemble de propriétés (attributs) qui décrivent ces caractéristiques. Des restrictions sont aussi définies sur ces propriétés.

Une ontologie est représentée comme un ensemble structuré sous la forme d'un graphe orienté qui contient des nœuds et des arcs. Les noeuds représentent le vocabulaire d'un domaine particulier, et les arcs représentent les relations (ou rôles) nommées entre les concepts. À partir de cette structure, la sémantique de chaque mot est déduite par les relations que ce mot possède dans l'ontologie, ce qui permet de restreindre les interprétations possibles. Voici un exemple d'un réseau sémantique avec les concepts en noeud et les relations en arc.



Cependant pour être exploitable par une machine, une ontologie doit respecter certaines règles : (1) être définie par une syntaxe formelle et une sémantique non ambiguë; (2) permettre la déduction de nouvelles connaissances présentées implicitement dans l'ontologie; et (3) doit posséder un niveau d'abstraction permettant son extension afin d'assurer sa pérennité.

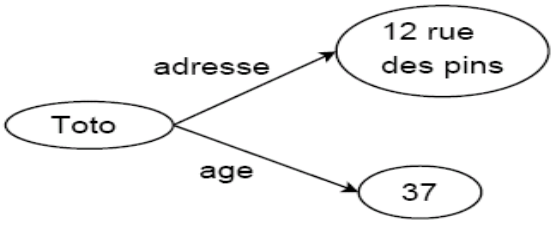
I.4.4.2 Langages du Web sémantique

Aujourd'hui, il existe différents langages de spécification avec des degrés d'expressivité différents, tels que RDF (*Ressource Description Framework),* DAML *(DARPA Agent Markup Language),* OIL *(Ontologie Inference Layer),* DAML+OIL, OWL *(Web Ontology Language),* OWL-S *(OWL-Services).*

RDF : C'est un langage d'ontologie dont la syntaxe se base sur le format XML. Ce dernier fournie une syntaxe de la structure des documents, mais n'impose aucune contrainte sémantique sur le sens de ces documents. RDF est développé par W3C pour exprimer des métas donnés semi structurées. Un document RDF contient des connaissance dans une forme de donnée appelée triplet {propriété/ prédicat, ressource/ sujet, valeur/objet}. Les triplets identifient les relations entre les concepts abstraits. L'ontologie RDF utilise l'URI (*Unified Ressource Identifier*) pour référencer d'autres concepts qui existent dans d'autres ontologies sur le Web. Par exemple une personne appelée Toto âgée de 37 ans et qui habite à l'adresse « 12 rue des pins » peut être représentée avec le langage RDF comme suit :

```
<rdf :RDF>
 <rdf :Description about='Toto'>
 <rdf :Property about='adresse'>
 12 rue des pins
 </rdf :Property>
 <rdf :Property about='age'>
 37
 </rdf :Property>
 </rdf :Description>
</rdf :RDF>
```

Représentation RDF/XML de : Toto 37 ans ans qui habite au 12 rue des pins



Représentation graphique de : Toto 37 qui habite au 12 rue des pins

Cependant, RDF ne permet pas d'exprimer des formes sophistiquées tels que le typage des données des propriétés, les caractéristiques de ces propriétés, l'énumération et l'utilisation des contraintes dans les concepts. C'est ce manque de sémantique que OWL comble par l'apport d'un vocabulaire plus riche.

OWL : Ce langage est initialement développé par le DARPA *(Defence Advanced Research Projet Agency)* sous le nom de DAML+OIL. OIL *(Ontologie Inference Layer*) est une initiative européenne, qui est fusionnée avec DAML pour former le langage DAML+OIL. W3C a renommé DAML+OIL par OWL et standardise ce langage en 2003. OWL permet la création des ontologies pour n'importe quel domaine, ainsi leur extension pour la description des ressources Web spécifiques. Il étend les concepts définis dans le schéma RDF afin d'intégrer d'autre types d'informations. Ce langage offre plus de flexibilité pour la définition de classes et de relations complexes, en utilisant plusieurs types de constructeurs (intersection, union, restrictions diverses, transitivité, cardinalité etc.). En s'appuyant sur une logique de description, OWL possède une sémantique formelle claire, ce qui permet de le doter de mécanismes de raisonnement (classification, inférence, vérification de cohérence).

OWL-S : OWL est à la base de la création de OWL-S (S pour Services), un langage très répondu pour la description des caractéristiques des services Web. OWL-S est un ensemble d'ontologies qui supporte la riche description des services Web, et facilite leur composition automatique. OWL-S comprend trois sous ontologies inter reliées, connues comme : *service profile, service model,et service grounding :*

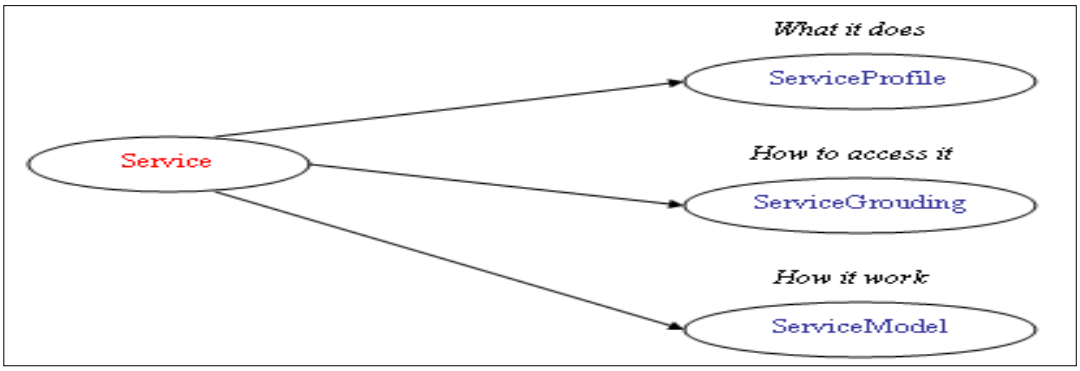


Fig. I.11: Modèle OWL-S

Service profile (Ce que fait le service) : description du service en terme d'offre à la communauté, constituée principalement des : entrées, sorties, préconditions, et effets qui sont résumés en IOPE (Inputs, Outputs, Preconditions, Effects).

Service model (Comment il le fait) : description du service en terme d'algorithme. Cette information décompose le service en terme de services plus simples. C'est en fait un modèle de processus qui décrie comment le service fonctionne.

Service grounding (Comment accéder concrètement au service) : spécifie les informations nécessaires pour l'invocation et l'exécution du service. C'est une description du service en terme des messages échangés. Cette information décrit comment accéder au service en faisant le lien avec un protocole de communication tel que SOAP. Cette partie spécifie également pour chaque type un format pour envoyer les données.

OWL-S possède un mécanisme extensible pour les paramètres du service. Ce mécanisme permet une description additionnelle des attributs non IOPE du service.

I.5 Convergence du Web et des technologies ubiquitaires

I.5.1 Représentation de connaissances

Les technologies de l'informatique ubiquitaire suscitent des besoins en termes de modèles et de mécanismes élaborés de représentation, de gestion et de partage des connaissances entre des applications et services ubiquitaires hétérogènes. Le Web sémantique constitue une réponse prometteuse à ces besoins. Du point de vue des applications ubiquitaires sensibles au contexte, le Web sémantique permet de garantir une certaine interopérabilité sémantique des connaissances du contexte. Il permet d'une part, le partage d'une représentation standard et intelligible de la sémantique des connaissances contextuelles entre les différentes applications et d'autre part, l'automatisation des processus de découverte, d'extraction et de raisonnement sur ces connaissances contextuelles [15]. Ainsi, les travaux menés sur le Web sémantiques semblent offrir un cadre intéressant pour la composition dynamique de fonctionnalités dans les environnements ubiquitaires.

L'approche sémantique des services Web est une étape vers la découverte et la composition dynamique de service, dans laquelle des systèmes intelligents tentent de construire des compositions de service à partir des exigences de l'utilisateur sans faire une sélection

manuelle des services. En se basant sur des techniques de représentation de connaissances, avec des ontologies décrivant un domaine d'une manière formelle tels que OWL-S, plusieurs chercheurs exploitent des techniques de planification à base de l'IA pour la composition de service en traitant la composition de service comme un problème de planification. À partir de la requête de l'utilisateur et un ensemble de services décrits comme actions, le planificateur doit trouver une collection de services Web qui satisfait cette requête.

I.5.2 Les services d'agents

Les seules connaissances dont dispose un service Web ne concernent que son état interne et il est incapable de percevoir son environnement. Par conséquent, le service Web ne peut fournir un service personnalisé à son utilisateur. L'introduction du paradigme agent vise à rendre les services Web plus autonomes, plus intelligents, et capables de percevoir leurs environnements (utilisateurs, autres agents, système). Ces propriétés sont nécessaires pour assurer l'auto organisation dynamique de services Web indépendants et pour répondre au mieux aux requêtes des utilisateurs. On parle alors de services d'agents. De plus, en dotant un service Web de la capacité de proactivité des agents, il peut par exemple s'auto exécuter, envoyer des alertes ou des notifications aux utilisateurs pour les informer de sa présence ou de son état d'exécution, sans nécessiter de requêtes d'invocation.

I.5.3 Architectures à agents orientées services

Nous avons présenté l'architecture orientée service, qui constitue une tendance actuelle pour l'intégration de fonctionnalités hétérogènes. Elle vise à permettre l'interopérabilité de services conçus indépendamment, et la gestion de systèmes ouverts, dans lesquels des services peuvent apparaître ou disparaître. Ce type d'architecture est actuellement disponible dans le cadre des services Web. Des travaux actuels visent à intégrer les services d'agents dans une architecture orientée service. On parle ainsi des architectures à agents orientés services. Cela permet de doter l'architecture orientée service d'une capacité d'autonomie, de proactivité et de coopération entre les agents. On parle alors d'orchestration ou de composition de services d'agents. Un exemple de ce type d'architecture est celui de l'intergiciel JADE WSIG qui est une extension de l'intergiciel agents JADE, où l'interaction entre agents repose sur l'invocation de services Web (Fig. I.12) [29].

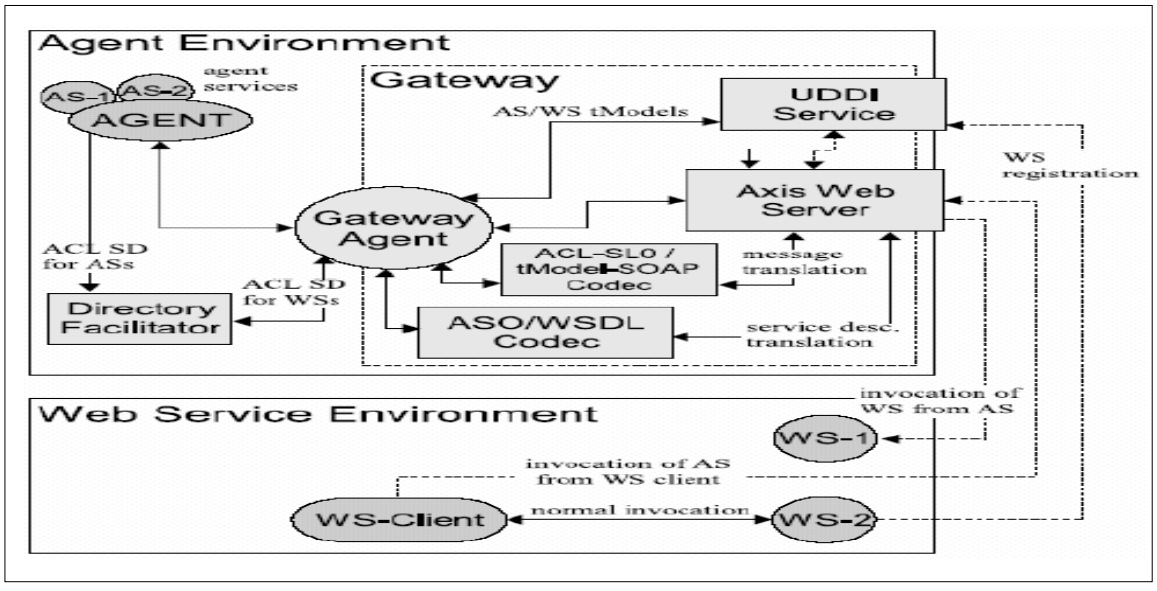


Fig. I.12 : Architecture de l'intergiciel agent orienté services WSIG

I.6 Synthèse

Dans ce premier chapitre, nous avons présenté le cadre général de notre travail, qui porte sur les services dans les environnements ubiquitaires. Nous avons exposé les différentes caractéristiques de ces environnements, et les difficultés inhérentes aux applications dans ce domaine, notamment les problèmes d'hétérogénéité et de prise en compte du contexte. Cela nécessite donc de construire sur des infrastructures de communication pour faire fonctionner tous les dispositifs de l'environnement ubiquitaire. Nous avons commencé par les premiers standards conçus pour la programmation distribuée qui sont au nombre de trois: CORBA *(Common Object Request Broker Architecture)* soutenu par l'OMG *(Object Modeling Group)*, DCOM *(Distributed Component Object Model)* de Microsoft et RMI *(Remote Method Invocation)* de Sun. Ces modèles offrent des mécanismes de récupération d'espace mémoire, de sécurité, de gestion du cycle de vie des objets, mais sont complexes, peu compatibles avec les pare-feux, et difficilement interopérables entre eux. Ils restent donc souvent confinés à l'intérieur des entreprises. Nous avons détaillé par la suite les architectures orientées services (SOA) qui prennent leur origine dans l'informatique distribuée (début 2000) et dans l'avènement du Web.

Les architectures SOA constituent la tendance actuelle de recherche, grâce à leurs succès dans la réalisation des services Web. Ces derniers ont prouvé également leur efficacité dans le domaine de l'informatique ubiquitaire notamment par leur aspect sémantique qui permet une représentation standard des connaissances. Des recherches actuelles dans le cadre des architectures à agents orientés services, visent à intégrer l'intelligence dans les services Web en s'appuyant sur le paradigme agent. Ceci permet de doter les services Web d'une capacité de perception de leur environnement.

Après avoir discuter la notion d'informatique ubiquitaire et de services, nous allons voir en détail la notion du contexte et de sensibilité au contexte, qui feront l'objet du prochain chapitre.

Chapitre II

Contexte et sensibilité au contexte

II.1 Introduction

Le domaine des technologies de l'information et de la communication est en constante évolution. De ce fait, nous assistons aujourd'hui à une multiplication des terminaux mobiles dans notre vie quotidienne (PDA, téléphones cellulaires...). De plus, ces dispositifs peuvent être connectés facilement aux réseaux sans fils [46]. Il est ainsi devenu tout à fait possible pour un utilisateur équipé de dispositifs divers (ordinateur de bureau, portable ou de poche, téléphone portable, etc.), de se connecter à Internet et d'accéder aux systèmes disponibles, depuis des lieux divers (son bureau, chez lui, l'aéroport, etc.) et à des moments différents. Ce nouveau type d'utilisateur, est appelé utilisateur nomade car il peut se déplacer à tout moment.

En raison de cette évolution, il est tout naturel de prendre en compte le contexte de l'utilisateur afin de mieux le satisfaire. C'est la raison pour laquelle nous observons aujourd'hui une émergence de plus en plus forte des applications qui tiennent compte de la notion du contexte. Ces applications dites sensibles au contexte ont pour but d'adapter les informations ou services qu'elles fournissent à l'utilisateur aux circonstances d'utilisation.

Les recherches menées sur le contexte sont très actives. Ce concept est même devenu un élément clé à prendre en compte lors de la conception des applications. Particulièrement dans le cadre de l'informatique ubiquitaire qui a renforcé son importance en initiant de nouvelles directions de recherche notamment sur les applications sensibles au contexte *(context-aware)* et l'intelligence ambiante.

Cependant, la notion du contexte reste vague, et il n'existe pas une définition unique et standard à ce concept. La plupart des définitions dépendent du domaine d'application, et se limitent uniquement à quelques éléments du contexte tels que : la localisation de l'utilisateur et les caractéristiques de son dispositif. De plus, la formalisation du contexte est très liée aux objectifs visés par l'application en question.

Dans ce chapitre, nous rappelons d'abord quelques définitions et classifications utilisées pour la notion du contexte, ainsi les différentes approches d'acquisition et de modélisation du contexte. Nous abordons, par la suite, la notion de sensibilité au contexte et les mécanismes d'adaptation du contexte aux applications. Enfin, nous présentons quelques architectures sensibles au contexte et nous terminons par une synthèse de ce chapitre.

II.2 La notion du contexte

II.2.1 Définition du contexte

La notion du contexte est pluridisciplinaire, et désigne un ensemble de circonstances qui encadre un événement ou un objet. Les auteurs dans [4] ont établi un corpus de plus de 150 définitions pour cette notion, allant de la psychologie à l'informatique. Il est donc difficile de trouver une définition satisfaisant toutes les disciplines

Dans la littérature, il existe plusieurs définitions du contexte appliquées à l'informatique. Ces définitions sont données de deux façons : par énumération et par description. Les définitions par énumération sont souvent utilisées dans les premières applications. Elles se limitent à quelques éléments du contexte tels que la localisation, l'environnement, l'identité, la date et heure, etc. Ainsi, Schilit et Theimer définissent le contexte comme la localisation et l'identité

des personnes et des objets environnants et les changements sur ces objets [64]. Pour Schilit, étudier le contexte c'est répondre aux questions : « Où es-tu ? », « Avec qui es-tu ? », « De quelles ressources disposes-tu à proximité ? ».

Actuellement, c'est les définitions par description qui sont utilisées. Brown définit le contexte comme «les éléments de l'environnement dont l'ordinateur à connaissance» [9]. La définition la plus adoptée par les chercheurs est celle de Dey [17] : « le contexte couvre toutes les informations pouvant être utilisées pour caractériser la situation d'une entité. Une entité est une personne, un lieu, ou un objet qui peut être pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application eux-mêmes ».

La plupart des chercheurs [60] [8] [75] ont fait le consensus sur les constats suivants : (i) il n'y a pas de contexte sans contexte : le contexte n'existe pas en tant que tel, il est défini en fonction des objectifs à atteindre qui représentent sa finalité (utilité). (ii) Le contexte est un espace d'information qui sert l'interprétation : la capture du contexte n'est pas une fin en soi mais elle doit servir certains objectifs. L'interprétation du contexte dépend alors de sa finalité. (iii) Le contexte est un espace d'informations partagé par plusieurs acteurs : ces acteurs peuvent concerner l'environnement, l'utilisateur, le service,…etc. (iv) Le contexte est un espace d'information évolutif : le contexte n'est pas figé, mais il se construit au cours du temps. D'après ces quatre axes de définition du contexte, nous déduisons qu'il est important de bien cerner et spécifier la finalité du contexte et définir les informations nécessaires et suffisantes pour servir cette finalité.

En informatique diffuse, le contexte physique est d'abord acquis à partir de données de capteurs disséminés dans l'environnement. Ces données sont, par la suite, élaborées en un format plus adapté à leur utilisation. Par rapport à la donnée brute produite par le capteur, la donnée traitée est moins précise mais plus adaptée [20]. Le contexte en informatique diffuse est très souvent formé autour des besoins d'une application particulière et d'un domaine spécifique.

II.2.2 Classification du contexte

Afin de mieux comprendre le contexte, il est important de passer par une catégorisation de ses variables. Plusieurs tentatives de classification ont été proposées dans la littérature, dont voici les plus importantes :

- Dey et Abowd [17] séparent les informations contextuelles en deux classes :

- **Une classe principale (primaire):** qui contient des informations sur le temps, la localisation, l'activité et l'identité d'un utilisateur. En fait toute variable contextuelle qui répond à l'une des question suivante : "Quand ? Où ? Quoi ? Qui ?". Le système utilisera ces informations catégorisées pour répondre au "Pourquoi ?" de l'occurrence d'une situation. Ces informations sont les plus utilisées et agissent comme indexe pour les autres types d'informations.

- **Une classe secondaire** : qui contient toutes les autres informations. Ces dernières sont considérées comme des attributs des catégories primaires. Par exemple : de l'identité, on peut facilement retrouver le numéros de téléphone, l'adresse, l'adresses email, la date de naissance, la liste des amis et d'autres informations personnelles. En connaissant la

localisation d'une entité, on peut déterminer les entités voisines ou les activités qui se produisent aux alentours.

- Dans les articles [33] [31], les auteurs donnent une classification du contexte selon sa finalité (ou utilité) :

- **Contexte relevant/non relevant :** les informations contextuelles n'ont pas toutes la même importance d'un domaine à l'autre. Il s'agit donc de choisir celles qui sont pertinentes (relevantes) par rapport à la finalité (objectif) désirée.

- **Contexte statique/dynamique :** le contexte est souvent dynamique, mais le taux de changement des informations diffère d'un type à un autre. L'information qui possède un taux de changement faible est supposée comme statique. Exemple : le nom d'une personne, le nom d'une ville, …etc. L'information statique est connue une fois pour toute, et n'a pas besoin d'un coût supplémentaire pour la retrouver. Contrairement à l'information dynamique qui change fréquemment.

- **Contexte direct/dérivé :** le premier niveau du contexte est appelé direct, car il contient des informations élémentaires (température, position,…etc.) qui nous renseignent directement sur un état. Le deuxième niveau du contexte est un niveau abstrait. Il contient des informations qui sont déduites à partir des informations élémentaires par des mécanismes de raisonnement. Ce contexte est alors appelé contexte dérivé. Le raisonnement sur le contexte aura évidemment un coût supplémentaire de traitement.

II.2.3 Caractéristiques des informations du contexte

Les informations contextuelles possèdent plusieurs caractéristiques. Nous citons quelques unes dans ce qui suit.

- **Hétérogénéité** : le contexte provient de différents capteurs disséminés dans l'environnement. Ces sources de contexte sont variées et différent en terme du type de l'information perçue (texte, image, son, vidéo,…etc.), ainsi qu'en terme de capacité et de qualité de perception. De plus, un même type d'information peut être fournis par plusieurs sources différentes. Par exemple, la localisation d'un individu peut être donnée par un GPS, un GSM, une caméra, un badge IR, etc. Ceci implique donc une large hétérogénéité du contexte que ce soit au niveau modélisation, ou bien au niveau gestion et qualité.

- **Variabilité (Dynamicité)** : le contexte change fréquemment pour diverses raisons. Tout d'abord, les utilisateurs sont très mobiles et leurs préférences changent constamment. Ensuite, l'environnement est très dynamique et les informations qu'il fournie change rapidement. Et enfin, les services offerts par les différents fournisseurs sont souvent modifiés. Il est donc évident que les informations du contexte soient très dynamiques et leur état devient rapidement obsolète s'il n'est pas mis à jour.

- **Imperfection** : les informations du contexte peuvent être ambiguës, incomplètes et parfois même incohérentes. Plusieurs causes peuvent être à l'origine de ces problèmes. D'abord, les environnements pervasives sont très dynamiques, ce qui signifie que la description de l'information peut devenir rapidement obsolète ou incohérente. D'autant plus que le contexte avant d'arriver au consommateur (l'utilisateur) final, il doit passer par plusieurs étapes de transformation afin de le présenter dans le format requis par le consommateur ;

ces facteurs peuvent provoquer un large délai entre la production et l'utilisation du contexte. Ensuite, les capteurs physiques (ou sondes) peuvent être à l'origine des erreurs de capture et d'un bruitage des informations de contexte. Enfin, les fournisseurs de contexte peuvent se déconnecter ou tomber en panne, ce qui signifie que le contexte devient inconnu pour les utilisateurs.

- **Complexité** : cette complexité est la conséquence directe des caractéristiques citées précédemment. De plus, les sources de contexte peuvent êtres distribuées et nécessitent l'acquisition, le traitement et le stockage du contexte. Le traitement à son tour nécessite plusieurs mécanismes de raisonnement sur le contexte selon les domaines d'application. Alors que le modèle du contexte doit supporter plusieurs formats de représentation de l'information contextuelle à des niveaux d'abstraction différents. Ces facteurs et bien d'autres rendent la gestion du contexte difficile et complexe. Cette complexité des informations de contexte nécessite des moyens de modélisation et d'interprétations spécifiques ainsi que des mécanismes d'adaptation qui permettent de prendre en compte leur nature.

II.2.4 Acquisition des informations du contexte

L'acquisition des informations de contexte constitue la première étape dans le processus de traitement du contexte. Concrètement, le contexte fourni à une application peut être acquis à partir des sources diverses. Prenons le cas, par exemple, d'un utilisateur mobile qui cherche le plus proche restaurant qui répond à ses préférences. L'utilisateur doit spécifier ses menus préférés et éventuellement d'autres préférences tel que le coût. Le système doit pouvoir répondre aux exigences de l'utilisateur en se basant sur les informations clairement spécifiées par l'utilisateur et sa position actuelle fournie par le système GPS, qui sera transformée en une adresse contenant le pays, la ville,…etc. Cet exemple nous illustre les différentes sources du contexte. D'abord, les préférences de l'utilisateur sont fournies explicitement par l'utilisateur au système. Ensuite, les coordonnés de l'utilisateur sont capturées par le dispositif de positionnement GPS. Enfin, ces coordonnés sont transformé en adresse. Ainsi, dans l'article [48], Mostéfaoui et al. distinguent trois types de contexte selon les méthodes utilisées pour collecter l'ensemble des informations de ce contexte :

- **Le contexte capturé** : il s'agit du contexte qui est acquis directement par les capteurs physiques disséminés dans l'environnement telles que les capteurs de température, de bruit, GPS, les caméras, …etc. Ce type de contexte peut être également acquis par des capteurs logiciels, par exemple les mesures des pertes ou de la gigue lors de la transmission de la vidéo sur des liaisons sans fil peuvent constituer le contexte d'un algorithme de régulation du débit. Aussi, la capture de la bande passante du réseau nécessite l'utilisation de systèmes spécifiques.

- **Le contexte explicitement fourni** : comme nous l'avons cité dans l'exemple précédent, l'utilisateur communique explicitement ses préférences au système (application).

- **Le contexte dérivé ou interprété** : nous avons déjà fait allusion à la notion du contexte dérivé dans la section II.2.2. Il s'agit en fait d'un contexte de haut niveau qui est déduit à partir des informations de contexte de bas niveau (contexte capturé). Dans l'exemple précédent, le pays et la ville où se trouve l'utilisateur représentent un contexte de haut niveau déduit à partir du contexte de bas niveau qui représente ses coordonnées collectées à partir d'un GPS.

II.2.5 Délivrance du contexte à l'application

À présent les informations du contexte sont acquises par les dispositifs chargés de cette tâche en l'occurrence les capteurs. Pour exploiter le contexte, une application peut soit accéder directement aux capteurs pour avoir l'information brute, la traiter et puis l'utiliser, ou bien s'appuyer sur un intergiciel qui décharge cette application de la fonction d'acquisition du contexte. Il existe deux méthodes permettant de délivrer les informations de contexte à l'application, la première est basée sur une approche où l'application est conduite par les capteurs et la deuxième est basée sur la séparation entre l'acquisition du contexte et l'application [18].

- **Approche conduite par les capteurs :** nous pouvons faire une analogie avec une application qui accède directement aux ressources matérielles d'une machine (disque dur, mémoire, etc.) sans passer par le système d'exploitation. L'application doit alors implémenter un code de bas niveau pour communiquer avec ces ressources. Cela oblige les développeurs de l'application à connaître les détails du matérielle. Ainsi, leur tâche devient fastidieuse. De plus, l'application devient très dépendante du matérielle, et par conséquent, elle ne sera pas interopérable. De même pour l'approche conduite par les capteurs, l'application est reliée directement aux pilotes des capteurs. Elle doit alors implémenter le code se rapportant aux capteurs en utilisant les protocoles dictés par ces derniers. Cette approche a l'inconvénient de mélanger les détails de bas niveau de l'acquisition du contexte avec la sémantique de l'application. De plus, elle rend l'application très dépendante des capteurs et incapable d'interagir avec d'autres capteurs pour une même information de contexte.

- **Approche de séparation entre le contexte et l'application :** dans cette approche, l'acquisitions du contexte est déléguée à un intergiciel qui se charge d'acquérir et stocker les informations de contexte indépendamment de toutes applications. Cette approche permet de cacher à l'application les détails se rapportant aux capteurs. L'application interagit avec cet intergiciel selon deux modes :

  - **le mode par interrogation** : qui consiste à envoyer une requête vers l'intergiciel pour récupérer la valeur du contexte. Ce mode est utile dans le cas où l'application utilise l'information de contexte une seule fois et suppose que l'application est proactive, c'est à dire qu'elle sait quand demander l'information du contexte.

  - **le mode par notification** : qui consiste à s'inscrire auprès de l'intergiciel pour être notifiée par ce dernier de la valeur du contexte. Ce mécanisme est approprié pour un besoin répétitif du contexte. Dans ce cas, l'application peut établir des conditions qui précisent quand elle doit être notifiée.

La séparation de l'acquisition des informations de contexte de leur utilisation présente un grand intérêt puisqu'elle permet la réutilisation des mécanismes d'acquisition par plusieurs applications, ainsi que l'utilisation de plusieurs types de capteurs par une même application.

II.2.6 Modélisation du contexte

Maintenant que l'application a reçu le contexte, elle doit le gérer, le manipuler et l'utiliser. Ceci passe inévitablement par la modélisation de ce contexte. Les caractéristiques variées des informations de contexte telles que l'hétérogénéité, la dynamicité et l'imperfection

(voir section II.2.3) nécessitent des modèles abstraits de description de contexte. Plusieurs travaux ont défini des modèles de contexte. Nous citons quelques uns dans ce qui suit.

- **Paires de clé-valeur :** c'est l'une des premières approches sur la modélisation du contexte qui est introduite par Schilit et al. [63]. L'information de contexte est modélisée par une paire clé-valeur sous forme *clé=valeur1 : valeur2 :... : valeurN*. La clé représente l'identifiant du contexte, et la valeur représente les informations associées à ce contexte. Par exemple la liste des noms des occupants d'une salle est la suivante : *OCCUPANTS = adams :schilit :theimer :weiser :welch.*

- **Modélisation orientée objet :** cette approche proposée par Henricksen et al. [33] se base sur un ensemble de concepts du paradigme orienté objet. Les informations de contexte sont regroupées en un ensemble d'entités. Chaque entité représente un objet conceptuel ou physique tel qu'une personne, un dispositif ou un réseau. Les propriétés des entités tel que le nom d'une personne sont représentées par des attributs. Les entités sont liées à leurs attributs à travers des associations.

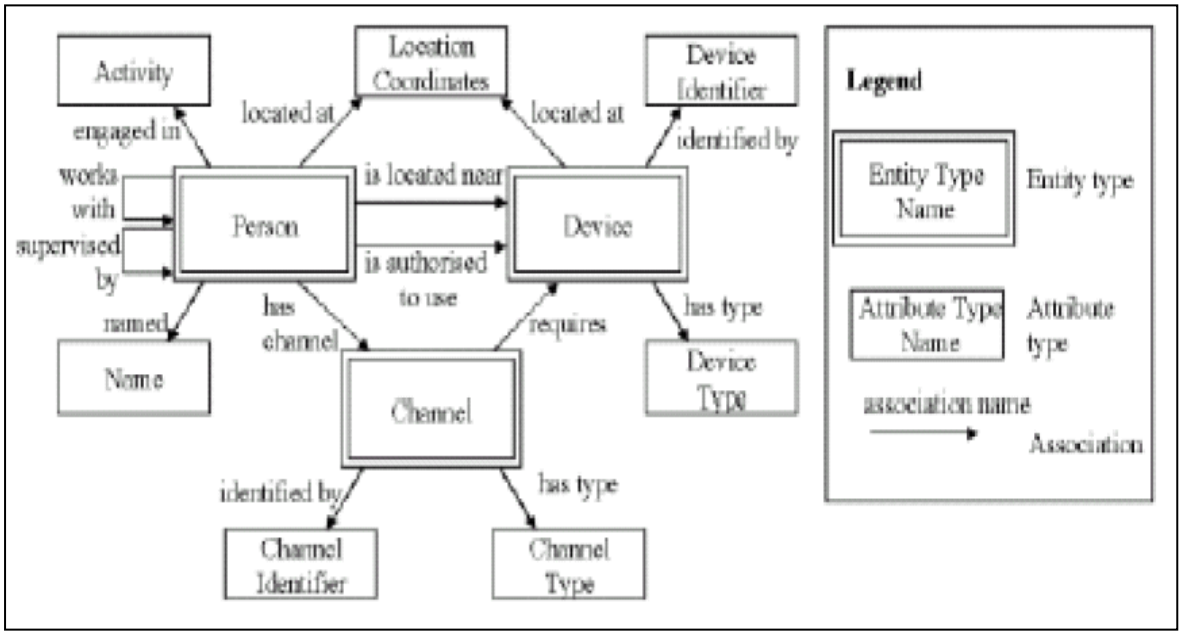


Fig. II.1 : Représentation entité-association-attribut

- **Modélisation par une ontologie :** les techniques de représentation de données proposées dans les langages à objets comme Java et C++ ne garantissent pas un niveau d'expressivité suffisant pour décrire les différentes classes du contexte. Pour ce faire, il est nécessaire de s'appuyer sur des techniques de représentation des connaissances avancées telles que les ontologies. Ces dernières offrent en effet un haut niveau d'expressivité pour construire un modèle du contexte disposant d'un vocabulaire riche et extensible. Dans une ontologie, les connaissances d'un certain domaine sont représentées formellement comme un ensemble d'objets ayant des relations entre eux. La modélisation du contexte par une ontologie a un réel intérêt ; elle permet le partage des informations de contexte dans un système distribué, et avec une sémantique bien définie, elle permet l'utilisation d'agents intelligents pour faire un raisonnement sur ce contexte. Une ontologie nécessite un langage de représentation basé sur un modèle sémantique. Ces dernières années, plusieurs langages d'ontologies ont été proposés tels que OWL et OWL-S. Ces langages sont utilisés pour la modélisation du contexte [56].

II.3 La sensibilité au contexte

II.3.1 Définition de la sensibilité au contexte

Dans la littérature, il existe beaucoup de synonymes au terme « sensible au contexte ». On parle de systèmes « adaptable au contexte », « réactifs », « dirigés par l'environnement » ou « qui prend en compte le contexte ». Ces appellations sont toutes proche de la notion de sensibilité au contexte et proviennent du terme anglais « *context-aware computing* » qui a été introduit pour la première fois par Schilit et Theimer en 1994 [64]. Ces auteurs ont défini ce terme comme étant un logiciel qui « s'adapte en fonction de sa localisation d'utilisation, de l'ensemble des personnes et des objets à proximité, ainsi que de la variation de ces objets à travers le temps ». Nous constatons que cette définition porte uniquement sur quelques variables tels que la localisation, l'identité des personnes et des objets voisins ainsi que les changements sur ces objets. Un peu plus tard, d'autres définitions ont vu le jour. Brown [10] dit qu'une application est sensible au contexte si elle peut effectuer des actions en fonctions du contexte utilisateur détecté par les capteurs. Dey et Abowd [17] proposent qu' « un système est sensible au contexte s'il utilise le contexte pour fournir des informations pertinentes ou des services utiles à l'utilisateur, l'utilité dépend de la tâche de l'utilisateur ». Dans [14], la sensibilité au contexte est définie comme étant la capacité du système à exploiter le contexte pour fournir à l'utilisateur des services pertinents.

L'adaptation de l'application aux informations du contexte est illustrée par la figure ci-dessous (Fig.II.2). Elle montre clairement que les éléments de contexte agissent sur le comportement de l'application. De même, il est possible pour l'application d'agir et de modifier le contexte. C'est ce qui est appelé boucle de contexte où la sortie contextuelle de l'application va modifier les valeurs des variables contextuelles.

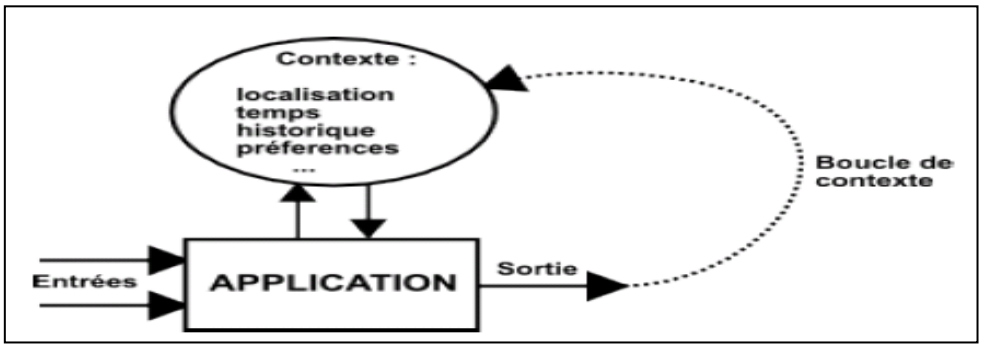


Fig. II.2 : Application tenant compte du contexte

II.3.2 Mécanismes d'adaptation

La modélisation des informations de contexte est primordiale pour leur gestion à savoir leur transport, leur échange et leur stockage. Mais vu la diversité des informations de contexte, cette modélisation est étroitement liée au domaine de leur utilisation. Il existe plusieurs mécanismes qui permettent aux applications d'être adaptables et réagir ainsi à des changements de contexte. Les mécanismes les plus connus [78] [2] sont les suivants:

- **Réflexivité :** La réflexivité représente la capacité d'un système à s'observer et agir sur lui même [7]. Un système réflexif doit être capable de manipuler des données représentant des informations sur son état durant son exécution. Ce système est composé de deux

niveaux : le niveau de base qui comporte les données et le code fonctionnel du système (le « quoi » du système) et le méta niveau qui comporte les données et le code non fonctionnel du système (le « comment » du système). Le méta niveau permet au système de s'observer et d'agir sur lui-même pour s'adapter.

- **Programmation par aspects :** Le paradigme de programmation par aspect a été mis en place afin d'élargir la possibilité de réutilisation du code, puisqu'il permet de développer des applications en séparant leur code métier de leur code technique (non fonctionnel) selon le principe de séparation des préoccupations [35]. Ceci rend le code indépendant de l'infrastructure utilisée. Cette séparation permet de structurer l'application en modules indépendants représentant différents aspects : un noyau qui représente le coeur fonctionnel de l'application (le « quoi » de l'application); plusieurs aspects qui représentent des propriétés transversales au noyau (le « comment » de l'application). La construction d'une application en utilisant ces différents modules nécessite une étape d'intégration du noyau avec les différents aspects. Cette intégration se traduit par l'établissement d'une jonction qui se situe au niveau d'un ensemble de points du flot d'exécution du noyau, appelés points de jonction. Ce type de programmation peut aussi être utilisé comme technique d'adaptation. Dans ce cas, si nous voulons intégrer un procédé d'adaptation dans une application, ce procédé doit être considéré comme du code technique.

- **L'utilisation de moteurs d'inférence :** Lorsque les informations de contexte sont représentées à l'aide d'une ontologie, l'intelligence d'un système sensible au contexte peut être implémentée à l'aide d'un moteur d'inférence qui raisonnera sur les informations de contexte. Un moteur d'inférences est implémenté par un ensemble de faits et de règles appliquées sur les informations de contexte collectées à partir des capteurs. Parmi les implémentations existantes de moteurs d'inférences, nous pouvons citer Flora2 [12] et Jena 2 [83].

II.3.3 Les services sensibles au contexte

Nous définissons un service sensible au contexte comme étant une entité logicielle capable d'acquérir et d'interpréter des connaissances contextuelles afin d'améliorer sa perception de l'environnement et/ou d'adapter son comportement en conséquence. Dans ce cas, la sensibilité au contexte est perçue comme un comportement applicatif intelligent permettant aux services de fournir des réponses en fonction du contexte d'utilisation.

Dans un environnement ubiquitaire, ces services sont appelés services ubiquitaires sensibles au contexte. Ces derniers possèdent la faculté de coopérer entre eux afin de réaliser une tâche bien déterminée, formant ainsi un système sensible au contexte capable d'accéder aux informations issues du contexte, les interpréter et les utiliser afin de fournir des services adaptés aux besoins des utilisateurs.

Les services ubiquitaires sensibles au contexte sont fournis par les différents dispositifs formant l'environnement ubiquitaire. Cela fait apparaître des verrous technologiques liés à l'interopérabilité de technologies matérielles/logicielles hétérogènes et aux contraintes induites par l'environnement ubiquitaire, en particulier sa nature dynamique. Comme nous l'avons vu dans le chapitre précédent, les services Web constituent une réponse à ces problèmes. Un service sensible au contexte est alors perçu comme un service Web muni d'un comportement intelligent.

II.3.4 Les intergiciels sensibles au contexte

Les intergiciels sensibles représentent des couches intermédiaires de gestion du contexte. Ils visent à offrir des cadres de développement standards et proposer des interfaces uniformes et réutilisables pour interagir avec le contexte. Ces intergiciels se basent essentiellement sur la représentation du contexte et l'adaptation des applications au contexte.

Nous avons vu dans la section II.2.5 deux approches pour délivrer le contexte à l'application : l'approche conduite par les capteurs et les intergiciels sensibles au contexte qui séparent entre l'application et le contexte. L'approche basée sur les intergiciels constitue la tendance actuelle des développeurs vu les avantages qu'elle offre en terme d'interopérabilité, de réutilisation et de rapidité de développement. Elle permet même de masquer l'hétérogénéité des environnements ubiquitaires en fournissant un haut niveau d'abstraction du contexte à l'application sous forme de cadres contenant un ensemble de procédures standards à suivre [79]. Un cadre ou canevas logiciel *(Framework)* est un ensemble de classes abstraites collaborant entre elles pour faciliter la création de tout ou d'une partie d'un système logiciel. Un cadre fournit un guide architectural en partitionnant le domaine visé en classes abstraites et en définissant les responsabilités de chacune ainsi que les collaborations entre classes.

Ces classes présentent un ensemble de mécanismes réutilisables pour la collecte des informations de contexte, leur interprétation, leur analyse, leur représentation et leur stockage, et peuvent même offrir des méthodes d'adaptations basées sur différentes techniques tels que la réflexivité ou l'utilisation de moteurs d'inférence. Ainsi, l'application placée au-dessus de ces intergiciels n'a pas à redévelopper tous ces mécanismes. La figure suivante illustre l'architecture de référence d'un intergiciel sensible au contexte:

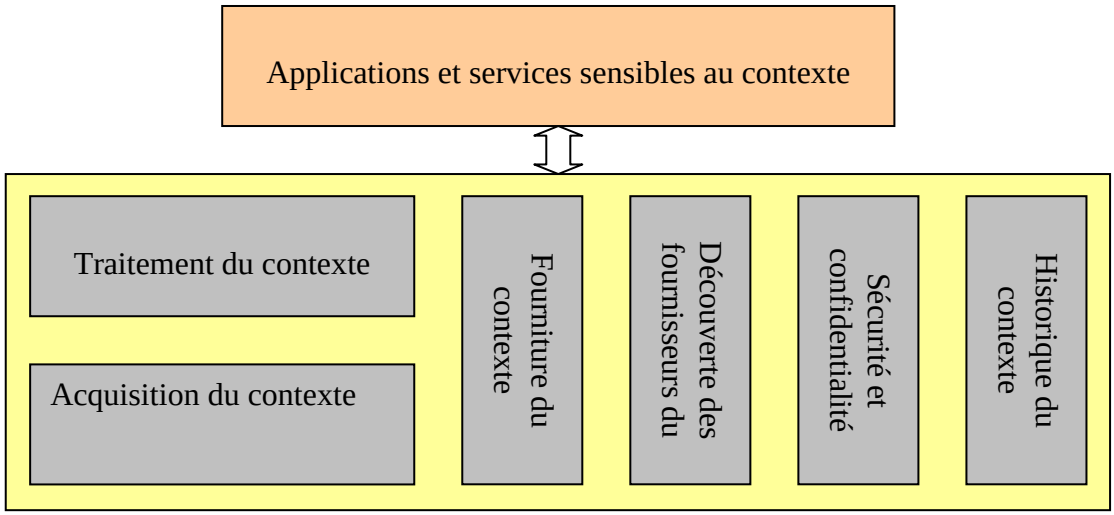


Fig. II.3 : Architecture de référence d'un intergiciel sensible au contexte

L'architecture de référence d'un intergiciel sensible au contexte dans un environnement ubiquitaire est composée d'entités logicielles offrant six fonctions principales (Figure II.3) : (i) acquisition (collecte ou capture) de données contextuelles, (ii) traitement de ces données (iii) fourniture des informations contextuelles aux applications et aux utilisateurs, (iv) découverte des fournisseurs de données contextuelles disponibles (v) contrôle d'accès aux fournisseurs de contexte et respect de la vie privé des utilisateurs (vi) gestion de l'historique du contexte.

Plusieurs intergiciels ont été développé en se basant sur cette architecture de référence. Ils sont classés en trois catégories [15] :

- **Les intergiciels d'adaptation au contexte** : ces intergiciels s'appuient sur des plates formes distribuées telles que CORBA ou CCM, qui fournissent des briques logicielles (objets ou composants) disposant de fonctions d'auto configuration et d'adaptation au contexte. Parmi les projets les plus marquants, on peut citer : *Camido, Sentient, Carisma*, etc.

- **Les intergiciels de gestion du contexte** : fournissent aux applications une couche d'abstraction du contexte offrant des fonctions d'acquisition et de traitement d'informations contextuelles. Dans cette catégorie d'intergiciels, on peut citer *le Context Toolikt* [72].

- **Les intergiciels de services sensibles au contexte** : concernent la gestion d'espaces intelligents (on parle aussi d'intelligence ambiante) et fournissent des services adaptés au contexte. Dans cette catégorie, on peut citer les projets CoBrA [13], Gaia [61], Socam [31]).

II.4 Synthèse

Dans ce chapitre, nous avons illustré la notion du contexte et de sensibilité au contexte. Il existe plusieurs définitions et classification du contexte selon le domaine d'application et les objectifs visés. La définition donnée par Dey [17] pour le contexte nous semble la plus appropriée. En effet, cette définition englobe toute information qui peut être pertinente pour l'interaction entre l'utilisateur et l'application. À la lumière de cette définition, nous construisons quatre classes de contexte à savoir : le contexte utilisateur *(user-context),* le contexte service *(service-context),* le contexte environnement *(env-context),* et le contexte ressource *(resource-context)* :

Le contexte utilisateur *(user-context) :* toute information relative à l'utilisateur telle que : ces préférences (centres d'intérêts), les informations indiquant son état physique ou émotionnel (la température corporelle, le rythme cardiaque, etc.), sa localisation, son activité courante, etc.

Le contexte service *(service-context) :* toute information non fonctionnelles du service tel que son coût, sa durée d'exécution, sa disponibilité, sa fiabilité, et sa réputation (en général, c'est la renommé du fournisseur du service). En faite, c'est les informations considérées par le modèle de qualité de service *(QoS : Quality of Service)*

Le contexte environnement *(env-context) :* contient des informations sur l'environnement de l'utilisateur (où est l'utilisateur ?) telles que le climat, la température, la luminosité, le bruit, la date, et quelques données sur la situation entourant l'utilisateur (les personnes ou objets à proximité), etc.

Le contexte ressource *(resource-context)* : contient toutes les informations sur les équipements (ressources) dont dispose l'utilisateur tels que : le terminal utilisé, le réseau de communication, les appareilles qui assistent l'utilisateur (notamment une personne

physiquement dépendante) dans sa vie quotidienne. Ainsi les propriétés de ces équipements : dispositif d'affichage, puissance de calcule, capacité de stockage, énergie, débits réseau, etc.

Certaines valeurs du contexte sont quantifiables (mesurables), d'autres doivent être paramétrées et numérisées. La méthode la plus communément utilisée est les fonctions d'utilités. Le demandeur doit spécifier le poids de chaque information qualitative. Le système va alors pouvoir attribuer une certaine mesure à cette information pour refléter le contexte et faciliter sa comparaison aux autres.

Quant à la sensibilité au contexte, les auteurs dans [14] la définissent comme étant la capacité du système à exploiter le contexte pour fournir à l'utilisateur des services pertinents. Plus concrètement, nous considérons que ce système exploite les informations des quatre classes précédentes du contexte pour fournir un service qui répond au mieux à la situation de l'utilisateur. Nous considérons aussi que ce contexte est fourni par des intergiciels sensibles au contexte. Comme nous l'avons montré, la séparation du contexte de l'application est la tendance actuelle. Elle permet de rendre l'application indépendante du contexte. Cette approche est adoptée par les la plupart des intergiciels sensibles au contexte.

Un autre aspect important concerne la modélisation du contexte. Parmi les approches adoptées pour cette modélisation est celle basée sur l'ontologie. Cette dernière fournie une représentation standard des connaissances, et interprétable par la machine. Les langages d'ontologie tels que OWL et OWL-S (voir chapitre I) peuvent être exploités dans ce sens.

Après avoir étudier la notion de service et de contexte, il reste maintenant à adaptés ces services aux contexte de l'utilisateur. Car la satisfaction de ce dernier constitue l'objectif du système. L'approche de composition de services sensibles au contexte vise cet objectif et constitue une jonction entre le contexte de l'utilisateur d'une part, et les services de l'autre part. Cette approche fera alors l'objet du prochain chapitre.

Chapitre III

Vers la composition de services

III.1 Introduction

Á l'inverse de l'informatique traditionnelle qui est orientée traitement, l'informatique ubiquitaire s'intéresse à l'adaptation des services aux besoins de l'utilisateur. On parle ainsi d'une informatique orientée services *(SOC : Service Oriented Computing).* Dans l'approche SOC, un service est vu comme une brique fondamentale dans le processus de développement d'applications. Il est indépendant de toutes plateformes logicielles ou matérielles et il peut être invoqué par des clients ou par d'autres services. L'un des défis majeurs des environnements ubiquitaires est de permettre à l'utilisateur de réaliser une tâche par la composition rapide des services disponibles dans l'environnement. En effet, dans de tels environnements, plusieurs équipements offrent différents services pour les clients qui sont soit des utilisateurs ou bien d'autres applications logicielles. Si ces services disponibles ne peuvent pas satisfaire individuellement la fonctionnalité désirée par l'utilisateur, il est alors possible de les combiner ensemble afin de satisfaire cette demande. Ainsi, la recherche de services appropriés pour atteindre un objectif donné devienne une tâche très importante. En particulier, le problème de combiner plusieurs services pour satisfaire une seule tâche, connu sous le nom du problème de composition de services, a reçu récemment beaucoup d'attention.

Nous distinguons deux niveaux d'abstraction des services proposés dans SOC. Les services de base qui fournissent des fonctionnalités élémentaires, et les services composés qui agrégent un ensemble de fonctionnalités dans des applications de haut niveau proches des besoins de l'utilisateur. La composition de services permet de combler l'écart abstrait entre les besoins de l'utilisateur et les services de base. Les développeurs peuvent résoudre des problèmes complexes en combinant les services de base disponibles et les ordonner pour mieux satisfaire les exigences de leurs problèmes. La composition de service accélère le développement d'applications, la réutilisation de services, et la réalisation de services plus complexes [44].

La composition automatique des services nécessite des descriptions profondes, claires, et compréhensibles de ces services. Dans le domaine des services Web, ces descriptions se basent sur les langages du Web sémantique tels que OWL-S. Étant donnée une spécification de haut niveau des objectifs d'une tâche particulière, la composition de service implique la capacité de sélectionner, de composer et de faire interopérer des services existants. Ainsi, en se basant sur des descriptions appropriées, les techniques de planification de IA peuvent êtres employées pour automatiser la composition des services déjà décrits. Cependant, le problème de composition de service Web diffère des problèmes de planifications classiques dans lesquels les tâches sont exécutées de manière prévisible et répétitive dans un environnement statique. Les services Web s'exécutent dans un environnement versatile où le nombre de services disponibles évolue très rapidement. L'information sur le monde réel est incomplète et change continuellement. De plus, la forte compétition engendrée par la multitude de fournisseurs de services oblige les entreprises à adapter leurs services pour mieux répondre aux besoins des clients et ce à moindre coût. En conséquence, les business process qui décrivent des services composés devront intégrer ces contraintes en exposant des possibilités réelles d'adaptabilité à leur environnement.

Ces contraintes constituent en fait les informations du contexte qui sont un élément important à prendre en compte lors du processus de composition des services. Elles augmentent l'efficacité de la composition proposée qui sera adaptée au contexte de l'utilisateur, et par conséquent, elle sera largement acceptée par ce dernier. L'utilisation du contexte, que ce soit dans le domaine de l'informatique ubiquitaire ou bien dans le domaine

des services Web, répond presque aux mêmes objectifs à savoir : l'aide à la personnalisation et à l'adaptation aux préférences de l'utilisateur, et le filtrage des services inappropriés lors du processus de composition afin d'améliorer son utilisation et son adoption. Un modèle de composition de services qui peut intégrer, tirer profit de l'utilisation du contexte pour délivrer des services composants pour le service composé, demeure un problème très avancé de recherche.

Dans ce chapitre, nous donnons quelques définitions et concepts liés à la composition de services et quelques exemples de son utilisation. Nous présentons également l'utilité de la composition de services dans des domaines très variés. Nous donnons par la suite un aperçu des récentes recherches sur la composition automatique de service Web avec le workflow et la planification à base de l'IA. Nous discutons à la fin l'influence du contexte sur la composition de services, avant de terminer par une synthèse de ce chapitre.

III.2 Concepts et définitions

III.2.1 Composition de services

- Un service Web est dit *composé* ou *composite* lorsque son exécution nécessite d'invoquer plusieurs autres services Web afin de faire appel à leurs fonctionnalités.

- Un processus est un enchaînement de tâches réalisées par des services en vu d'atteindre un objectif donné. Dans le domaine du Web, on parle du processus Web (*Web Process*), et dans le domaine industriel, on parle de processus métier (*Business Process*).

- Composition de services (*Services Composition*) : Techniques permettant d'assembler des services pour réaliser des processus par des primitives de contrôles (boucles, tests, traitement d'exception, etc.) et d'échanges (envoi et réception de messages). La composition de services Web spécifie quels services ont besoin d'être invoqués, dans quel ordre, quelles sont les données à échanger, et comment traiter des situations d'échecs. L'objectif de la composition de service est de créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services existants en vue d'apporter une valeur ajoutée. La composition des services Web peut se faire de deux manières: orchestration et chorégraphie [54].

III.2.2 Orchestration

L'orchestration décrit un ensemble d'actions dans lesquelles un service donné peut ou doit s'engager. Un coordinateur prend le contrôle de tous les services Web impliqués et coordonne l'exécution des différentes opérations des services Web qui participent dans le processus. Les services Web n'ont pas de connaissance d´être partie dans une composition. Seulement le coordinateur de l'orchestration a besoin de cette connaissance. La figure suivante montre le Workflow dans l'orchestration des services Web.

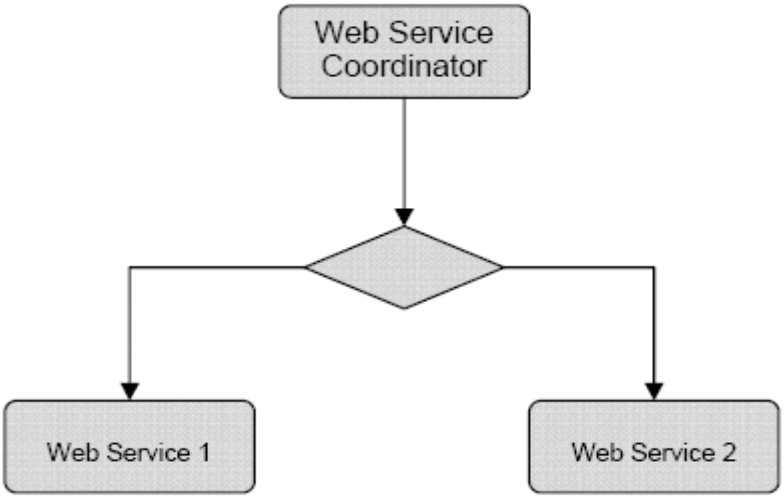


Fig. III.1 : Orchestration de Services Web

III.2.3 Chorégraphie

Contrairement à l'orchestration, la chorégraphie n'a pas de coordinateur central. Elle s'intéresse aux interactions entre les services. La chorégraphie est un effort de collaboration dans lequel chaque participant du processus décrit l'itération qui l'appartient. Chaque service Web qui participe à la chorégraphie connaît exactement quand ses opérations doivent être exécutées et avec qui l'interaction doit avoir lieu. La collaboration dans la chorégraphie des services Web peut être représenté de la manière suivant:

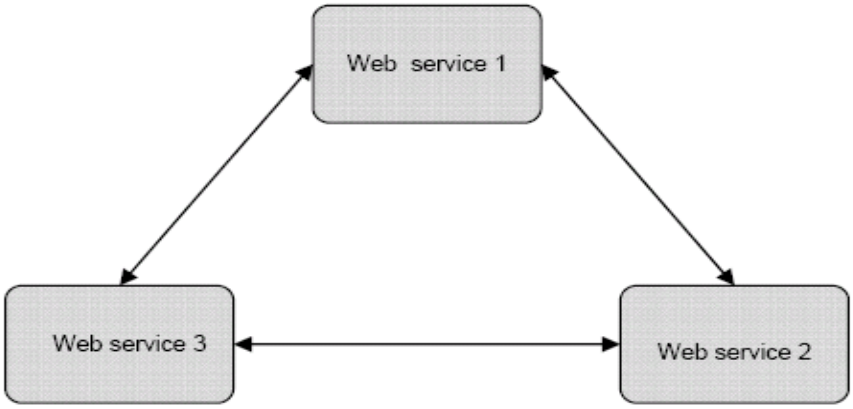


Fig. III.2 : Chorégraphie de Services Web

L'orchestration est un rapprochement plus flexible que la chorégraphie pour la composition des services Web. En effet, le coordinateur de tout le processus est connu, et les services Web peuvent être insérés sans soucis, parce qu'ils n'ont pas connaissance d'appartenir au processus. Toutefois, l'approche par chorégraphie est plus adaptée si les services sont vus comme des agents qui interagissent entre eux afin de mener le processus de composition.

III.2.4 Définition formelle d'un problème de composition de services

Formellement, un service Web, *W*, possède deux ensembles de paramètres [50] :

- W_{in}= {I_1, I_2,…} : paramètres d'entrés (une requête SOAP)
- W_{out} = {O_1, O_2, …} : paramètres de sorties (une réponse SOAP).

Quand *W* est invoqué avec tous les paramètres d'entrés, W_{in}, il retourne les paramètres de sortie, W_{out}. Pour invoquer *W*, tous les paramètres d'entrée dans W_{in} doivent êtres fournis (W_{in} sont obligatoires). Soit une requête *R* avec des paramètres d'entrée R_{in} et des paramètres de sortie R_{out} . Le problème de composition consiste à trouver un service Web *W* qui peut satisfaire R de telle sorte que (1) $R_{in} \supseteq W_{in}$, et (2) $R_{out} \supseteq W_{out}$. Trouver un service Web qui peut satisfaire R tout seul se reporte au problème de découverte de service Web. Quand il est impossible pour un seul service Web de satisfaire entièrement la requête R, il faut alors composer plusieurs services Web, $\{W^1, W^2, \ldots, W^n\}$ d'une manière séquentielle ou parallèle tel que (1) pour tout $W^i \in \{W^1, W^2, \ldots, W^n\}$, $W^i{}_{in}$ sont fournis quand $W^i{}_{out}$ sont requis à une certaine étape de la composition et (2) $(R_{in} \bigcup W^1{}_{out} \bigcup \ldots \bigcup W^n{}_{out}) \supseteq R_{out}$. Ce problème est appelé problème de Composition de Service Web.

III.2.5 Synthèse des approches de *matching*

Le problème de composition de services nécessite l'intégration d'informations des sources hétérogènes. Puisque les services Web individuels sont crées séparément, leurs vocabulaires posent souvent des problèmes d'abréviations et de formats. De plus, deux termes avec deux orthographes différentes peuvent signifier un même sens sémantique, et ainsi sont interchangeables (ex. « prix » et «coût »). Inversement, deux termes avec une même orthographe peuvent avoir des sens différents (ex. « titre » peut signifier soit « titre d'un livre » ou « titre d'une fonction »). En réponse à ces défis, les chercheurs ont développé plusieurs systèmes de *matching* qui peuvent se classer en trois catégories :
Approche 1 : matching exact utilisant l'équivalence syntaxique ;
Approche 2 : matching approximatif utilisant des fonctions de distance ;
Approche 3 : matching sémantique utilisant les ontologies (ex. RDF et OWL).

*C*onsidérant x et y des données objets (ex. paramètres d'un service Web ; « champ d'enregistrement »). x et y sont représentés sous forme de deux vecteurs d'attributs : $x = (x_1, x_2, \ldots, x_k)$, et $y = (y_1, y_2, \ldots, y_k)$, où k est la dimension des deux vecteurs.

Approche 1 : les deux objets x et y sont considérés être match (semblables) si est seulement si $x = y$. Cependant, avec cette approche, deux objets avec des représentations légèrement différentes ne peuvent pas être matchées.

Approche 2 : si les deux objets x et y sont suffisamment similaires selon une certaine fonction de distance ($d(x, y)$ est au dessus d'un certain seuil), alors les deux objets sont considérés semblable. La fonction distance $d(x, y)$ sert à quantifier la « similarité » entre x et y. Elle possède les propriétés suivantes : (1) $d(x, y) \geq 0$, telle que l'égalité se vérifie si est seulement si $x = y$, (2) $d(x, y) = d(y, x)$: symétrie (3) $d(x, y) \leq d(x, z) + d(z, y)$: inégalité triangulaire. Les fonctions de distance les plus connues sont : TF-IDF, Jaccard, SoftTF-IDF, Jaro, et Levenstein distance [6].

Approche 3 : bien que l'approche 2 est plus flexible que l'approche 1, il n'est pas suffisant d'identifier que le « prix » et le « coût » sont interchangeables. Comme réponse, les chercheurs ont créé la vision du Web sémantique dans laquelle les données possèdent des structures et des ontologies qui décrivent leurs sémantiques. Basée sur le fondement de la sémantique Web, l'approche 3 peut adresser le problème de *matching* des ontologies pour trouver le *mapping* (correspondance) sémantique entre deux ontologies, spécifiées par des langages tels que OIL, DAML+OIL, OWL et RDF.

Notons que le choix de l'approche pour faire le *matching* n'entre pas dans le cadre du problème de la composition de services. On suppose que ces tâches de *matching* sont précalculées et présélectionnées.

III.3 Domaines d'application de la composition de services

Il y a plusieurs espaces différents d'application où la composition automatique des services est utile :

III.3.1 Les services Web

Il existe plusieurs tâches ordinaires réalisées par les utilisateurs du Web chaque jour. Quand l'objectif nécessite une interaction avec différentes parties, la tâche devient pénible. Par exemple opter pour un voyage peut nécessiter l'achat des biais d'avion, réserver une chambre d'hôtel, et louer un véhicule. Localiser tous ces services avec les spécificités requises et les coordonner ensemble nécessite une composition automatique.

III.3.2 Les applications B2B

La composition est très importante au niveau des applications B2B *(Business To Business)* où les associations en ligne peuvent êtres formées sans aucun contrat à priori. Une entreprise qui veut commander certains articles et régler les détails de leur envoie, peut accomplir cette tâche en combinant les services fournis par des fabricants et des compagnies de transport. Cette composition permet la formation dynamique des communautés de commerce.

III.3.3 Les applications de Grid

Le Grid offre un cadre de calcule pour résoudre des problèmes à grand échelle en sciences appliquées. Plusieurs tâches du Grid nécessitent la coordination et la combinaison de plusieurs services et ressources. Composer ces services en workflows de différentes complexités est requis par les différentes tâches. Par exemple, un scientifique exerçant en bioinformatique peut avoir besoin des données sur l'ADN, appliquer quelques testes spécifiques sur ces données, puis transformer les résultats en un certain format. Chacun de ces services peut être fourni par différentes sources et doivent êtres combinés ensemble pour satisfaire l'objectif.

III.3.4 L'informatique ubiquitaire

L'exposition des fonctionnalités des équipements comme des services Web offre une méthode uniforme pour la description de leurs capacités et nous permet de composer ces services ensemble. Dans les environnements d'aujourd'hui marqués par une présence intense des équipements divers, plusieurs tâches nécessitent la composition des services telles que ceux offerts par des imprimantes, des projecteurs, des stations de travail ou bien les services disponibles sur le Web.

III.3.5 Quelques caractéristiques

Tous ces exemples proviennent pratiquement des domaines différents, toutefois ils partagent quelques caractéristiques fondamentales :

- **Disposition distribuée** : les descriptions de services sont crées par des différentes sources qui ne partagent pas nécessairement des connaissances et des compréhension communes. Cela implique que les services qui seront utilisés pour créer la composition doivent êtres découverts depuis des sources distantes. Cette découverte doit prendre en considération des incompatibilités possibles entre les vocabulaires des descriptions de service Web.

- **Information incomplète** : le système de composition aura des informations incomplètes du monde réel. Quand la taille et la nature du Web sont considérées, il est pratiquement impossible de disposer de toute l'information nécessaire et de façon exacte.

- **Intercaler l'exécution et la composition** : à un moment donné, le système de composition doit exécuter les services qui fournissent les informations nécessaires au processus de composition. Cela provient du fait que les informations relevants du problème ne sont pas toutes connues à l'avance, et certaines sont fournies par l'exécution des services. Par conséquent, il est nécessaire d'intercaler l'exécution et la composition.

- **L'échelle du Web** : le système de composition de services Web doit pouvoir assurer le passage à l'échelle du Web où le nombre de services disponibles peut être de l'ordre de millions. Il n'est pas possible de manipuler ce nombre avec des approches simples.

III.3.6 Quelques scénarios illustratifs de la composition de services

III.3.6.1 Recherche d'un restaurant préféré

Considérant l'exemple suivant : supposant qu'il existe deux services Web disponibles dans UDDI :

```
<message name='findRestaurant_Request'>
<part name='zip' type='xs:string'>
<part name='foodPref' type='xs:string'>
</message>
<message
name='findRestaurant_Response'>
<part name='name' type='xs:string'>
<part name='phone' type='xs:string'>
<part name='addr' type='xs:string'>
</message>
```

(a) findRestaurant

```
<message name='findDirection_Request'>
<part name='fromAddr' type='xs:string'>
<part name='toAddr' type='xs:string'>
</message>
<message name='findDirection_Response'>
<part name='map' type='xs:string'>
<part                    name='direction'
type='xs:string'>
</message>
```

(b) findDirection

findRestaurant *:* c'est un service qui prend en entrée un code postal et le repas préféré. En sortie, il fournie le nom, le numéro de téléphone, et l'adresse du plus proche restaurant situé dans la zone du code postal et qui offre le repas préféré qui est déjà spécifié .

findDirection *:* c'est un service qui prend en entrée une adresses de départ et une adresse de destination. En sortie, il fournie la direction de conduite en montrant sur une carte image, le chemin pour atteindre l'adresse destination partant de l'adresse de départ.

Soit la requête suivante : une personne visite « Alger » pour un voyage d'affaire et elle s'arrête à l'hôtel « Sheraton » situé à l'adresse « Staouali, 16001, Alger.». Cette personne souhaite avoir un repas dans un restaurant traditionnel à proximité de l'hôtel avec une direction de conduite pour lui montrer le chemin. Appelant cette requête r. Notons qu'aucun des deux services Web ne peut satisfaire r tout seul. Cependant, *findRestaurant* peut trouver un restaurant traditionnel à proximité de l'hôtel, mais ne peut pas donner la direction de conduite. D'autre part, le service Web *findDirection* peut donner une direction d'un emplacement à un autre, mais ne peut pas localiser le restaurant. Donc, il faut combiner les deux services Web pour satisfaire conjointement la requête r comme suit : (1) invoquer *findRestaurant* (« 16001 », « Traditionnel ») pour avoir le restaurant le plus proche, soit « BarakaRestaurant Boulevard Am. 16001, Alger » et (2) invoquer le service Web *findDirection* « Staouali, 16001, Alger. », « BarakaRestaurant Boulevard Am. 16001, Alger » pour avoir la direction de conduite.

III.3.6.2 Service d'impression à partir d'un mobile

Soit l'architecture suivante qui fournie des services à des application déployées sur des équipements mobiles :

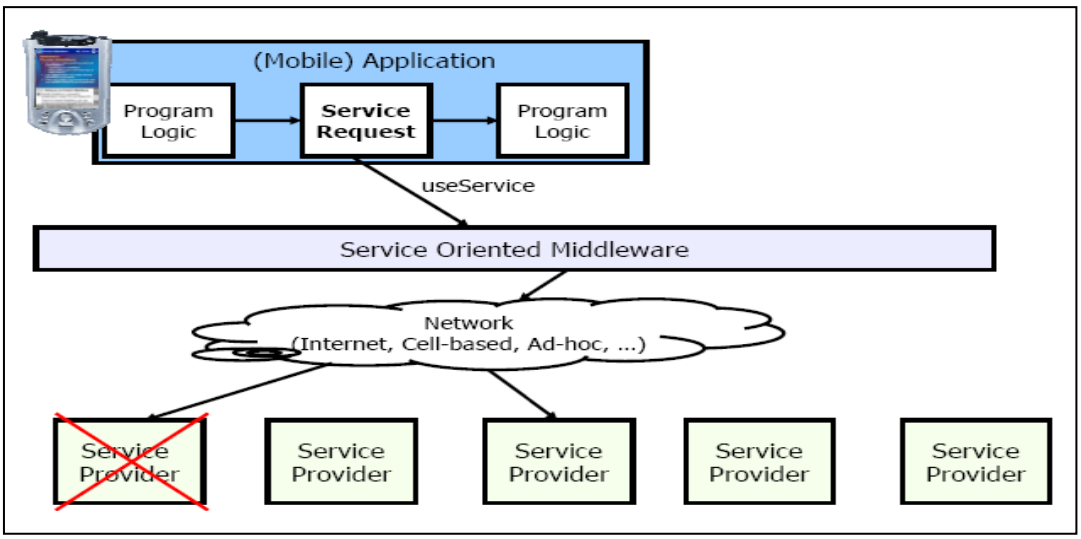


Fig. III.3 : Architecture d'accès aux services

Supposant qu'un utilisateur veut lancer l'impression d'un document MS Word (format .doc). Il se trouve qu'il n'existe aucun service qui répond à cette fonctionnalité comme le montre la figure suivante :

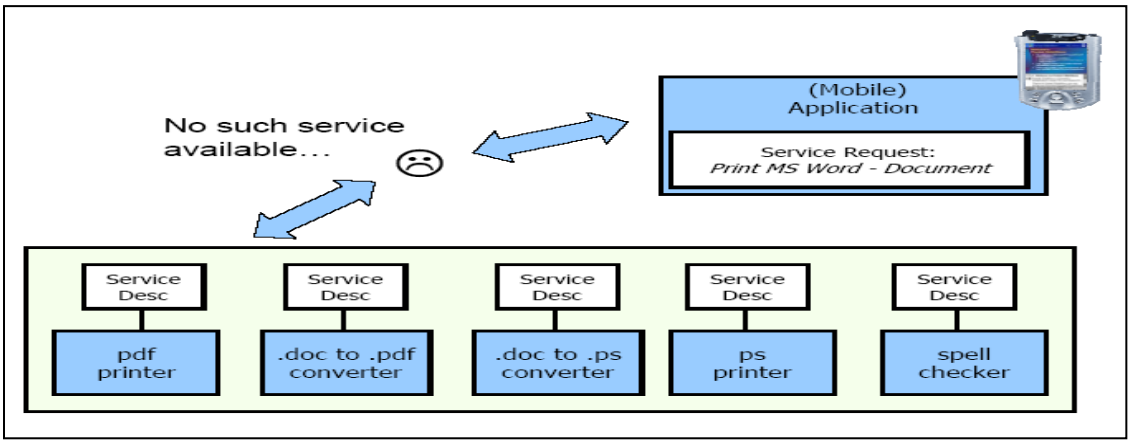


Fig. III.4 : Service demandé non disponible

En réalité, la demande de l'utilisateur peut être satisfaite si on combine plusieurs services. Il suffit par exemple d'invoquer d'abord le service qui convertie les documents .doc en .pdf. Ensuite, invoquer le service qui imprime les documents .pdf. La figure suivante illustre cette composition de ces deux services :

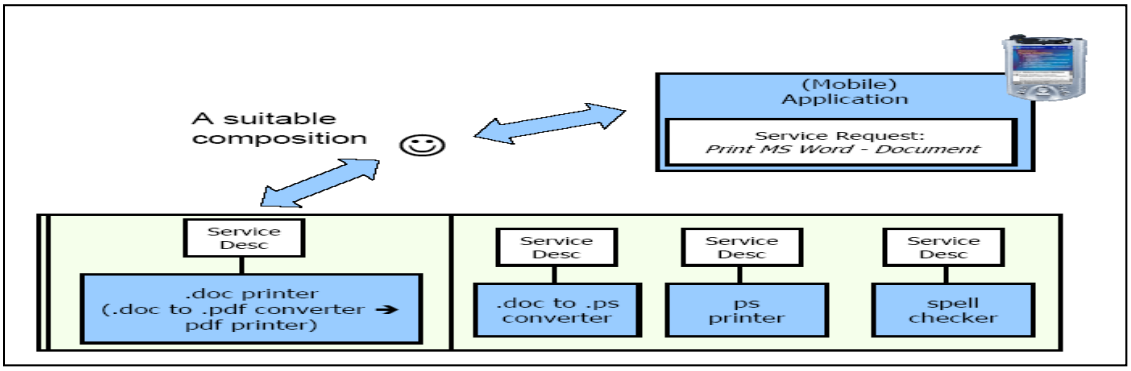


Fig. III.5 : Composition du service demandé

III.4 Architecture et méthodes de composition de services

III.4.1 Architecture générale d'un modèle de composition de services

Un cadre général du système de composition de service est illustré dans la figure ci dessus [59]:

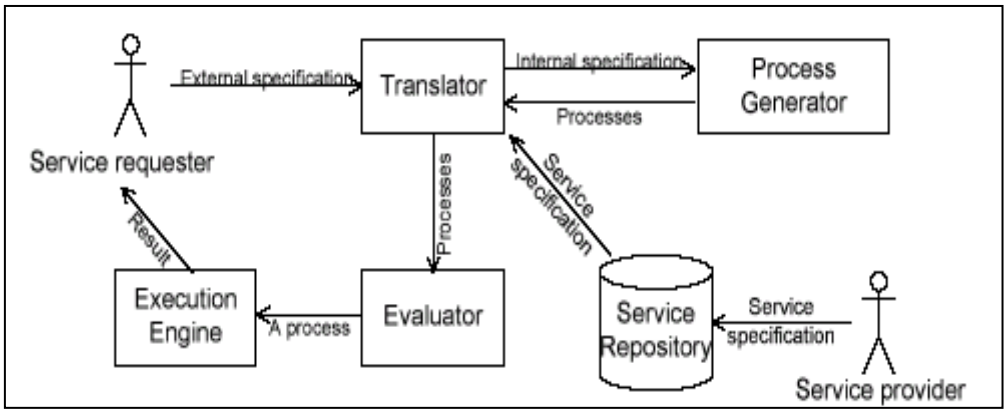


Fig. III.7 : Un cadre pour le système de composition de services

Le système de composition possède deux types de participants, le fournisseur de services et le demandeur de services. Les fournisseurs proposent des services pour l'utilisation. Les demandeurs de services consomment l'information ou les services offerts par les fournisseurs de services. Le système contient aussi les composants suivants : un translateur, un générateur de processus, un évaluateur, un moteur d'exécution et une base de services. Le translateur translate entre les langages externes utilisés par les participants et les langages internes utilisés par le générateur de processus. Pour chaque requête, le générateur de processus essaye de générer un plan qui compose les services disponibles dans la base des services pour satisfaire la requête. Si plusieurs plans sont trouvés, l'évaluateur évalue tous ces plans et propose le meilleur pour l'exécution. Le moteur d'exécution exécute le plan et retourne le résultat au fournisseur de service. Plus précisément, le processus de la composition automatique de service inclue les phases suivantes :

- **Présentation d'un service** : d'abord, les fournisseurs de service annoncent leurs services dans une place publique tel que UDDI. Les attributs essentiels pour décrire un service Web incluent la signature, les états et les valeurs non fonctionnelles. La signature est représentée par les entrées du service, ses sorties et ses exceptions. Elle fournie des informations sur la transformation des données durant l'exécution du service Web. Les états sont spécifiés par les prés conditions et les poste conditions. On les modélise comme la transformation d'un ensemble d'états à un autre dans le monde réel. Les valeurs non fonctionnelles sont les attributs qui sont utilisés pour évaluer les services, tels que le coût, la qualité du service et les aspects de sécurité.

- **Translation des langages** : la plupart des systèmes de composition de services distinguent entre les langages internes et externes de spécification de service. Les langages externes sont utilisés par les utilisateurs pour accroître leur accessibilité dans le sens où ces derniers peuvent exprimer leurs besoins d'une manière relativement simple. Ces langages sont souvent différents des langages internes qui sont utilisés par le générateur de processus de composition qui nécessite des langages formels et plus précis tels que les langages de programmation logique. Jusqu'à présent, les utilisateurs utilisent souvent les langages standard de services Web, tels que WSDL et OWL - S. Ainsi, les composants de translation entre les langages standard de services Web et les langages internes doivent êtres développés.

- **Génération du modèle de composition des processus** : le demandeur de service peut exprimer ces exigences dans un langage de spécification de service. Le générateur de processus essaye alors de répondre aux exigences en composants les services de base annoncés par les fournisseurs de services. Le générateur de processus prend souvent les fonctionnalités des services comme entrées, et un modèle de processus qui décrit le service composé. Le modèle de processus contient un ensemble de services sélectionnés et le flot de contrôle et de données entres ces services.

- **Evaluation du service composé** : il arrive souvent que plusieurs services possèdent des fonctionnalités similaires. Donc il est possible que le planificateur génère plusieurs services composites qui satisfassent les exigences de l'utilisateur. Dans ce cas, les services composés sont évalués par leurs utilités globales en utilisant les informations fournies par les attributs non fonctionnels. La méthode la plus communément utilisée est les fonctions d'utilités. Le demandeur doit spécifier le poids de chaque attribut non fonctionnel et le meilleur service composé est celui qui se classe le premier.

- **Exécution du service composé** : après avoir sélectionner un seul processus composé, le service composé est prés à être exécuté. L'exécution du service Web composé peut être vue comme une séquence de messages transmis selon le modèle de processus. Le flot de données du service composé est définie par le transfère des données de sortie d'un service composant aux prochains services composant qu'il précède directement dans le service composé.

III.4.2 Méthodes de composition de services

III.4.2.1 Composition de services à base du workflow

Le consortium WfMC (*Workflow Management Coalition*) propose la définition suivante d'un workflow : « un workflow est un processus d'une organisation, gérable par un outil workflow. Il est établi dans le but principal d'automatiser l'exécution du processus, mais il peut aussi servir à le simuler et à l'analyser ». Toujours selon le WfMC : « un système workflow définit, gère et réalise des procédures en exécutant des programmes dont l'ordre d'exécution est prédéfini dans une représentation informatique de la logique de ces procédures - les workflows ».

Dans la technique de composition de service avec un workflow prédéfini, un utilisateur doit spécifier le workflow du service composé requis, incluant les nœuds et les flots de contrôle et de données entre ces nœuds. Les nœuds sont vus comme des services abstraits qui contiennent des services concrets (des activités). Ces derniers sont alors sélectionnés et reliés pendant l'exécution selon plusieurs méthodes. Cette approche est largement adoptée par les membres de la communauté des systèmes d'informations [59].

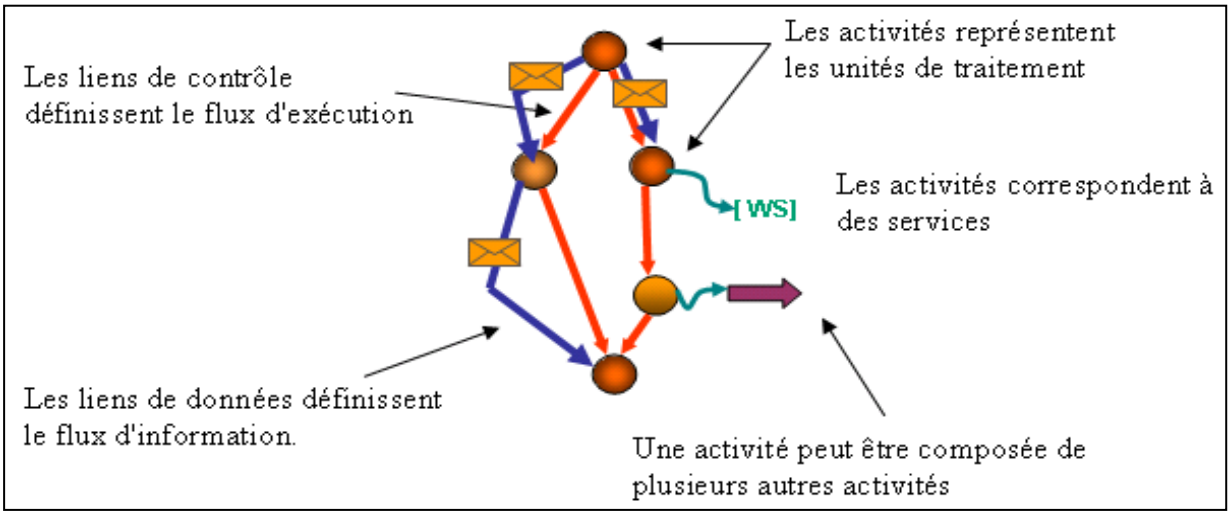


Fig. III.9 : Modélisation d'un processus par un workflow

Dans les méthodes de composition basées sur le workflow, on distingue la génération statique et dynamique de workflow. Le demandeur construit un modèle de processus abstrait avant que la planification de composition commence. Le modèle abstrait de processus inclue un ensemble de tâches et leurs dépendances de données. Chaque tâche correspond à une partie de la requête qui est utilisée pour chercher le service concret pour la réaliser. Dans ce cas, uniquement la sélection et l'enchaînement des services concrets qui sont faits automatiquement par le programme.

Plusieurs standards de spécification de flot de services Web ont été développées dans l'industrie telles que : *IBM's Web Service Flow Language (WSFL), BEA Systems' Web Services Choreography Interface (WSCI),* et *Business Process Execution Language for Web Services (BPEL4WS, ou simplement BPEL).* Les chercheurs ont développé BPEL en combinant WSFL et WSCI avec la spécification Microsoft's XLANG.

Les systèmes de composition de workflow fournissent un outil de composition manuelle et semi automatique des services. Cependant, ils présentent quelques inconvénients :

- L'utilisateur est requis pour spécifier le workflow, chose qui n'est pas toujours praticable comme le cas de l'adaptation du contenu des services. Le type et le nombre de services requis pour l'adaptation sont déterminés durant l'exécution de la requête.
- La découverte et la sélection des services n'assurent pas le passage à l'échelle quand le nombre de services augmente.
- Si le service n'est pas disponible, l'exécution échoue.

À cause des inconvénients que nous avons cités précédemment, la composition manuelle ou semi automatique du workflow n'est pas tout à fait appropriée. Dans la section suivante, nous présentons les méthodes de composition automatique de services qui sont basées sur la planification de l' IA.

III.4.2.2 Composition de services à base d' IA

La seconde catégorie comprend des méthodes de planification à base de l'IA. L'hypothèse de base de ces méthodes est que chaque service est une action qui modifie l'état du monde réel à travers son exécution. Puisque les services (actions) sont des composants logiciels, les paramètres d'entrée/sortie agissent respectivement comme des préconditions et des effets dans le contexte de la planification. Si l'utilisateur peut spécifier les entrées et sorties requises par le service composé alors ce dernier est généré automatiquement par des planificateur de l'IA sans aucune connaissance d'un workflow prédéfini.

Avec l'introduction du Web sémantique, plusieurs méthodes de composition de service Web utilisant la planification de l'IA sont apparues dans la littérature [42] [43]. En général, un problème de planification peut être décrit comme un quadruplet (S, S0, G, A), avec :

- S : l'ensemble de tous les états possibles
- $S0 \in S$: l'état initial
- $G \in S$: le but à atteindre
- A : l'ensemble de toutes les actions possibles qui changent un état à un autre.

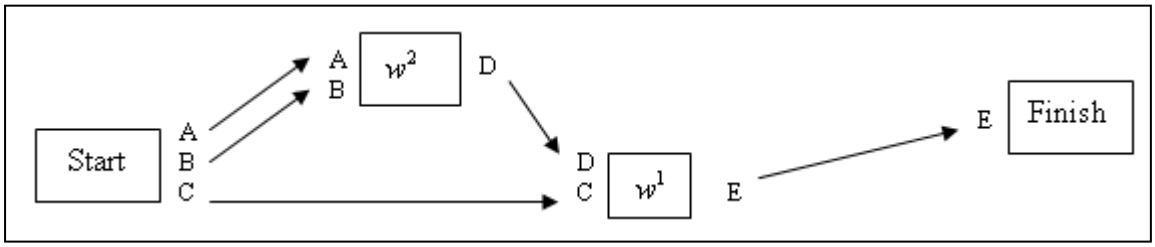
Étant donné une représentation des services comme des actions, le problème de composition de services est vu comme un problème de planification. De telle sorte qu'à partir de l'objectif de l'utilisateur et un ensemble de services Web, un planificateur doit trouver une collection de requêtes de services Web qui aboutissent à cet objectif. Cette ressemblance démontre la pertinence de la planification à base de l'IA pour entreprendre le problème de composition de service Web. Dans les sections suivantes, nous présentons quelques méthodes de composition de service Web basées sur la planification de l'IA.

Graphplan :

Dans ce modèle, le problème de composition de services est représenté par un quadruplet $\langle P, W, r_{in}, r_{out} \rangle$ tels que :

- P : ensemble de paramètres d'entrées / sorties des services. Les valeurs des paramètres des entrées d'un service sont considérées comme des préconditions de ce service, alors que les valeurs des ses sorties sont considérées comme des effets de ce même service.
- W : ensemble de services $W = \{ w_1, w_2, ..., w_n \}$
- r_{in} ($r_{in} \subseteq P$) : état initial
- r_{out} ($r_{out} \subseteq P$) : état final (but)

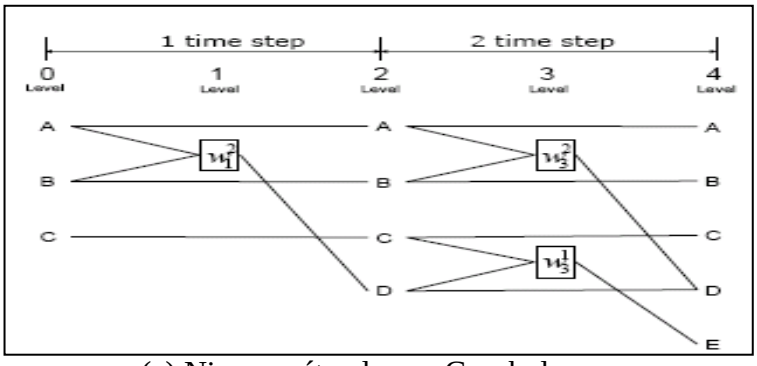
Nous illustrons l'application du modèle Graphplan sur l'exemple suivant :



P = {A, B, C, D, E}, $W = \{ w^1, w^2 \}$

r_{in} = {A, B, C}, r_{out} = {E}

Avec la planification Graphplan, le graphe de l'exemple précèdent est étendu en deux étapes appelées « *time steps* » afin d'atteindre le but recherché. Une étape « *time step* » contient deux niveaux : le premier niveau exprime les services dont les préconditions sont vérifiées, et l'action de maintient de paramètres. Le deuxième niveau exprime les effets de ces services et les paramètres maintenus. Un service w est considéré comme une action. Il est noté $w^{indice-service}_{indice-niveau}$. La figure suivante montre le graphe de planification obtenu pour cet exemple :

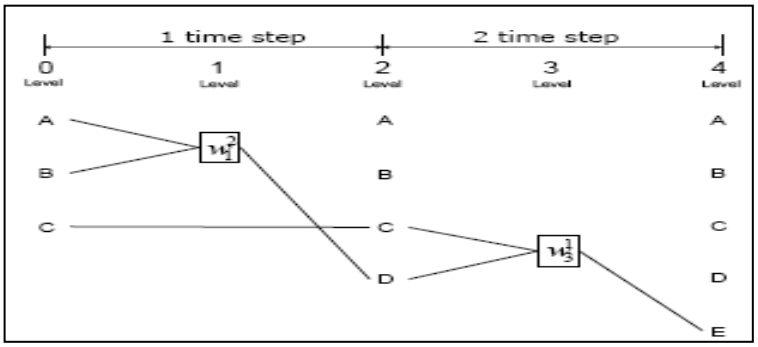


(a) Niveaux étendus en Graphplan

- Niveau 0 : commence avec l'état initial : A, B, et C
- Niveau 1: contient toutes les actions possibles dont les préconditions sont satisfaites au niveau 0. l'action w_1^2 est devenue possible grâce à A et B. les paramètres A, B, et C sont aussi maintenus tels quels sont pour le prochain niveau grâce à une action de maintient appelée « *no-op* ».
- Niveau 2 : contient tous les effets possibles des actions du niveau 1. A, B, et C sont toujours possible grâce à l'action de maintient. D devient aussi possible grâce à l'action w_1^2.

- Niveau 3 : contient toutes les actions du niveau 1 et les actions additionnelles. L'action w_3^1 devient possible par l'addition de D au niveau 2.
- Niveau 4 : contient tous les effets possibles des actions du niveau 3. E devient possible de l'action w_3^1. Ainsi, le but E recherché est satisfait à ce niveau.

Une fois que le but est atteint, Graphplan procède par une recherche en arrière (*backward*) afin de trouver un plan de composition valide comme la montre la figure suivante :



(b) graphe de solution de la planification

SATPlan :

Le Graphplan précédent peut être converti en un ensemble d'expressions logiques comme proposé par [34]. D'abord, l'état initial au temps 0 est exprimé par :

$A_0 \wedge B_0 \wedge C_0 \wedge \neg D_0 \wedge \neg E_0$

Le but au niveau final est décrit comme E_4. Ensuite, les relations entre les actions et leurs préconditions sont décrites comme suit :

$W_1^2 \rightarrow A_0 \wedge B_0$, $KeepA_1 \rightarrow A_0$, $KeepB_1 \rightarrow B_0$, $KeepC_1 \rightarrow C_0$, $W_3^2 \rightarrow A_2 \wedge B_2$,

$KeepA_3 \rightarrow A_2$, $KeepB_3 \rightarrow B_2$, $W_3^1 \rightarrow C_2 \wedge D_2$, $KeepC_3 \rightarrow C_2$, $KeepD_3 \rightarrow D_2$

KeepAction, correspond à l'action de maintient des action « no-op » dans Graphplan. De plus, les relations d'inférence entre chaque fait et toutes les actions du niveau précédent sont décrites comme des disjonctions :

$A_4 \rightarrow KeepA_3$, $B_4 \rightarrow KeepB_3$, $C_4 \rightarrow KeepC_3$, $D_4 \rightarrow W_3^2 \vee KeepD_3$, $E_4 \rightarrow W_3^1$,

$A_2 \rightarrow KeepA_1$, $B_2 \rightarrow KeepB_1$, $C_2 \rightarrow KeepC_1$, $D_2 \rightarrow W_1^2$

Finalement, les quatre expressions précédentes sont combinées dans une seul conjonction, comme une instance d'un problème de satisfiabilité (SAT), qui se résout par des méthodes appropriées telles que : la méthode complète qui se base sur des tables de vérité. La solution finale au problème est :

$A_0 \wedge B_0 \wedge C_0 \wedge W_1^2 \wedge KeepC_1 \wedge C_2 \wedge D_2 \wedge W_3^1 \wedge E_4$

HTN :

La planification HTN (*Hierarchical Task Network*) [65] est une méthodologie de planification à base de l'IA qui crée des plans par la décomposition des tâches. Les planificateurs HTN différent des planificateurs classiques par le pourquoi et le comment de la planification. L'objectif d'un planificateur HTN est de produire une séquence d'actions qui réalise une certaine activité ou tâche. La description du domaine de planification comprend un

ensemble d'opérateurs décrivant comment une tâche primitive peut être exécutée, et aussi un ensemble de méthodes, qui décrivent la manière de décomposer la tâche à des sous tâches. La planification procède, en utilisant ces méthodes, à la décomposition des tâches récursivement à des sous tâches de plus en plus petites, jusqu'à ce que le planificateur arrive à des tâches primitives qui peuvent être exécutées en utilisant les opérateurs de la planification. Plusieurs systèmes basés sur la technique HTN ont été proposés, le plus connu c'est le système SHOP2 (*Simple Hierarchical Ordered Planner* 2) [66].

La convenance de la planification HTN pour le problème de composition de services provient de la grande similarité opérationnelle entre le langage OWL-S et la planification HTN :

- Des processus OWL-S atomiques correspondent à des actions primitives.
- Des processus OWL-S composés correspondent à des actions complexes ou à des méthodes HTN
- OWL-S et HTN utilisent la décomposition hiérarchique pour traiter la complexité. Plus précisément, OWL-S utilise des modèles de contrôle pour décomposer les processus composés à des sous processus, et la planification HTN utilise des méthodes pour décomposer des actions complexes à des sous tâches simples.

Les processus composés dans OWL-S correspondent à des méthodes dans HTN, et les processus atomiques correspondent à des opérateurs dans HTN. Une fois ce *mapping* est effectué, et à partir d'un objectif et d'un état initial, le planificateur HTN trouvera une séquence de services Web qui réalise l'objectif.

COP :

Dans l'article [32], l'auteur propose une formalisation générique du problème de composition de service Web basée sur le problème d'optimisation de contraintes (COP : *constraint optimization problem*) ; cette formalisation est basée sur l'intervention incrémentale de l'utilisateur pour trouver l'optimal service Web composé selon quelques critères prédéfinis au moment de l'exécution. Le but de cette contribution est de traiter avec plusieurs caractéristiques de nature critique des services Web tels que l'environnement dynamique et distribué, et l'incertitude d'information sur les services Web, etc. Cette approche permet aussi l'intervention de l'utilisateur humain pour accroître le processus de résolution.

MDP :

Dans l'article [19], Doshi et al. proposent une nouvelle approche pour la composition automatique des services Web à base du workflow. Cette approche s'appuie sur le formalisme MDP (processus de décision Markovien) pour modéliser la nature dynamique et incertaine de l'environnement ainsi que le comportement stochastique des services Web. Le formalisme MDP utilisé dans cet article modélise la prise de décision comme un problème stochastique, de nature séquentiel et entièrement observable (avoir une perception complète de l'environnement qui est supposé accessible). Dans ce formalisme, le monde réel est modélisé par un ensemble d'états et d'actions. Une action n'aboutit pas toujours au même état, puisque les actions ne sont pas déterministes (incertaines). L'introduction de la fonction de transition permet de raisonner sur ce type d'actions. Une fonction du coût (ou inversement, de récompense) est également introduite afin de spécifier le coût de l'exécution de chaque action à partir de chaque états. La solution du MDP consiste donc à trouver la meilleure politique selon le critère de performance considéré. La politique est une application qui, à un état du système, associe une action à effectuer. C'est simplement un passage (*mapping*) des états aux

actions. La solution ainsi trouvée est une séquence de politiques indicées par le temps qui spécifie l'action optimale à exécuter pour chaque état. La solution est obtenue après une certaine période de temps appelée horizon. Si l'horizon est infini la politique obtenue est dite stationnaire, sinon la politique n'est pas stationnaire car la bonne action à exécuter peut dépendre du temps restant. Toutefois, les auteurs ne spécifie pas la manière de composer les services, mais il suppose que le schéma de composition existe déjà sous forme d'un workflow prédéfini dans lequel, les coûts et les probabilités de passage pour chaque service sont connus à l'avance. De plus, cet article, ne donne ni modélisation ni représentation du contexte. Il fait complètement abstraction de l'architecture de déploiement et des détails de l'implémentation.

III.4.3 Prise en compte du contexte dans la composition de services

La composition de services à partir de fonctionnalités présentes dans un environnement ubiquitaire doit évidemment tenir compte des informations de contexte. Il s'agit en fait de combiner un ensemble de services pour réaliser une tâche donnée dans un contexte particulier. On parle ainsi d'un problème de composition de services sensibles au contexte. Ce genre de problèmes sont soumis à deux types de contraintes : des contraintes liées à la tâche à effectuer et des contraintes liées au contexte.

Malgré de nombreux travaux qui ont porté sur la composition de service et sur le domaine du contexte, peu de travaux qui ont adressé la problématique de composition de services sensibles aux contexte. Cette problématique a été étudiée notamment dans le domaine de l'informatique ubiquitaire et mobile [36] [53], afin de permettre l'intégration d'information de contexte dans un processus de composition par la planification à base de l'IA. Nahrstedt et al. [49] utilisent l'idée des algorithmes de planification globale pour la composition dynamique de services. Les auteurs proposent un modèle qui calcule d'abord un plan initial. Par la suite, ce plan est révisé si nécessaire durant l'exécution. Zhao et al. [80] proposent un modèle de composition dynamique appliqué aux services composites spécifiés dans BPEL4WS, afin d'améliorer la QoS. Dans l'article [58], les auteurs se basent sur les *statecharts* et le DAG (*Direct Acyclic Graph*) pour représenter les plans d'exécution de la composition. Mais ils ne proposent pas un algorithme formel pour construire le DAG dynamiquement. Vukovic [72] propose aussi un système dans lequel la composition est influencée par le contexte. Ce système se base sur le planificateur SHOP2 et BPEL4WS. Saif et al. [62] intègrent l'aspect contexte dans un mécanisme de planification dans un environnement intelligent, et indiquent que la manière de satisfaire un but particulier peut dépendre du contexte. Mokhtar et al. [45] présentent une approche pour la composition de services sensibles au contexte basée sur l'intégration du workflow dans un environnement ubiquitaire. Dans [38] [39], Maamar et al. discutent la manière d'utiliser le contexte dans la personnalisation des services Web. Dans un cadre de composition de services Web pour l'informatique mobile, Mostéfaoui et al. [47] définissent des informations de contexte pertinentes au niveau d'un service particulier appelées *contexte d'intérêt*. La composition est ainsi dirigée non seulement par la satisfaction d'un but de haut niveau, mais aussi par les contraintes des services.

III.5 Synthèse

Dans ce chapitre, nous avons détaillé la notion de composition de services, et nous avons donné un panorama de méthodes qui sont utilisées pour réaliser cette composition, notamment celles basées sur le workflow et les techniques de planification à base de l'IA. Les avancés réalisées dans le domaine du Web sémantique en terme de représentation de connaissances ont ouvert des nouvelles voies de recherches sur la composition de services. En effet, partant d'un objectif bien défini et d'une spécification claire et compréhensible des services, plusieurs méthodes de planification à base de l'IA peuvent êtres exploitées pour réaliser un plan de composition de services.

Dans un environnement ubiquitaire, la composition de services doit aussi prendre en compte la notion du contexte afin de fournir des services adaptés aux besoins des utilisateurs. Ces services sont sensibles au contexte puisqu'ils s'adaptent dynamiquement aux changements du contexte. Ce qui fait, même leur composition doit s'adaptée à ces changements. Des récentes recherches réalisées par Doshi et al.[19] ont montré l'efficacité des processus de décision Markoviens (MDP) quant à la modélisation de la nature dynamique et incertaine de l'environnement ainsi que le comportement stochastique des services Web.

Notre travail s'inscrit dans la continuité de ces travaux et vise à proposer un modèle de composition de services sensibles au contexte. Ce travail passe d'abord par la modélisation du contexte en se basant sur les quatre classes que nous avons défini précédemment (voir synthèse, chapitre II), puis la modélisation des services sensibles au contexte. Ensuite, nous proposons un modèle pour la construction d'un plan de composition d'une façon automatique en s'inspirant des modèles Graphplan et SATPlan. Nous introduisons aussi les processus de décision Markovien (MDPs) et le modèle d'apprentissage Bayésien afin de tenir compte du comportement non déterministe des services et de l'incertitude sur le contexte.

Partie 2

Proposition d’un modèle de composition de services

Chapitre IV

Construction automatique d’une composition de services

IV.1 Introduction

Dans ce chapitre, nous présentons la première partie de notre modèle de composition de services sensibles au contexte. Dans une première étape, il s'agit de modéliser le contexte en se basant sur l'approche par ontologie afin de faciliter son utilisation d'une manière standard. Nous avons scindé ce contexte en quatre classes : le contexte utilisateur, le contexte environnement, le contexte ressources, et le contexte service. Les trois premières classes sont modélisées en OWL, tandis que la dernière est modélisée en OWL-S. Dans ce travail, nous considérons deux types de services à savoir : les services concrets et les services abstraits. Un service concret est une action qui fourni un résultat après son exécution. Un service abstrait est un service qui n'est pas directement exécutable, mais il contient un ensemble de services concrets qui fournissent les mêmes fonctionnalités. Dans une seconde étape, nous proposons une architecture générale pour la composition de services sensibles au contexte. Cette architecture comporte plusieurs modules qui collaborent entre eux afin de fournir un service adapté aux besoins de l'utilisateur. Dans une troisième étape, nous détaillons l'approche proposée pour la vérification et la construction automatique d'un plan de composition. Nous présentons également l'algorithme FCoSC *(Feasibility & Construction of Service Composition)* que nous avons développé pour cette première partie.

IV.2 Modélisation du contexte

IV.2.1 Approche basée ontologie pour représenter les classes du contexte

Nous avons défini dans le chapitre II quatre classes de contexte à savoir : le contexte utilisateur *(user-context)*, le contexte service *(service-context)*, le contexte environnement *(env-context)*, et le contexte ressource *(resource-context)* :

- **Le contexte utilisateur *(user-context)* :** toute information relative à l'utilisateur telles que : ces préférences (centres d'intérêts), les informations indiquant son état physique ou émotionnel (la température corporelle, le rythme cardiaque, etc.), sa localisation, son activité courante, etc.

- **Le contexte service *(service-context)* :** toute information non fonctionnelles du service telles que : son coût, sa durée d'exécution, sa disponibilité, sa fiabilité, et sa réputation (renommé du fournisseur du service). Ce contexte représente, en quelque sorte, les informations considérées par le modèle de qualité de service *(QoS : Quality of Service).*

- **Le contexte environnement *(env-context)* :** contient des informations sur l'environnement de l'utilisateur (où est l'utilisateur ?) telles que : le climat, la température, la luminosité, le bruit, la date, et quelques données sur la situation entourant l'utilisateur (les personnes ou objets à proximité), etc.

- **Le contexte ressource *(resource-context)* :** contient toutes les informations sur les équipements (ressources) dont dispose l'utilisateur tels que : le terminal utilisé, le réseau de communication, les appareilles qui assistent l'utilisateur (notamment une personne physiquement dépendante) dans sa vie quotidienne. Ainsi les propriétés de ces équipements : dispositif d'affichage, puissance de calcule, capacité de stockage, énergie, débits réseau, etc.

La relation entre les quatre classes du contexte est définie par le graphe ci-dessous :

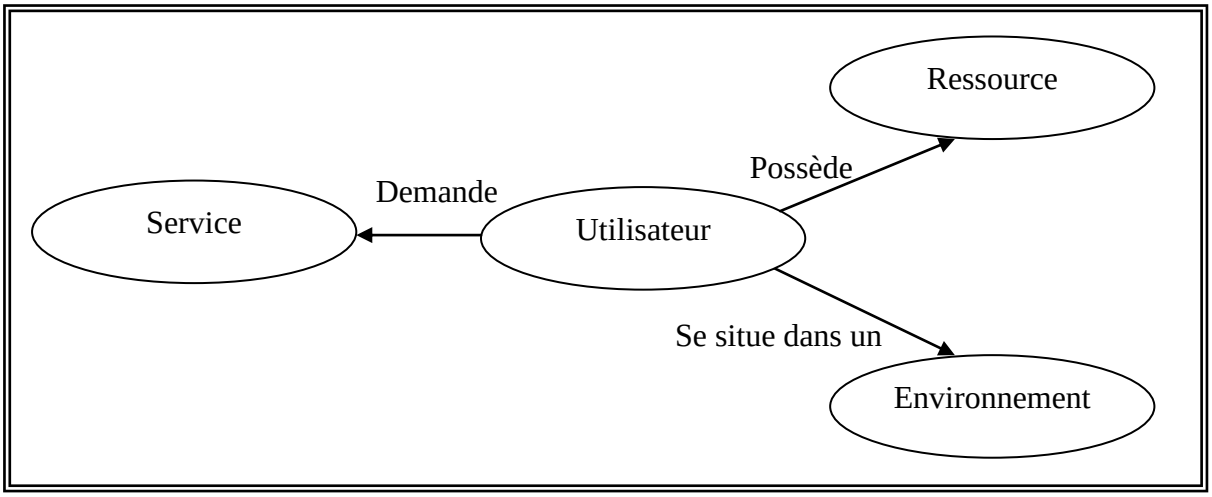


Fig. IV.1 : Graphe des quatre classes du contexte

IV.2.2 Exemple d'instanciation

Le schéma suivant illustre un exemple d'applications des quatre classes précédentes. Soit une personne appelée « Josef » qui possède un équipement « PDA » et se trouve en ville de « New York ». Cette personne désire effectuer un voyage.

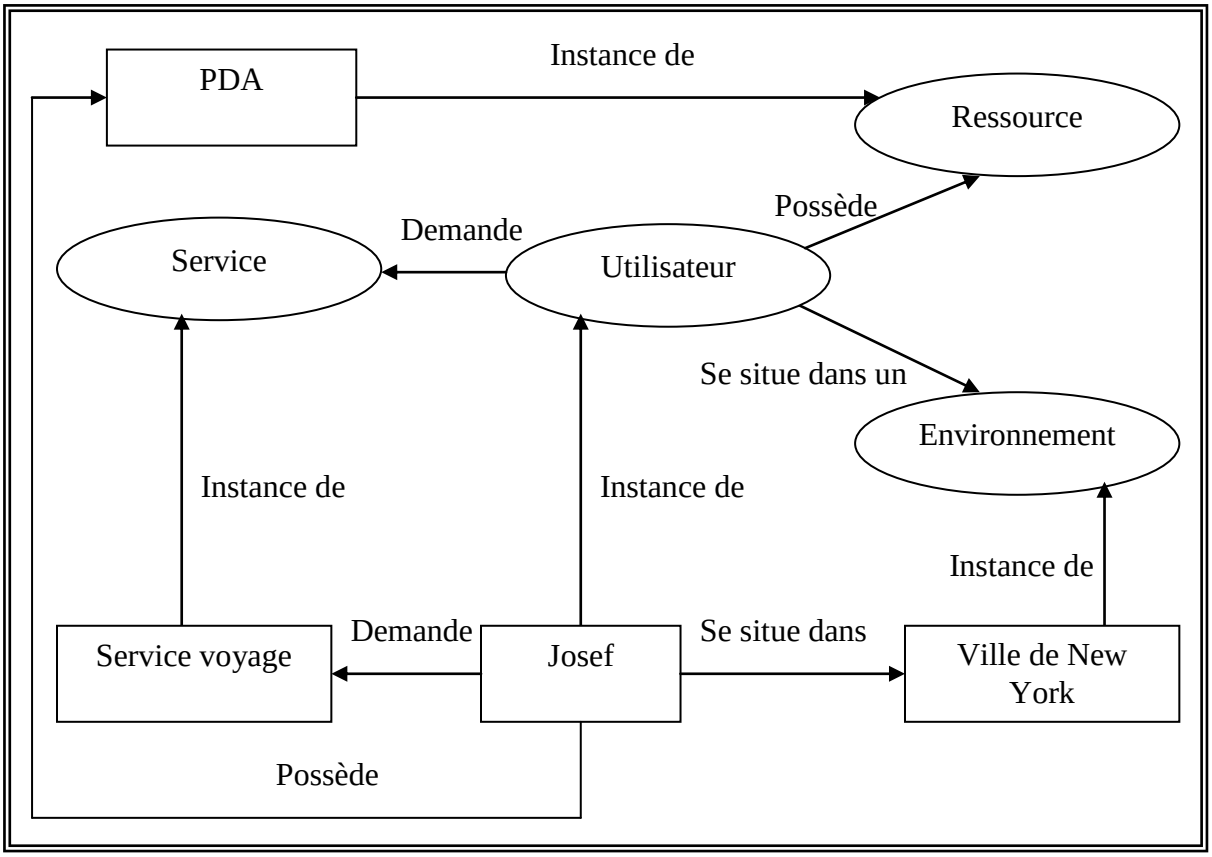


Fig. IV.2 : Exemple d'instanciation des classes de contexte

Le contexte est représenté sous forme de prédicats logiques. Un état contextuel est une conjonction de prédicats. Par exemple : *At (New York, Josef) ∧ Destination (Boston)∧ Has (PDA).* Selon le domaine d'application, des prédicats appropriés sont utilisés.

IV.2.3 Outils pour représenter l'ontologie du contexte

Dans l'ontologie que nous proposons, la description fonctionnelle des connaissances contextuelles est représentée en OWL à travers trois classes principales: *User*, *Environment*, et *Resource*. La classe *Service* est représentée en OWL-S. Le schéma ci-dessous illustre la représentation en OWL et OWL-S des quatre classes du contexte :

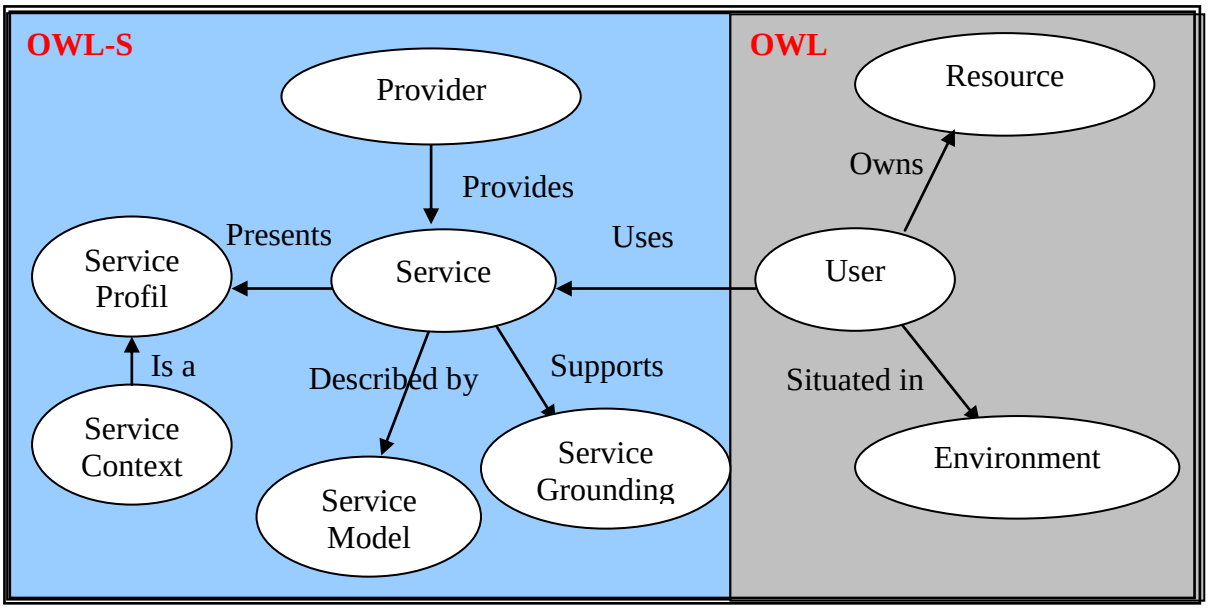


Fig. IV.3 : Représentation des classes du contexte en OWL/OWL-S

OWL : Le schéma précédent peut être représenté en OWL comme suit :

```
<owl:Class rdf:ID="User"/>
<owl:Class rdf:ID="Environment"/>
<owl:Class rdf:ID="Resource"/>
<owl:Class rdf:ID="Service"/>

<owl:ObjectProperty rdf:ID="Situated in">
<rdfs:range rdf:resource="#Environment"/>
<rdfs:domain rdf:resource="#User"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Owns">
<rdfs:range rdf:resource="#Resource"/>
<rdfs:domain rdf:resource="#User"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Uses">
<rdfs:range rdf:resource="#Service"/>
<rdfs:domain rdf:resource="#User"/>
</owl:ObjectProperty>
```

Fig. IV.4 : Représentation des connaissances contextuelles en OWL

La notion du temps est prise implicitement dans ce modèle, puisque le temps constitue un contexte de la classe *environment*. Par conséquent, ce modèle décrit le contexte d'un utilisateur à un instant donné, ainsi que le service demandé par ce dernier.

Il est facile d'étendre les classes précédentes à l'aide des constructeurs *owl:subclassOf* (sous classe) et *owl:subpropertyOf* (sous propriété). Ces constructeurs permettent de définir de nouvelles classes ou de nouvelles propriétés, permettant une description plus détaillée de la classe mère. Par exemple, on peut spécifier que la classe « *Device* » est une sous classe de la classe « *Resource* ».

```
<owl:Class rdf:ID="Device ">
<rdfs:subClassOf rdf:resource="Resource" />
<owl:Class>
```

OWL-S : contient trois sous ontologies inter reliées : *service profil, service model, et service grounding. Service profile* est l'ontologie qui nous intéresse dans cette partie. En effet, elle permet de décrire un service en terme d'offre à la communauté, constituée principalement des : entrées, sorties, préconditions, et effets qui sont résumés en IOPE (Inputs, Outputs, Preconditions, Effects). OWL-S possède un mécanisme extensible pour les paramètres du service. Ce mécanisme permet une description additionnelle des attributs non IOPE du service. Donc, il est possible étendre OWL-S avec des attributs du contexte. Nous proposons d'ajouter les informations de contexte du service comme sous classe de la partie *Service Profil.* Cette sous classe appelée « *Service Context* » sert à définir plus en détail les paramètres de qualité de service. Voici un exemple de contexte d'un service :

```
type= "W-service"; // Web service
isMobile= "No" ; // whether the service can move to other places
price = 5; // per invocation
time = 10; // per execution
provider-identifier ="Rezyd" ;
secured = yes;
protocol = https;
port= 80;
probability = 0.8; // service response probability
```

IV.3 Modélisation d'un service sensible au contexte

Dans notre approche, nous proposons deux types de services sensibles au contexte à savoir : les services concrets et les services abstraits.

IV.3.1 Service concret (*concret service*)

Définition : Un service concret est un service qui réalise une certaine fonctionnalité en agissant sur des données en entrée afin de produire des données en sortie. Il possède deux types de paramètres : (1) les paramètres fonctionnels représentés par les *entrées* et *sorties* du service. (2) les paramètres non fonctionnels représentés par le *QoS* du service. Les conditions

et les valeurs des entrées sont appelées *préconditions* alors que les valeurs des sorties sont appelées *effets*. Le service s'adapte au contexte de l'utilisateur selon un certain nombre de règles appelées règles de contexte.

Notation : Un service concret i est noté cs_i. Il est décrit par un tuplet $\langle cs_i^{in}, cs_i^{out}, préc, eff, Rc, QoS \rangle$ tels que :

cs_i^{in} : ensemble des entrées de cs_i ;

cs_i^{out} : ensemble des sorties de cs_i ;

$préc(cs_i)$: ensemble des préconditions de cs_i ;

$eff(cs_i)$: ensemble des effets de cs_i ;

$Rc(cs_i)$: ensemble des règles de contexte de cs_i.

$QoS(cs_i)$: ensemble des paramètres de qualité de service (QoS) de cs_i.

Le schéma suivant représente un service concret cs_i :

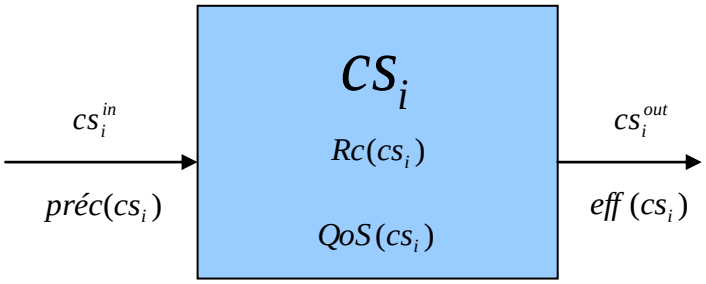


Fig. IV.5 : Représentation d'un service concret

IV.3.2 Les règles de contexte (Rc)

Définition : Les règles de contexte présentent trois avantages à savoir : l'inférence de nouvelles connaissances contextuelle à partir du contexte courant de l'utilisateur, l'exécution des actions appropriées pour un certain contexte, et l'adaptation des sorties du service en fonction du contexte. Plusieurs règles peuvent êtres définies et intégrées par des experts de différents domaines (médecine, commerce, …etc.). Ces règles sont de la forme :

- ***Si*** *antécédent* ***alors*** *conséquent*, noté : *antécédent* ===> *conséquent*
- antécédent et conséquent = Liste d'atomes (conjonction)
- un atome est un prédicat unaire ou binaire de la forme (C(x), P(x, y)).

Exemples :

- Si John est au travail Alors John est indisponible.
  $Personne(John) \wedge Travail(John) \Rightarrow Indisponible(John)$

- Si une personne est immobile et son rythme cardiaque élevé alors elle est malade.
  $Personne(x) \wedge immobile(x) \wedge rythme\_cardiaque(x, élevé) \Rightarrow malade(x)$

- Si une personne est en classe entre 8h00 et 10h00, alors son téléphone portable passe en mode silencieux.

$Personne(x) \wedge localisation(x, classe) \wedge heure(y) \wedge domaine(y, [8,10]) \Rightarrow portable(z, x)$
$\wedge silencieux(z)$

- Si la présentation commence alors éteindre les projecteurs et baisser les rideaux.
  $Présentation(x) \wedge commence(x) \Rightarrow projecteur(y, x) \wedge éteindre(y) \wedge rideaux(z, x)$
  $\wedge baisser(z)$

Les règles de contexte peuvent êtres également spécifiées sous forme d'un tableau qui indique aux services concrets la façon avec laquelle ils doivent s'adapter pour un contexte donné. Le service concret cherche dans les entrées du tableau le cas correspondant au contexte courant. Si le cas est trouvé alors le service adapte ses sorties, sinon il garde les sorties par défaut. Voici le tableau de correspondance entre le contexte et les sorties du service :

Contexte / Cas	Contexte utilisateur	Contexte environnement	Contexte ressource	Sorties adaptées
Cas 1	Cu_1	Ce_1	Cr_1	Sa_1
Cas 2	Cu_2	Ce_2	Cr_2	Sa_2
. . .	. . .	. . .	. . .	. . .
Cas n	Cu_n	Ce_n	Cr_n	Sa_n

Tableau IV.1 : Tableau d'adaptation des sorties selon le contexte

Exemple : le tableau ci-dessous montre un exemple d'adaptation des sorties d'un service qui transmet un flux vidéo selon le contexte d'un utilisateur.

Contexte / Cas	Entrées						Sorties	
	Contexte Utilisateur		Contexte Ressource		Contexte Environnement		Sorties adaptées	
	Activité	Localisation	Réseau	Equipement	Voisinage	Bruit	Son	Affichage
Cas 1	conduite	Dans le véhicule	GPRS	PDA	Seul	Moyen	Haut	adapté
Cas 2	En marche	Dans la rue	GPRS	Téléphone portable	Seul	Moyen	Haut	adapté
Cas 3	au travail	Dans le bureau	LAN	Ordinateur portable	Personnels	Bas	Bas	complet

Tableau IV.2 : Exemple de correspondance du contexte

IV.3.3 Estimation du QoS d'un service concret

L'estimation de la qualité de service pour un service concret se base sur les préférences de l'utilisateur qui constituent une partie importante de son contexte. Les services sont classés par ordre décroissant pour chaque paramètre *i* du QoS noté : PQ_i. Par exemple : pour le paramètre coût, le service qui possède le coût min sera classé en première position, et le service qui possède le coût max sera classé en dernière position. De même pour le paramètre sécurité, le service le plus sécurisé se classe en première position et le moins sécurisé en dernière position. Le tableau suivant donne le classement des services par rapport à chaque paramètre PQ_i :

Paramètres QoS / *Services*	PQ_1	PQ_2	...	PQ_k
Service 1	$C_{1,1}$	$C_{1,2}$	...	$C_{1,k}$
Service 2	$C_{2,1}$	$C_{2,2}$	...	$C_{2,k}$
⋮	⋮	⋮	⋮	⋮
Service n	$C_{n,1}$	$C_{n,2}$	...	$C_{n,k}$

Tableau IV.3 : classement des services selon les paramètres QoS

$C_{i,k}$: représente le classement du service i par rapport au paramètre PQ_k.

Les préférences de l'utilisateur se manifestent sous forme d'un ensemble de poids attribués pour chaque paramètre QoS. Le poids P_k reflète le degré d'importance du paramètre PQ_k pour l'utilisateur. Ainsi, la qualité de service pour un service i par rapport à un utilisateur est estimée par la formule suivante :

$$R(i) = \sum_k P_k * (\frac{1}{C_{i,k}}).$$

Nous appelons cette quantité la récompense obtenue en exécutant le service i. elle est notée $R(i)$. Nous verrons plus en détail cette notion de récompense dans le chapitre qui suive.

IV.3.4 Exemples de services concrets

Un premier exemple concerne un service qui permet de débiter un compte bancaire : les entrées sont le code bancaire et la somme à débiter. Les sorties sont la somme débitée et une confirmation du retrait. Les préconditions sont : le code doit être valide, la somme à débiter doit être inférieur à la somme disponible dans le compte. Les effets sont : la somme disponible sera mise à jour. Les paramètres QoS du service sont : le niveau de sécurité, le temps de retrait, le coût du retrait, la somme disponible, …etc. Dans cet exemple l'ensemble de règles de service est vide.

Un second exemple concerne un service de surveillance par caméra : son entrée est la position à surveiller. La sortie est un flux vidéo. La précondition est que la position à surveiller doit appartenir à l'une des position suivante : salon, chambre, cuisine (par exemple une caméra mobile portée sur un robot qui peut couvrir ces trois zones). Les effets sont : un affichage vidéo et une notification de l'état de la position spécifiée. L'ensemble des règles de contexte contient deux règles :

- On considère que si le salon est vide (personne ne se trouve dedans) et les équipements qui s'en trouvent (télévision, radio, etc.) sont allumés alors le service effectue une notification afin de les éteindre. Cela se traduit par la règle suivante :

$$position(y, salon) \wedge vide(y) \wedge equipement(z, [TV, Lampe, radio]) \wedge allumé(z) \Rightarrow eteindre(z)$$

- On considère également que dans le cas du four, la situation est anormale lorsque la porte est restée ouverte alors que le four est allumé. Cela se traduit par la règle suivante :

$$position(y, cuisine) \wedge equipement(z, four) \wedge allumé(z) \wedge porte(z, ouverte) \Rightarrow alerte\_sur(z)$$

IV.3.5 Service abstrait (*abstract service*)

Définition : Un service abstrait est une tâche qui peut être réalisée par plusieurs services concrets. Ces derniers offrent la même fonctionnalité et possèdent les mêmes entrées et sorties. Un service abstrait représente donc une classe de services concrets d'un domaine particulier.

Notation : Un service abstrait i est noté AS_i. Il est décrit par un triplet $\langle AS_i^{in}, AS_i^{out}, AS_i^{CS} \rangle$:

AS_i^{in} : ensemble des entrées de AS_i

AS_i^{out} : ensemble des sorties de AS_i

AS_i^{CS} : ensemble des services concrets de AS_i tel que $AS_i^{CS} = \{cs_{i,1}, cs_{i,2}, ..., cs_{i,n}\}$ et $\forall cs_{i,j}, cs_{i,k} \in AS_i^{CS} / (cs_{i,j}^{in} = cs_{i,k}^{in} = AS_i^{in}) \wedge (cs_{i,j}^{out} = cs_{i,k}^{out} = AS_i^{out})$

Le schéma suivant représente un service abstrait AS_i :

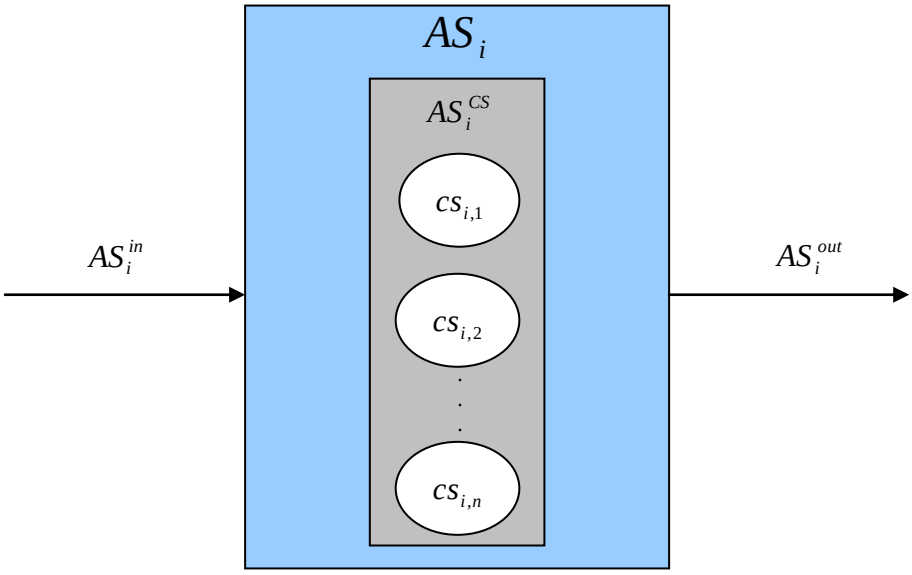


Fig. IV.6 : Représentation d'un service abstrait

IV.3.6 Exemples de services abstraits

Un service de vol par exemple peut être rendu par plusieurs compagnies aériennes. Le service de vol est un service abstrait alors que les services de chaque compagnie aérienne sont des services concrets.

Les exemples sont nombreux dans le domaine de l'informatique ubiquitaire ou un même service peut être rendu par plusieurs dispositifs. Un service d'affichage par exemple peut être rendu par plusieurs dispositifs tels que : une TV, un PDA, un smart phone, …etc. Une imagerie numérique peut être rendue par plusieurs caméras telles que : une caméra à basse résolution, une caméra à haute résolution, une caméra externe à la maison, une caméra interne, …etc.

Remarque : un service abstrait est un ensemble de services concrets ayant tous des informations de contexte différentes. Le service abstrait est réalisé par le service concret dont le contexte est le plus proche de la situation de l'utilisateur. Si la situation de l'utilisateur change, le service abstrait sera réalisé par un autre service concret dont le contexte est le mieux adapté à la nouvelle situation. Donc, d'après notre définition, un service abstrait est un service sensible au contexte puisqu'il s'adapte à la situation de l'utilisateur en offrants des services concrets pertinents. De même, un service concret est sensible au contexte puisqu'il s'adapte selon un certain nombre de règles de contexte.

IV.4 Architecture du système de composition de services

Le schéma ci-dessous (Fig. IV.7) représente l'architecture de notre système de composition de services sensibles au contexte. Tout d'abord, La requête de demande de services peut être envoyée par plusieurs acteurs à savoir : l'utilisateur lui même, un intergiciel sensible au contexte, ou un système automatique intégré dans l'environnement de l'utilisateur, par exemple un système de surveillance de l'utilisateur, ou un agent assistant chargé d'accompagner l'utilisateur. Il est également à signaler qu'un intergiciel sensible au contexte récupère le contexte et le stocke dans une base de contexte en respectant l'ontologie que nous avons définie précédemment. Ce contexte sera utilisé par les services afin de s'adapter à la situation de l'utilisateur.

Une fois la requête reçue, un module d'interprétation de requêtes se charge de l'interpréter en un ensemble de tâches à réaliser. Ces tâches seront envoyées par la suite au module de découverte et de composition de services qui construit un plan de composition de service. Ce plan sera transmis au module de sélection et d'exécution de services qui s'occupe de la sélection des services pertinents afin d'exécuter le plan de composition. Ce module permet à l'ensemble de services concrets d'accéder à la base de contexte.

Un module de découverte de services cherche les services proposés par les fournisseurs de services et les stocke dans la base de services concrets. Ce module effectue une recherche périodique afin de tenir compte des modifications au niveau des services et mettre à jour la base. À chaque service abstrait dans la base de services abstrait correspond un ensemble de services dans la base de services concrets. Cette correspondance est effectuée par un processus de classification des services concrets.

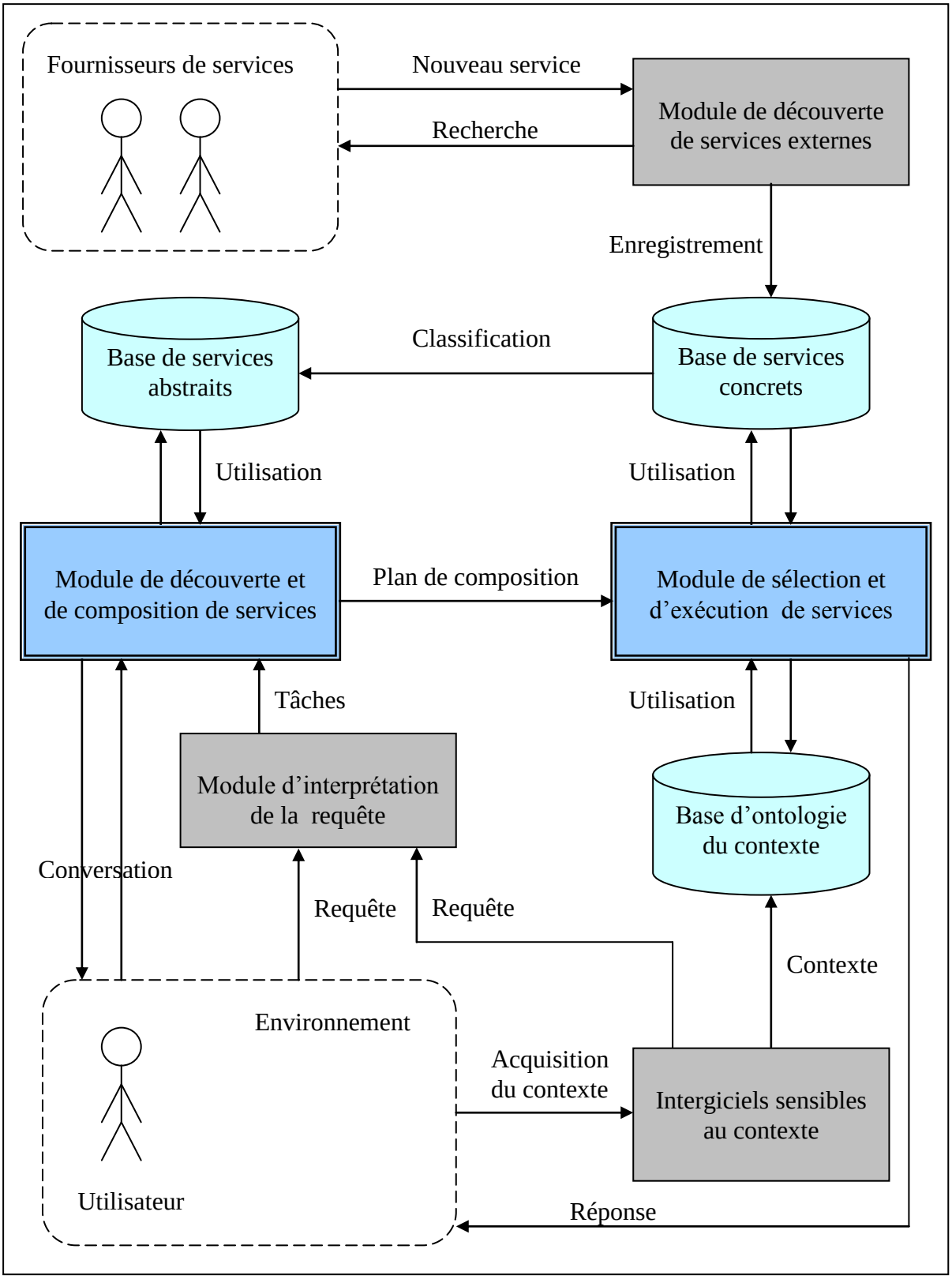


Fig. IV.7 : Architecture du système de composition de services

IV.5 Approche pour la composition des services

IV.5.1 Hypothèses et définitions

Dans cette partie, nous développons le module de découverte et de composition de services. Nous proposons une approche de composition automatique de services qui se base sur les hypothèses suivantes :

H1 : Les services abstraits disponibles sont déjà découverts par le module de découverte de services externes (voir schéma précédent), et se présentent sous la forme suivante :

AS_i	AS_i^{in}	AS_i^{out}	AS_i^{CS}
AS_1	$e_{1,1}, e_{1,2}, ..., e_{1,g}$	$s_{1,1}, s_{1,2}, ..., s_{1,h}$	$cs_{1,1}, cs_{1,2}, ..., cs_{1,k}$
AS_2	$e_{2,1}, e_{2,2}, ..., e_{2,i}$	$s_{2,1}, s_{2,2}, ..., s_{2,j}$	$cs_{2,1}, cs_{2,2}, ..., cs_{2,l}$
. .	. .	. .	. .
AS_n	$e_{n,1}, e_{n,2}, ..., e_{n,p}$	$s_{n,1}, s_{n,2}, ..., s_{n,q}$	$cs_{n,1}, cs_{n,2}, ..., cs_{1,m}$

Tableau IV.4 : Services abstraits

Ce tableau est noté AS et contient N services abstraits.

H2 : La requête reçue par le module de composition est considérée comme une conjonction de tâches, indépendantes l'une de l'autre, avec leurs paramètres d'entrées/sorties respectifs. Elle se présente sous la forme suivante :

$$T_1(T_1^{in}, T_1^{out}) \wedge T_2(T_2^{in}, T_2^{out}) \wedge ... \wedge T_n(T_n^{in}, T_n^{out})$$

H3 : Un service abstrait doit être invoqué avec tous ses paramètres d'entrées et avec au moins un paramètre de sortie. Cela signifie que tous les paramètres d'entrée d'un service sont obligatoires, alors que ses paramètres de sortie peuvent ne pas être entièrement utilisés.

Définition 1 : Un plan de composition noté *plan_composition*[][] est une matrice qui contient tous les services abstraits de chaque tâche T_i de la requête, tel que les lignes représentent les indices des tâches, et les colonnes représentent les services qui composent les tâches. L'ordre séquentiel des services correspond à l'ordre croissant des indices des colonnes.

Définition 2 : Un problème de découverte de services pour satisfaire une tâche $T_i(T_i^{in}, T_i^{out})$ consiste à trouver un service $AS_j \in AS$ tel que : $(AS_j^{in} \subseteq T_i^{in}) \wedge (AS_j^{out} \supseteq T_i^{out})$.

L'algorithme de découverte de services pour une tâche (SDT : *Service Discovery for Task)* est le suivant :

Algorithme SDT (T_i)

Début
$découvert \leftarrow faux$
$j \leftarrow 1$
Tant que $(j \leq N) \wedge (\neg(découvert))$ faire

Si $(AS_j^{in} \subseteq T_i^{in}) \wedge (AS_j^{out} \supseteq T_i^{out})$ alors
$découvert \leftarrow vrai$
$plan\_composition[i][1] \leftarrow AS_j$
Sinon
$j \leftarrow j+1$
Fin si
Fin tant que
Retourner *découvert*
Fin

Définition 3 : Un problème de composition de services pour satisfaire une tâche $T_i(T_i^{in}, T_i^{out})$ consiste à trouver une séquence de services $AS_1, AS_2, ..., AS_n$, tels que :

1. Les AS_i sont invoqués séquentiellement de 1 à n
2. $(T_i^{in} \cup AS_1^{out} \cup AS_2^{out} \cup ... \cup AS_n^{out}) \supseteq T_i^{out}$
3. Le nombre total de services n est minimisé
4. La récompense totale $\sum_{i=1}^{n} R(AS_i)$ est maximisée
5. La probabilité de réponse de chaque AS_i est maximisée

Un problème de composition de services comporte alors deux étapes essentielles :

1. Vérification de la faisabilité d'une composition de services et construction automatique d'un plan pour cette composition dans le cas où elle est faisable. Cette étape vérifie les trois premières conditions (1, 2, 3).

2. Sélection des services concrets pour réaliser le plan de composition construit lors de la première étape. Cette étape vérifie les deux dernières conditions (4,5).

Définition 4 : Soient $AS_i, AS_j \in AS$:

- Matching total : AS_i peut *matcher totalement* AS_j si est seulement si $AS_i^{out} \supseteq AS_j^{in}$

- Matching partiel : AS_i peut *matcher partiellement* AS_j si est seulement si :

$(AS_i^{out} \not\supseteq AS_j^{in}) \wedge (AS_i^{out} \cap AS_j^{in} \neq \emptyset)$

Définition 5 : La source d'un paramètre p notée $source(p)$ est le service abstrait AS_i qui délivre ce paramètre. $source(p) = \{ AS_i / p \in AS_i^{out} \}$

IV.5.2 Principe de l'approche proposée

Une composition de services dans laquelle les services sont matchés totalement ou partiellement pose les problèmes suivants :

1. Un même paramètre d'entrée d'un service peut avoir plusieurs sources, alors que le service a besoin uniquement d'une seule source pour ce paramètre.
2. Des services AS_k en plus peuvent apparaître lors de la composition. Ces services satisfassent les deux conditions suivantes :
   - à une étape donnée de la composition, tous les paramètres d'entrée de AS_k sont satisfaits ;
   - mais les sorties de AS_k ne participent ni au sorties de la tâche à réaliser T_i^{out} ni aux entrées des autres services AS_j du $plan\_composition[i][]$. Cela se traduit par : $(\forall AS_j \in plan\_composition[i][]) \wedge (\forall p \in (AS_j^{in} \cup T_i^{out}) : p \notin AS_k^{out}$.

Notre approche consiste alors à sélectionner une et une seule source pour chaque paramètre qui participe à la composition. Elle consiste également à conserver uniquement les paramètres et les services qui participent effectivement à la composition.

Proposition 1 : Chaque paramètre p possède une et une seule source si et seulement si $(\forall AS_i, AS_j \in AS \ / \ AS_i^{out} \cap AS_j^{out} = \phi)$.

Preuve :

Par définition d'une source d'un paramètre p : $source(p) = \{ AS_i / p \in AS_i^{out} \}$. Supposant d'abord que chaque paramètre possède une et une seul source. Soit deux services quelconques AS_i et AS_j de AS. Soit un paramètre p quelconque tel que $p \in AS_i^{out}$ donc $source(p) = AS_i$. Si on a encore $p \in AS_j^{out}$ alors $source(p) = AS_j$, mais cela signifie que p possèdent plusieurs sources, ce qui contredit la supposition. Donc $(\forall AS_i, AS_j \in AS \ / \ AS_i^{out} \cap AS_j^{out} = \phi)$.

Montrer l'autre sens de l'équivalence, c'est-à-dire : si $(\forall AS_i, AS_j \in AS \ / \ AS_i^{out} \cap AS_j^{out} = \phi)$ alors chaque paramètre p possède une et une seule source, revient à montrer que : s'il existe au moins un paramètre p qui possède plusieurs sources alors $\exists AS_i, AS_j \in AS / AS_i^{out} \cap AS_j^{out} \neq \phi$. Soit un paramètre p. Supposant que p possède plusieurs sources : Alors $\exists AS_i, AS_j \in AS / (p \in AS_i^{out}) \wedge (p \in AS_j^{out})$ donc $AS_i^{out} \cap AS_j^{out} \neq \phi$ puisqu'au moins le paramètre p est commun entre les deux services. Donc $\exists AS_i, AS_j \in AS / AS_i^{out} \cap AS_j^{out} \neq \phi$.

D'ou la preuve de la proposition 1.

En se basant sur la proposition 1, nous déduisons que: si $\exists AS_i, AS_j \in AS / AS_i^{out} \cap AS_j^{out} \neq \phi$ alors il existe au moins un paramètre p qui possède plusieurs sources. Nous posons alors un certain nombre de règles pour traiter ce cas de figure.
$AS_i^{out} \cap AS_j^{out} \neq \phi \Rightarrow$ Trois cas possibles :

$$\begin{cases} 1.\ (AS_i^{out} \subset AS_j^{out}) \vee (AS_i^{out} \supset AS_j^{out}) \\ 2.\ AS_i^{out} = AS_j^{out} \\ 3.\ (\exists p / p \in AS_i^{out} \cap AS_j^{out}) \wedge (\exists s / s \in AS_i^{out} \wedge s \notin AS_j^{out}) \wedge (\exists s' / s' \notin AS_i^{out} \wedge s' \in AS_j^{out}) \end{cases}$$

Règles :

Cas 1 : $(AS_i^{out} \subset AS_j^{out}) \vee (AS_i^{out} \supset AS_j^{out})$ dans ce cas le service dont les sorties sont incluses dans les sorties de l'autre doit être retiré de la composition puisqu'il n'apporte aucune valeur ajoutée pour la composition. La règle suivante retourne vrai si un service AS_i doit être retiré de l'ensemble E (Ensemble de services candidats à la composition) et faux dans le cas contraire :

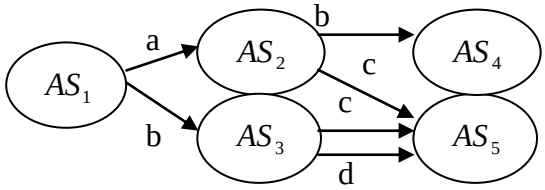
- $R_1(AS_i, E)$: Si $(\exists AS_j \in E / AS_i^{out} \subset AS_j^{out})$ alors $(R_1 \leftarrow vrai)$ Sinon $(R_1 \leftarrow faux)$

Cas 2 : $AS_i^{out} = AS_j^{out}$ dans ce cas les deux services apportent les mêmes valeurs pour la composition, nous retirons alors le service le plus grand en terme de paramètres d'entrées. Ce choix est motivé par le faite qu'un service qui possède plus de paramètres d'entrées aura plus de chance d'être relié à plusieurs autres services dans la composition, donc la suppression de ce service de la composition provoque probablement la suppression d'un certain nombre de services si ces derniers sont reliés uniquement au service supprimé. Dans le cas où les deux services possèdent le même nombre d'entrées, alors nous retirons celui dont l'indice est le plus grand. La règle associée à ce cas est la suivante :

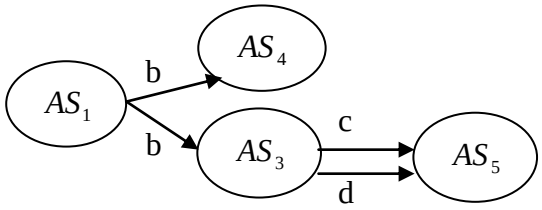
- $R_2(AS_i, E)$:

Si $(\exists AS_j \in E / AS_i^{out} = AS_j^{out}) \wedge ((card(AS_i^{in}) > card(AS_j^{in})) \vee ((card(AS_i^{in}) = card(AS_j^{in})) \wedge (i > j)))$
Alors $(R_2 \leftarrow vrai)$ Sinon $(R_2 \leftarrow faux)$

Cas 3 : $(\exists p / p \in AS_i^{out} \cap AS_j^{out}) \wedge (\exists s / s \in AS_i^{out} \wedge s \notin AS_j^{out}) \wedge (\exists s' / s' \notin AS_i^{out} \wedge s' \in AS_j^{out})$ dans ce cas un service sera retiré de la composition si on peut trouver n'importe quel paramètre de sa sortie dans les autres services. Voici un exemple :



AS_2 est un service en plus dans cette composition, puisque le paramètre b est délivré par AS_1 et le paramètre c est délivré par AS_3. Ainsi le schéma précédent devient :



Pour traiter ce cas, il suffit de chercher l'existence des paramètres de sortie d'un service dans les sorties des autres services d'un ensemble *E* (représente l'ensemble des services d'une même étape de composition), c'est à dire $(\forall p \in AS_i^{out} / (\exists AS_j \in E / p \in AS_j^{out}))$. Il faut connaître également les paramètres qui sont déjà délivrés (ex. le paramètre b de l'exemple précédent). Pour cette raison, nous gardons l'historique des sources de ces paramètres en introduisant la définition suivante :

Définition 6 : Un processus de marquage est un processus qui consiste à attribuer une et une seule source pour chaque paramètre qui participe à la composition.

Paramètre	marque	source
p	$0 \vee 1$	$vide \vee AS_i$

$marque(p) = 0 \Rightarrow source(p) = vide$: Le paramètre p n'est pas utilisé dans la composition ;
$marque(p) = 1 \Rightarrow source(p) = AS_k$: Tel que AS_k est le service abstrait qui a délivré le paramètre p dans la composition.

Procédure $Marquage(AS_i)$:
$\forall p \in AS_i^{out}$, Si $(marque(p) = 0)$ alors
$(marque(p) \leftarrow 1) \wedge (sourcce(p) \leftarrow AS_i)$
$Card\_marquage(AS_i) \leftarrow Card\_marquage(AS_i) + 1$

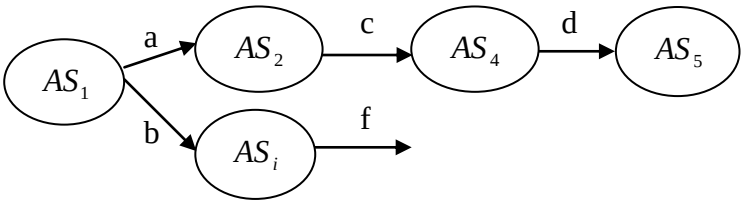
Le processus de marquage est effectué à chaque étape de la composition. Donc si un paramètre p est délivré à une étape antérieure alors sa marque est à 1. la règle suivante stipule alors que si tous les paramètres de sortie d'un service sont déjà délivré (marqué) ou bien ils appartiennent à des services du même niveau E, alors ce service sera retiré de la composition.

- $R_3(AS_i, E)$: Si $(\forall p \in AS_i^{out} / (marque(p) = 1) \vee (\exists AS_j \in E / p \in AS_j^{out}))$ alors $(R_3 \leftarrow vrai)$ Sinon $(R_3 \leftarrow faux)$

Proposition 2 : Le nombre de services obtenus par l'application des règles R_1, R_2 et R_3 est supérieur ou égale au nombre de services nécessaires pour la composition. .

Preuve :
Soient : E_1 L'ensemble de services obtenues après application des règles R_1, R_2 et R_3.
E_2 L'ensemble de services nécessaires pour la composition.

Soit un service $AS_i \in E_2$, si $AS_i \notin E_1$ alors AS_i ne vérifie pas les règles R_1, R_2 et R_3. Or si AS_i ne vérifie pas ces règles alors il est retiré de la composition. Donc $AS_i \notin E_2$ (contradiction). Donc $AS_i \in E_1$, d'ou $E_2 \subseteq E_1$ 1
Soit maintenant un service $AS_i \in E_1$. C'est à dire AS_i vérifie les règles R_1, R_2 et R_3. Mais les sorties de AS_i peuvent ne pas être utilisées à aucun moment de la composition. Voici un contre exemple :



Le service AS_i vérifie les règles mais ses paramètres de sortie ne sont pas nécessaires pour la suite de la composition. Donc $AS_i \notin E_2$ d'ou $E1 \not\subset E_2$ 2
De 1 et 2, il résulte : $E_2 \subseteq E_1$ donc $Card(E_1) \geq Card(E_2)$
D'où la preuve de la proposition 2.

Pour que le nombre total de services obtenus par l'application des règles R_1, R_2 et R_3 soit égale exactement au nombre de services nécessaires pour la composition, nous introduisons la définition suivante :

Définition 7 : Un processus de démarquage est un processus qui consiste à retirer la marque pour un paramètre p si ce dernier ne participe pas à la composition.

Donnant un ensemble *S* des paramètres nécessaires à la composition et un service AS_i. Tous les paramètres *p* pour lesquelles AS_i constitue la source, si p n'est pas dans *S* alors nous retirons la marque de *p* . Voici la procédure de démarquage :

$Démarquage(AS_i, S)$
$\forall p / source(p) = AS_i$, Si $p \notin S$ alors
$(marque(p) \leftarrow 0) \wedge (source(p) \leftarrow vide)$
$Card\_marquage(AS_i) \leftarrow Card\_marquage(AS_i) - 1$

Un service sera donc retiré automatiquement de la composition si ce dernier ne marque aucun paramètre, c'est à dire : $(Card\_marquage(AS_i) = 0)$ d'où la règle suivante :

$R_4(AS_i)$: Si $(Card\_marquage(AS_i) = 0)$ alors ($R_4 \leftarrow vrai$) Sinon ($R_4 \leftarrow faux$)

IV.5.3 Un algorithme pour la composition de services

En s'appuyant sur les quatre règles définies précédemment ainsi que les deux processus de marquage et de démarquage, nous proposons un algorithme de composition qui vérifie la faisabilité de la composition et construit un plan de composition dans le cas où la

composition est faisable. Nous avons appelé cet algorithme FCoSC *(Feasibility & Construction of Service Composition)*

Algorithme FCoSC (T_i)

$S \leftarrow T_i^{in}$, $W \leftarrow AS$, $C \leftarrow \phi$
Faisable $\leftarrow$ vrai, Change $\leftarrow$ faux
$j \leftarrow 1$
Début
$\forall p \in T_i^{in}$, (Marque (p) $\leftarrow 1$) $\wedge$ (Source (p) $\leftarrow T_i$)
Tant que $(\neg(T_i^{out} \subseteq S)) \wedge$ (Faisable) faire

Si ($W = \phi$) alors
Faisable $\leftarrow$ faux
Sinon
Pour k = 1 à N faire
Si ($AS_k^{in} \subseteq S$) alors
$C \leftarrow C \cup \{ AS_k \}$
Change $\leftarrow$ vrai
Fin si
Fin pour
Si ($\neg$ (Change)) alors
Faisable $\leftarrow$ faux
Sinon
$W \leftarrow W - C$
Pour tout ($AS_k \in C$) faire
Si ($R_1(AS_k, C)$) $\vee$ ($R_2(AS_k, C)$) alors
$C \leftarrow C - \{ AS_k \}$
Sinon
Marquage (AS_k)
Si ($R_3(AS_k, C)$) alors
$C \leftarrow C - \{ AS_k \}$
Sinon
$plan\_composition[i][j] \leftarrow AS_k$
$j \leftarrow j + 1$
Fin si
Fin si
Fin pour
$C \leftarrow \phi$
$S \leftarrow \{ p / Marque(p) = 1 \}$
Fin si
Fin si

Fin tant que

// Révision du plan de composition et élimination des services en plus //

$S \leftarrow T^{out}$
$k \leftarrow j-1$
Tant que ($k \geq 1$) $\wedge$ (Faisable) faire
  $as \leftarrow plan\_composition[i][k]$
  Démarquage (as, S)
  Si ($R_4(as)$) alors
    Supprimer la case $plan\_composition[i][k]$ du vecteur $plan\_composition[i][]$
  Sinon
    $S \leftarrow S \cup \{ as^{in} \}$
  Fin si
  $k \leftarrow k-1$
Fin tant que
  Retourner Faisable
Fin

IV.5.4 Principe de l'algorithme

L'algorithme *FCoSC* est un algorithme planificateur qui est représenté par un quadruplet $\langle S, W, T_i^{in}, T_i^{out} \rangle$ tels que :

- S : ensemble de paramètres (état de la composition)
- W : ensemble de services abstraits
- T_i^{in} : état initial
- T_i^{out} : état final (but)

Une étape de la composition est représentée par un ensemble C de services dont les paramètres appartiennent à *S*.

Mise à jour de S et W à une étape i :

$S = S \cup \{ p / p$ est marqué à l'étape i$\}$
$W = W - C$: retirer de W les services dont les paramètres sont vérifiés.

Conditions d'arrêt :

- Si $T_i^{out} \not\subset S$ et $W = \phi$: tout les services sont explorés mais le but n'est pas encore atteint, donc il n'y a pas de solution pour la composition.
- Si C n'est pas modifié après la mise à jour : l'ensemble des paramètres *S* ne satisfait aucun service et le but n'est pas encore atteint, donc pas de solution pour la composition.
- Si $T_i^{out} \subseteq S$: composition trouvée.

IV.5.5 Déroulement de l'algorithme sur un exemple

Supposant un ensemble de services représenté par le tableau *AS* suivant :

Service abstrait	Entrées	Sorties	Description
AS_1	a	b	Pdf converter
AS_2	a	c	Ps converter
AS_3	b	d	Pdf printer
AS_4	b	e	Pdf reader
AS_5	c	f	Ps reader
AS_6	g, h	i	Find restaurant
AS_7	g, i	j	Find direction

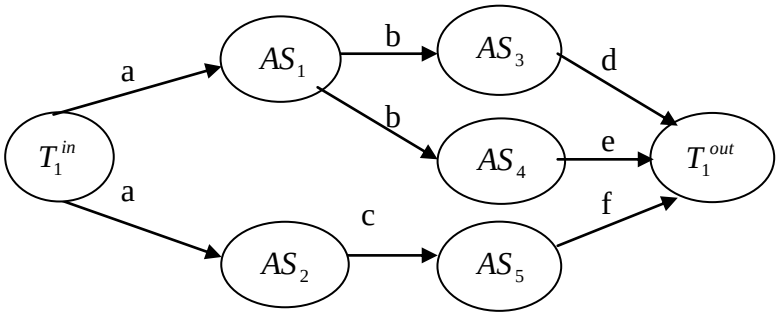
Le tableau des paramètres est le suivant :

Paramètres	Désignation	Description
a	*.doc	Fichier de format doc (Word)
b	*.pdf	Fichier de format pdf
c	*.ps	Fichier de format ps
d	impression_OK	Confirmation de l'impression
e	lecturePDF_OK	Confirmation de la lecture d'un fichier pdf
f	lecturePS_OK	Confirmation de la lecture d'un fichier ps
g	position	Position actuelle de l'utilisateur
h	repas_préféré	Menu préféré
i	@restaurant	Adresse d'un restaurant
j	carte_direction	Carte image indiquant la direction à suivre

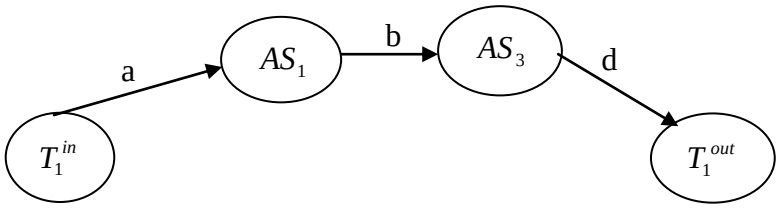
Soit un utilisateur en déplacement désirant lancer l'impression d'un document qui se trouve sur le PC de son bureau, pour le récupérer rapidement à son arrivée au bureau. De plus, comme il est midi, l'utilisateur doit prendre un déjeuner dans le plus proche restaurant. Sa requête est alors interprétée de la manière suivante :

Imprimer (fichier.doc ; impression_OK) $\wedge$ *Chercher_restaurant (position, repas_préféré ; carte_direction)*

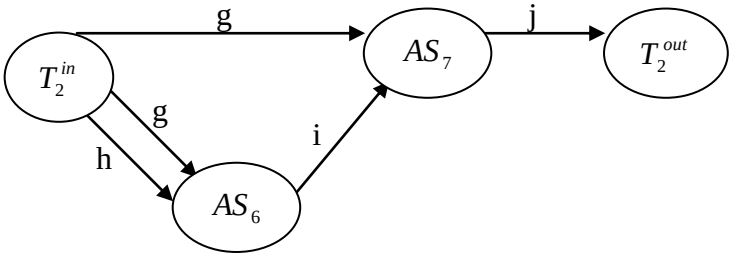
La première tâche (T_1) *: Imprimer (fichier.doc ; impression_OK) :* L'application de l'algorithme *FCoSC* donne le schéma suivant : (avant le processus de démarquage)



L'application du processus de démarquage sur le schéma précédent donne :



La deuxième tâche (T_2) *: Chercher_restaurant (position, repas_préféré ; carte_direction) :* L'application de l'algorithme *FCoSC* donne le schéma suivant : (avant et après le processus de démarquage)



Finalement, le plan de composition pour la requête envoyée est représenté par la matrice *plan_composition*[][]suivante :

Plan de la tâche (T_1)	AS_1	AS_3
Plan de la tâche (T_2)	AS_6	AS_7

Ce plan est maintenant prés à être exécuté afin de rendre la réponse à l'utilisateur. Dans le prochain chapitre, nous verrons comment ce plan sera pris en charge concrètement par le module de sélection et d'exécution de services.

Chapitre V

Approche basée MDP pour la composition de services

V.1 Introduction

Ce chapitre illustre en détail une approche de composition de services qui traite le non déterminisme des services, l'incertitude sur le contexte, et la nature dynamique de l'environnement. Pour se faire, nous introduisons les processus de décision markoviens (MDP) et l'apprentissage Bayésien. En effet, le formalisme MDP permet de prendre en compte le non déterminisme des actions (services) par les probabilités de transition. De plus, la fonction de récompense du MDP permet de guider la sélection vers le service qui satisfait le maximum de contexte. L'approche d'apprentissage Bayésien adapte la solution trouvée aux changements survenus dans l'environnement. Dans ce chapitre, nous commençons par un aperçu sur les processus de décision markoviens. Par la suite, nous formalisons notre problème de composition de services comme un processus de décision markovien en définissant tous les éléments nécessaires. Enfin, nous illustrons la manière d'introduire l'apprentissage Bayésien pour réviser les probabilités initiales, avant de présenter un algorithme qui intègre cet apprentissage dans le processus de sélection des services.

V.2 Processus de décision markoviens (MDP)

Les chaînes de Markov permettent de décrire la dynamique des systèmes stochastiques. Mais elles sont peu adaptées pour la représentation des systèmes dans lesquels, il faut intervenir afin de choisir l'action à exécuter pour influencer l'évolution de ces systèmes. Les processus de décision Markoviens (MDP) permettent de formaliser de tels systèmes. Ils constituent une extension des chaînes de Markov dans lesquelles sont ajoutés un ensemble d'actions et une fonction de récompense.

V.2.1 Les composantes d'un MDP

V.2.1.1 Définition d'un processus de décision markovien (MDP)

Un *processus de décision markovien* est défini par un tuple $\langle S, A, T, R, H \rangle$ dans lequel :

- S : est un ensemble d'états s qui désignent les configurations possibles du monde réel.
- A : est un ensemble d'actions a qui modifient l'état du monde réel.
- $T : S \times A \times S \to [0,1]$: est une fonction de transition. Elle donne la probabilité de passer entre les instants t et $t+1$ de l'état s à l'état s' sachant que l'action choisie est a. Elle est notée : $T(s,a,s') = P(s_{t+1} = s' / s_t = s, a_t = a)$.
- $R : S \times A \times S \to \Re$: est une fonction de récompense (appelée aussi renforcement ou gain.). Elle donne la récompense reçue en passant de l'état s à l'état s' sachant que l'action choisie est a. Elle est notée : $R(s,a,s')$.
- H : est l'horizon du MDP tel que : $0 \prec H \leq \infty$. Il définit une limite temporelle au delà de laquelle ce qui peut arriver n'est pas significatif pour le système. Dans un MDP à horizon fini H, uniquement les H prochaines décisions qui sont considérées.

Les processus de décision Markoviens modélisent des problèmes de décision en environnement stochastique. Dans chaque état, il faut décider de l'action à exécuter afin d'agir sur l'évolution du système. Voici un exemple de transition dans un MDP :

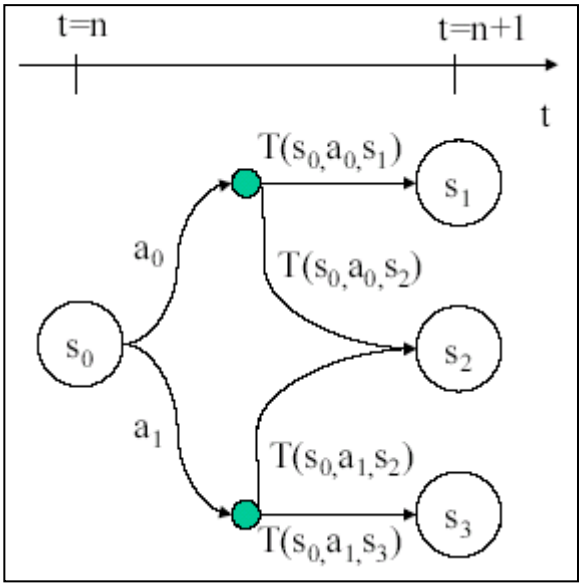


Fig. V.1 : Exemple de graphe de transition d'un MDP

Lorsque le système se trouve dans l'état s_0, il peut accomplir deux actions a_0 ou a_1. Lorsqu'il effectue l'action a_0 , il a une probabilité $T(s_0, a_0, s_1)$ d'arriver dans l'état s_1 et une probabilité $T(s_0, a_0, s_2)$ d'arriver dans l'état s_2. Lorsqu'il effectue l'action a_1, il a une probabilité $T(s_0, a_1, s_2)$ d'arriver dans l'état s_2 et une probabilité $T(s_0, a_1, s_3)$ d'arriver dans l'état s_3.

V.2.1.2 L'ensemble des actions

L'exécution d'une action a à partir d'un état s est stochastique et peut donc mener à différents états s'. Cet aspect probabiliste est représenté par la fonction de transition T. Le système étant markovien, seul l'état courant est nécessaire pour décider quelle action à réaliser. Le formalisme MDP suppose que toute action ait une durée d'une unité de temps. Chaque instant t représente une étape de décisions qui consiste à exécuter une action qui se termine à l'instant $t+1$.

V.2.1.3 La fonction de transition

La fonction de transition d'un MDP formalise la dynamique du système. Elle vérifie la propriété de Markov et tiens compte des actions. Elle se résume alors par l'équation suivante : $P(s_{t+1} = s' / s_0, a_0, s_1, a_1, ..., s_t, a_t) = P(s_{t+1} = s' / s_t, a_t)$.Comme pour les chaînes de Markov, chaque état doit être défini de façon à résumer toute l'information nécessaire à la prédiction de l'évolution du système, et par conséquent à la décision. Lorsque la dynamique du système est indépendante du temps, le système est dit homogène. La probabilité de passer d'un état s à un instant t à un état s' à un instant $t+1$ est indépendante de t, d'où : $P(s'_{t+1} / s_t, a_t) = P(s' / s, a)$ $\forall t$

V.2.1.4 La fonction de récompense

La fonction de récompense d'un MDP permet de représenter les objectifs du système et de guider son comportement. Elle définit les situations préférées ainsi que les comportements non désirés. Elle associe à chaque couple état - action (s,a) une valeur numérique traduisant le fait que l'exécution de l'action a soit souhaitée ou non à partir de l'état s. Il existe d'autres façons de définir une fonction de récompense. Il est possible de récompenser le fait de passer d'un état s à un état s' ($R(s,s')$) ou bien tout simplement le fait d'être dans un état s' ($R(s')$). A chaque décision, le système cherche à maximiser la somme de ses récompenses. Soit r_t la récompense obtenue à l'étape t. Pour des problèmes à horizon fini H, à chaque étape t, le système cherche à maximiser ses H récompenses à venir, soit: $R_t = r_{t+1} + r_{t+2} + ... + r_{t+\mathrm{H}}$. Lorsque l'horizon du problème est infini, un facteur d'atténuation $\gamma \in [0, 1[$ est utilisé afin de permettre au système de calculer une récompense pondérée à long terme. On obtient alors : $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...$. La valeur du facteur d'atténuation influence le comportement du système. Lorsqu' elle est proche de 0, les récompenses obtenues dans un futur lointain seront considérées comme insignifiantes et le système favorisera ses récompenses à court terme. Si elle tend vers 1, le système considérera indifféremment ses récompenses à court terme et à long terme.

Dans ce travail, on se restreindra à des MDP :

- finis (les ensembles S et A sont finis) ;
- homogènes (T et R sont indépendants du temps).
- à horizon fini ou infini;

V.2.2 La résolution d'un MDP

Résoudre un MDP consiste à déterminer un comportement du système qui lui permette d'atteindre ses objectifs. Pour ce faire, il est nécessaire de déterminer une politique d'action du système.

V.2.2.1 La notion de politique

Une politique est une fonction décrivant comment le système doit agir dans chaque situation possible. La politique π fait correspondre à chaque état s du système, une action a à exécuter, d'où : $\pi : S \rightarrow A$. Pour chaque problème de décision formalisé sous forme un MDP, il existe un ensemble Π de politiques possibles $\pi \in \Pi$. Résoudre un MDP consiste à trouver la meilleure politique, c'est-à-dire celle maximisant le critère de performance. Cette solution est appelée politique optimale et est notée π^*. Bellman [5] a démontré que tout MDP possède au moins une politique optimale déterministe et stationnaire. De ce fait, la recherche de la politique peut être réalisée dans l'espace des politiques déterministes. Une politique

déterministe associe à chaque état une et une seule action. Elle s'oppose aux politiques stochastiques (ou stratégies mixtes) associant à chaque état différentes actions possibles suivant une certaine distribution de probabilités.

V.2.2.2 L'évaluation d'une politique

Afin d'établir un classement des politiques et de pouvoir déterminer la meilleure d'entre elles, un critère de performance doit être spécifié. Il permet de définir une mesure d'utilité associant à chaque politique π, une valeur V^{π}. En fonction de leurs valeurs, les politiques pourront alors être classées et la politique optimale sera identifiée. Le critère de performance d'un système consiste à maximiser ses récompenses à long terme. La politique optimale π^* est telle qu'elle maximise l'espérance de récompense à long terme en tout état du système, soit : $\forall \pi \quad \forall s \in S$

$V^{\pi^*}(s) \geq V^{\pi}(s)$

V.2.2.3 Le principe d'optimalité de Bellman

Bellman [5] a montré que la fonction de valeur d'une politique peut être calculée par récurrence, grâce à l'équation de Bellman suivante :

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') * V_t^{\pi}(s').$$

De plus, la fonction de valeur de la politique optimale d'un MDP satisfait l'équation de point fixe suivante : $V_{t+1}^{*}(s) = \max_{a \in A}[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') * V_t^{*}(s')]$.

Un système d'équations peut alors être déduit en appliquant l'équation de Bellman à chaque état. Il a été montré que ce système admet une solution unique à partir de laquelle une politique optimale peut être déduite. Bien que l'équation de Bellman admet une et une seule solution V^*, il peut exister plusieurs politiques optimales si celles-ci ont toutes pour valeur V^*. L'équation de Bellman est à la base de plusieurs algorithmes de résolution des MDPs, en particulier *Value Iteration* que nous présenterons dans la section suivante.

V.2.2.4 L'algorithme Value Iteration

L'équation de Bellman constitue la base de l'algorithme *Value Iteration* (également appelé algorithme d'itération des valeurs), l'un des algorithmes les plus utilisés pour la résolution de MDPs à horizons finis ou infinis.

Algorithme Value Iteration $\langle S, A, T, R, \varepsilon, \gamma \rangle$

Entrées *:* *un MDP* $\langle S, A, T, R \rangle$
un paramètre de précision ε
un facteur d'atténuation γ
une valeur initiale V_0 *arbitraire*

Début
1 $t \leftarrow 0$
2 pour tous $s \in S$ *faire*
3 $V_0(s) \leftarrow V_0$
4 fin pour
5 répéter
6 $t \leftarrow t+1$
7 pour tous $s \in S$ *faire*

8 $$V_t(s) = \max_{a \in A}[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') * V_{t-1}(s')]$$

9 fin pour
10 jusqu'à $\max_{s \in S} \mid V_t(s) - V_{t-1}(s) \mid < \varepsilon$

/ obtenir la politique optimale de la fonction de valeur */*

7 pour tous $s \in S$ *faire*

10 $$\pi^*(s) = \arg\max_{a \in A} \left\{ [\Re(s,a) + \gamma \sum_{s' \in S} T(s,a,s') * V_{t-1}(s')] \right\}$$

9 fin pour
Return π^*
Fin

V.3 Modélisation de la composition de services par un MDP

V.3.1 Définition des éléments du MDP

Un service concret $cs_{i,j}$ est considéré comme étant une action à exécuter notée a. Dans un environnement incertain, lorsqu'une action s'exécute, le système peut avoir deux types de réponses possibles : ***positive*** avec une probabilité p, et ***négative*** avec une probabilité $q = 1 - p$:

- Réponse positive : le service invoqué répond et avec les spécifications demandées. Cet état est souhaitable puisqu'il nous assure la réalisation de la tâche.

- Réponse négative : soit le service invoqué ne répond pas. Plusieurs raisons peuvent être à l'origine de cet état telles que : la panne du réseau, la maintenance du serveur, la suppression du service,…etc. Dans ce cas, l'état du service est inconnu. Ou bien le service invoqué ne fourni pas les fonctionnalités demandées. Il se peut que le service a été modifié ou bien il a changé de contexte. Dans cet état, nous trouvons également les services qui tombent en panne lors de leur exécution.

Nous associons une variable aléatoire booléenne X_i pour chaque service abstrait AS_i tel que :

$$X_i = \begin{cases} 1 \text{ si la réponse du service } cs_{i,j} \text{ invoqué est positive} \\ 0 \text{ si la réponse du service } cs_{i,j} \text{ invoqué est négative} \end{cases}$$

Le problème de composition de services est modélisé par un MDP $\langle S, A, T, R \rangle$ tel que :

- S : ensemble des états du système de composition. Un état s est défini comme étant la valeur de la variable aléatoire X_i associée au service AS_i qui est en cours d'exécution.

- A **:** ensemble des actions $cs_{i,j}$ des services AS_i du plan de composition. $A = \{ AS_1, AS_2, ..., AS_i, ..., AS_n \}$ tels que : $AS_i = \{ cs_{i,1}, cs_{i,2}, ..., cs_{i,k} \}$ $i \in \{1, 2, \ldots, n\}$, $k \in N$. $AS_i \in plan\_composition[][]$.

- T : la probabilité de transition d'un état s à un état s' en effectuant l'action a. Elle est notée : $T(s,a,s') = P(s_{t+1} = s' / s_t = s, a_t = a)$.Ces probabilités traduisent l'estimation de réponse pour chaque service lors de son exécution. Initialement, il est possible d'avoir quelques estimations sur la probabilité de réponse pour chaque service (action). Ces estimations représentent la certitude avec laquelle les demandes vont être satisfaites en se basant sur des connaissances sur les services tels que la disponibilité et la fiabilité.

- R : la récompense souhaitée en exécutant l'action a à partir de l'état s. Elle dépend de la satisfaction du contexte par l'action a. Elle est notée : $R(s,a)$.La récompense porte sur un ensemble de critères de qualité qui satisfassent au mieux le contexte de l'utilisateur, par exemple : le temps de réponse, le coût, la sécurité, les préférences de l'utilisateur, etc. la récompense d'une action a en partant d'un état s est calculée par la formule suivante :

$$R(cs_{i,j}) \leftarrow \sum_k P_k * (\frac{1}{C_{i,k}}) \quad \textit{(voir section } \text{IV.3.3)}$$

Afin de résoudre le MDP pour la composition de services, nous modifions l'algorithme value itération au niveau de la ligne 8, et devient :

$$V_t(s) = \max_{a \in AS_i} [R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{t-1}(s')]$$

pour tous état $s = X_i$

V.3.2 Représentation graphique du MDP

Un graphe MDP pour la composition de service noté MDP-G est un graphe qui relie un état initial à un état final en passant par des nœuds principaux. Chaque nœud principal est relié à un ensemble de nœuds secondaires. La description des éléments du MDP-G est donnée par le tableau suivant :

Elément	Notation graphique	Description
Nœud initial	x_I	Nœud représentant l'état initial T^{in}
Nœud final	x_F	Nœud représentant l'état final T^{out}
Nœud principal	X_i	Nœud représentant l'état d'un service abstrait AS_i. Il est à 1 si le service est réalisé, sinon il est à 0.
Nœud secondaire	$x_{i,j}$	Représente l'action $cs_{i,j}$ à exécuter pour le nœud principal X_i
Arc en flèche	→	Arc entre : nœud initial et un nœud principal, ou bien entre deux nœuds principaux, ou bien entre un nœud principal et nœud final. Un nœud principal doit être à 1 pour pouvoir passer au nœud principal suivant.
Arc en flèche à double sens	(P_j, R_j) ↔	Arc entre un nœud principal X_i et chacun de ces nœuds secondaires $x_{i,j}$. P_j est la probabilité de réponse du nœud $x_{i,j}$, et R_j est la récompense obtenue du nœud $x_{i,j}$

Tableau V.1 : Description des éléments du graphe MDP-G

Exemple : Voici un exemple de graphe MDP-G avec deux nœuds principaux X_1 et X_3 , tels que : X_1 possède quatre nœuds secondaires, et X_3 possède cinq nœuds secondaires.

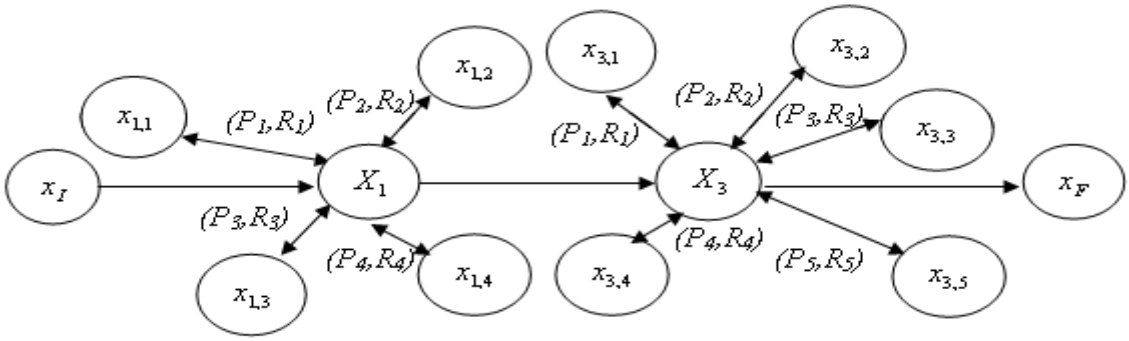


Fig. V.2 : Exemple d'un graphe MDP-G

V.4 Apprentissage Bayésien

La question qui se pose pour le formalisme MDP, est de savoir d'oŭ vient le modèle initial de probabilités ? Au départ, on ignore les lois qui régissent l'environnement. Deux approches pour estimer ces probabilités sont envisagées. Une première source de cette estimation est l'histoire des données anciennes. Une technique d'extrapolation appropriée permet aux estimations d'être projetées sur la situation actuelle. Comme une approche alternative, il s'agit de supposer qu'initialement ces estimations ne sont pas connues et il faudra alors les apprendre petit à petit à travers les interactions avec l'environnement. Cette dernière approche s'appuie sur une méthode de calcule basée sur le modèle d'apprentissage Bayésien pour réviser les estimations initiales.

Initialement, il est possible d'avoir quelques estimations sur la probabilité de réponse pour chaque service (actions) en se basant sur des connaissances sur les services tels que la disponibilité et la fiabilité. Les estimations mesurent le non déterminisme des services invoqués. Elles se manifestent dans la fonction de transition T du MDP. La politique optimale π^* est dépendante de ces estimations. Par conséquent, l'action optimale change quand les probabilités sont modifiées.

Dans le modèle d'apprentissage Bayésien, quand les estimations initiales sont complètement inconnues, des probabilités égales sont attribuées pour chaque réponse de l'invocation d'un service. Par la suite, le modèle essaie de les apprendre à travers les interactions avec l'environnement. Les réponses obtenues par l'invocation des services sont utilisées pour mettre à jour ces probabilités. De cette manière, l'exécution des actions et l'apprentissage sont intercalés.

L'algorithme d'apprentissage Bayésien [69] maintien un conteur d'expériences noté *exper* initialisé à 1 pour chaque état du système. L'exécution d'une action a fait passer le système à l'une des réponses possibles : négatif auquel cas le système reste dans le même état, ou positif auquel cas le système passe à un nouvel état. L'expérience associée à l'état obtenu est incrémentée de 1. Tandis que les autres probabilités sont mises à jour afin de maintenir la somme de ces probabilités à 1. Soit l'exemple suivant :

$T(s,a,s')$:La probabilité de passage de l'état s à l'état s' avec l'action a (réponse positive).
$T(s,a,s)$: La probabilité de rester dans l'état s en exécutant l'action a (réponse négative).
Supposant que lors de l'exécution de a, nous avons eu une réponse positive. Les probabilités T de passage sont alors mises à jour en probabilités T' de la manière suivante :

$$T'(s,a,s') = \frac{[T(s,a,s') \times \exp er] + 1}{\exp er + 1} \quad \textit{(Réponse positive)}$$

$$T'(s,a,s) = \frac{T(s,a,s) \times \exp er}{\exp er + 1} \quad \textit{(Réponse négative)}$$

Nous pouvons vérifier que $T'(s,a,s') + T'(s,a,s) = 1$

V.5 Algorithme de sélection de services basé MDP

V.5.1 Présentation de l'algorithme

Cet algorithme se base sur le graphe MDP-G défini précédemment ainsi que l'apprentissage Bayésien sur les probabilités de réponse des services. Nous appelons cet algorithme *MDP-SCWL* (*MDP Service Composition With Learn).* MDP-SCWL exécute d'abord l'algorithme *value iteration* afin d'obtenir la politique optimale. Ensuite, il intercale l'exécution des actions et l'apprentissage sur leurs probabilités de réponse. Lorsqu'une action ne répond pas, l'algorithme effectue un apprentissage Bayésien et cherche à la remplacer immédiatement par la meilleure action de l'état en cours. En effet, d'après l'algorithme *value iteration,* la valeur V d'un état s dépend des valeurs V' des états suivants s'. Donc, le changement d'une action de l'état en cours n'affecte pas les actions des états restants. L'avantage de cet algorithme est d'exécuter une seule fois l'algorithme *value iteration,* donc un gain considérable en terme de temps de réponse à l'utilisateur. De plus, une fois l'apprentissage est effectué, l'algorithme peut donner une réponse en temps réel à l'utilisateur, puisque le remplacement des actions défaillantes est effectué immédiatement. Enfin, comme l'algorithme intercale la sélection et l'exécution, il est facile de savoir si les précondition d'un service sont satisfaits ou non en se basant sur les effets des services déjà exécutés.

Algorithme MDP_SCWL (MDP-G)

Extraire les éléments S, A, T, R du graphe MDP-G

$\pi^* \leftarrow$ *Value Iteration* $\langle S, A, T, R, \varepsilon, \gamma \rangle$ // π^* *la politique optimale*

μ *// Mesure la distance entre deux probabilités*

σ *// Erreur de l'estimation de probabilités*

Début

Répéter // *boucle d'apprentissage*

  $s \leftarrow$ l'état suivant de l'état initial x_I dans le graphe MDP-G.

  Tant que $s \neq x_F$ faire

    $a \leftarrow \pi^*(s)$ // Une action optimale a

    Exécuter a

    Mettre à jour $T(s,a,s')$ par $T'(s,a,s')$ en utilisant l'apprentissage Bayésien

    Si réponse (a) est négative alors

      $$\pi^*(s) = \arg\max_{x \in s}[R(s,x) * T'(s,x,s')]$$

    Sinon

      $s \leftarrow s'$ // s' est l'état suivant de l'état s dans le graphe MDP-G.

    Fin si

  Fin tant que

Jusqu'à ($\left\| \mu(T,T') \right\|_\infty < \sigma$)

Fin

V.5.2 Fonctionnement des modules de composition et de sélection de services

Pour donner une réponse finale à la requête, les modules de composition et de sélection de services se basent sur les algorithmes *SDT*, *FCoSC* et *MDP-SCWL* en interagissant avec le demandeur du service. Dans le cas où une tâche de la requête ne peut pas être satisfaite, le module de composition de service interagit avec le demandeur de service en lui demandant s'il peut annuler cette tâche et passer aux tâches restantes ou non. Si le demandeur accepte de l'annuler alors le module passe aux autres tâches, sinon la requête est annulée puisque cette tâche est jugée (par le demandeur) importante pour toute la requête.

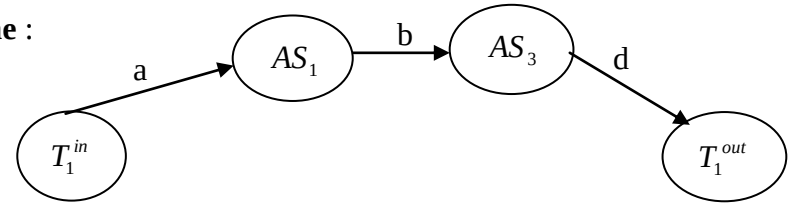
Algorithme REQUEST_RESPONSE

M : le nombre de tâches de la requête
$requete\_refusée \leftarrow faux$ // indique si la requête est acceptée ou non.
$i \leftarrow 1$ // compteur de tâches
Début
// Cette partie est exécutée par le module de découverte et de composition de services
Tant que $(i \leq M) \wedge (\neg(requete\_refusée))$ faire
  Si $(\neg(SDT(T_i))) \wedge (\neg(FCoSC(T_i)))$ alors
    Demander au « demandeur de service » s'il veut annuler la tâche T_i et exécuter le reste
    Si réponse = « non » alors
      $requete\_refusée \leftarrow vrai$
    Sinon
      $i \leftarrow i+1$
    Fin si
  Sinon
    $i \leftarrow i+1$
  Fin si
Fin tant que
// Cette partie est exécutée par le module de sélection et d'exécution de services
Si $(\neg(requete\_refusée))$ alors
  Pour i = 1 à M faire
    $MDP-G \leftarrow$ le graphe de la tâche T_i telle que T_i n'est pas annulée
    $MDP\_SCWL(MDP-G)$
  Fin pour
Sinon
  Afficher message « la requête ne peut pas être satisfaite »
Fin si
Fin

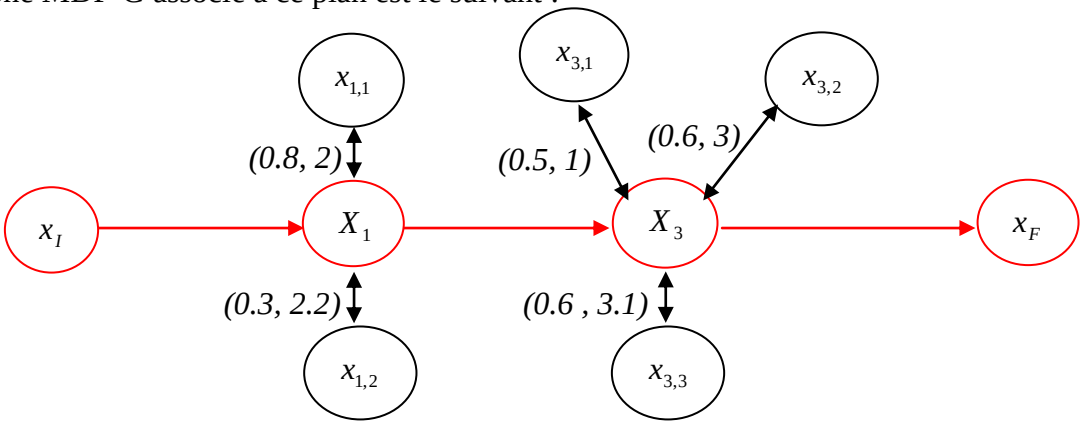
V.5.3 Exemple d'application de l'algorithme

Nous reprenons les schémas de l'exemple précèdent (chapitre IV) pour les représenter sous forme de graphes MDP-G :

Plan de la première tâche :

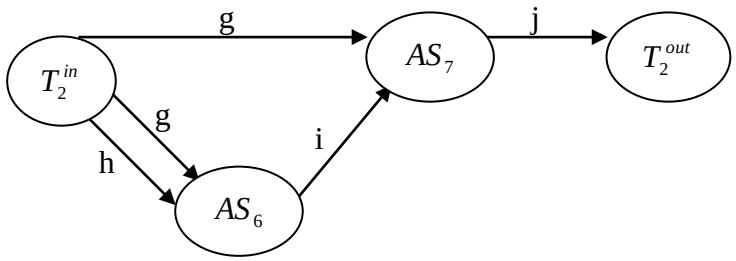


Le graphe MDP-G associé à ce plan est le suivant :

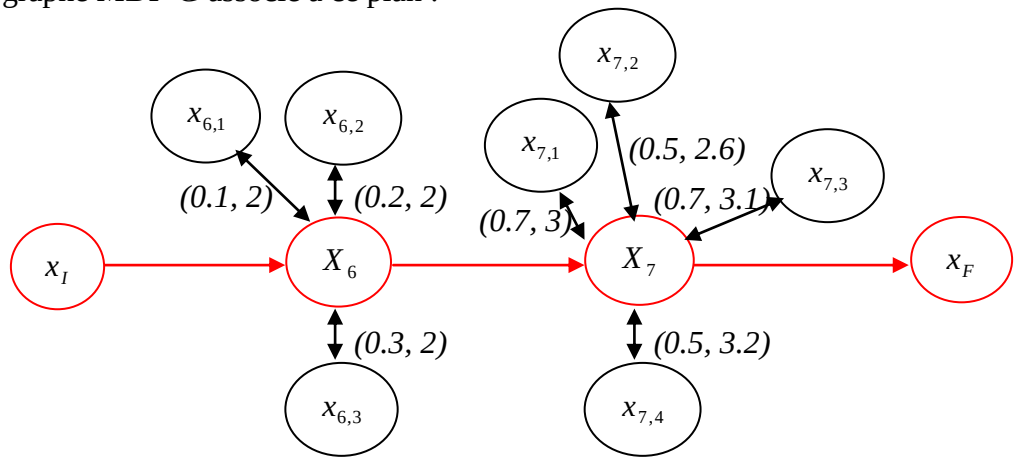


Les nœuds en rouge représentent le plan de composition partant de l'état initial à l'état final. Les nœuds $x_{1,1}$ et $x_{3,3}$ représentent les meilleures actions données par la politique π^* pour les états X_1 et X_3 respectivement . Le nombre d'actions (services concrets), leurs probabilités et leurs récompenses sont choisis aléatoirement juste à titre illustratif. Par exemple pour le service abstrait AS_1, représenté par le nœud X_1, possède deux services concrets : *Standard Pdf Converter* et *Professional Pdf Converter* qui sont représentés par les nœuds $x_{1,1}$ et $x_{1,2}$ respectivement.

Plan de la deuxième tâche :



Soit le graphe MDP-G associé à ce plan :



Afin de prendre en compte les services qui peuvent s'exécuter en parallèle et donner plus de coopération, de distribution et d'intelligence au modèle de composition de services, nous proposons un modèle de composition basé sur la notion d'agent. Ceci constitue l'objet du chapitre suivant.

Chapitre VI

Approche basée agents pour la composition de services

VI.1 Introduction

Dans ce chapitre, nous présentons une approche de composition de services qui se base sur le paradigme d'agents. Il s'agit de faire coopérer un ensemble d'agents afin d'atteindre l'objectif assigné à la composition de service. Nous commençons ce chapitre par présenter la notion d'agent et de systèmes multi agents (SMA). Nous définissons par la suite les agents qui participent à la composition de service et l'organisation de leur communication. Enfin, nous présentons les algorithmes de fonctionnement de chaque agent.

VI.2 La notion d'agent

VI.2.1 Définition

D'après le dictionnaire « Le petit Robert », le mot agent signifie :
« - L'être qui agit.
- Ce qui agit, opère ; force, corps, substance intervenant dans la production de certains phénomènes. »
Le terme « agir » provient du latin *agere* et signifie selon le dictionnaire « Le petit Larousse » : « *faire quelque chose, s'occuper, produire un effet* ».

Ferber [21] propose une définition de la notion d'agent dans laquelle nous retrouvons les concepts clefs d'un agent :
« On appelle agent une entité physique ou virtuelle :
– qui est capable d'agir dans un environnement,
– qui peut communiquer directement avec d'autres agents,
– qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
– qui possède des ressources propres,
– qui est capable de percevoir (mais de manière limitée) son environnement,
– qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
– qui possède des compétences et offre des services,
– qui peut éventuellement se reproduire,
– dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit. »
Cette définition, si elle n'est pas universelle (il existe d'autres définitions du mot agent), a l'avantage de présenter une grande partie des caractéristiques dont peut être doté un agent.

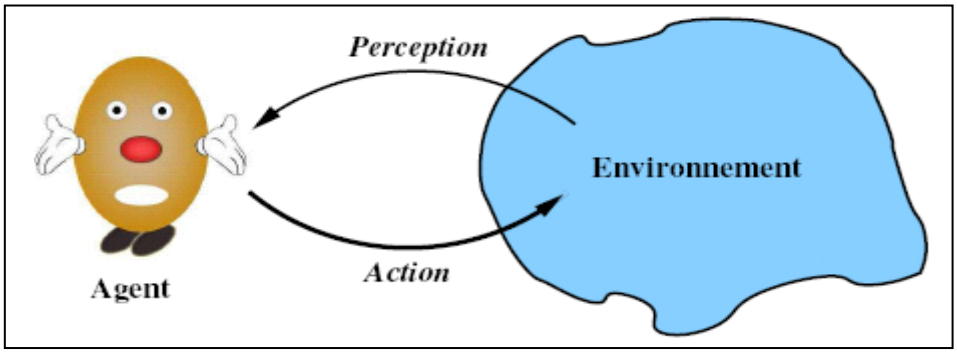


Fig. VI.1 : Agent et environnement

VI.2.2 Caractéristiques

Il s'agit ici d'une liste non exhaustive des caractéristiques d'un agent, d'autres peuvent s'ajouter et se combiner :

- **L'autonomie :** considérons un agent placé dans un environnement et devant décider, à partir de ses perceptions, quelle action exécuter. Si cette décision est prise sans l'intervention d'un tiers, l'agent est dit autonome.
- **L'interactivité** : l'agent doit pouvoir exercer des actions sur son environnement et réciproquement.
- **La réactivité** : c'est la capacité d'un agent à percevoir son environnement (qui pourra être l'utilisateur au travers une interface graphique, un ensemble d'agents, …) et à décider comment agir en respectant le temps qui lui est imparti.
- **La capacité d'apprendre** : un agent doit avoir la capacité d'apprendre afin d'acquérir des connaissances, des informations ou des habitudes.
- **La capacité sociale** : les agents interagissent avec les autres agents (et éventuellement des êtres humains) grâce à des langages de communication entre agents. Cette capacité sera la base pour la coopération entre les agents.
- **La coordination :** l'agent est capable de coordonner ses actions par rapport à un utilisateur ou un autre agent.
- **La compétition :** un agent est capable d'agir dans un environnement où d'autres agents interviennent. Le but est le même pour tous les agents présents, mais un seul l'atteindra. Les autres échoueront et, forcément, tous les coups sont permis.
- **La rationalité** : un agent rationnel est un agent qui prend à tout moment la meilleure décision. Nous pouvons alors nous interroger sur ce qu'on entend par « la meilleure décision ». Mathématiquement, la recherche de ce qui est « meilleur » ou « optimal », se traduit par l'optimisation d'une fonction caractérisant les objectifs de l'agent. cette fonction est appelée fonction d'utilité. Elle définit des préférences sur les états. Elle permet donc d'associer à chaque état *s* un nombre réel décrivant le degré de satisfaction de l'agent lorsqu'il est dans *s*.

VI.3 Système Multi-agents

VI.3.1 Définition

De la même façon qu'il existe un grand nombre de définitions du terme d'agent, plusieurs définitions des systèmes multi-agents (SMA) ont été données. Toutes ces définitions s'organisent autour des idées de multiplicité des entités et d'interaction, comme le montre la définition de Wooldridge [76] *« Un système multi-agent est constitué d'agents interagissant entre eux »*.

L'observation du monde qui nous entoure permet de constater que tout système est composé d'un ensemble d'entités individuelles interagissant ensemble. Dans la nature, les systèmes multi-agents les plus visibles sont les sociétés animales : fourmis, hommes, moutons, voire les associations animales ou végétales qui, par leur action commune, créent un comportement de groupe ou plus simplement coopératif. Un comportement intelligent

émergent alors de la combinaison de comportements simples. C'est cette idée qui est exploitée par les SMA pour résoudre des problèmes plus complexes.

L'étude du comportement d'un SMA dans son environnement consiste à examiner les éléments suivants : les interactions avec l'environnement, la communication entre agents, et l'organisation de la société des agents.

VI.3.2 Exemple

Les fourmis sont un exemple assez courant de système multi-agents. Ces charmants insectes ont la capacité de trouver le chemin le plus court entre deux points sans avoir a priori de notion de distance. Les données du problème de base dans le cas des fourmis est que le chemin cherché doit relier la fourmilière à une source de nourriture, deux chemins de longueurs différentes étant envisageables, mais les fourmis n'ayant aucune mesure ni notion de distance. Les fourmis partant de la fourmilière vont prendre indifféremment l'un ou l'autre chemin, à l'aller comme au retour. Sauf que, au retour, ces dernières déposeront sur le chemin choisi une trace chimique (des phéromones), qui aura pour effet d'inciter les autres fourmis à suivre la piste marquée. Et les dépôts successifs auront pour effet de renforcer ce marquage. Or, le chemin le plus court étant aussi le plus rapide, donc il sera vite fait marqué que l'autre voie. Les fourmis étant d'autant plus attirées par un chemin fortement marqué de phéromones, et le phénomène va en s'amplifiant.

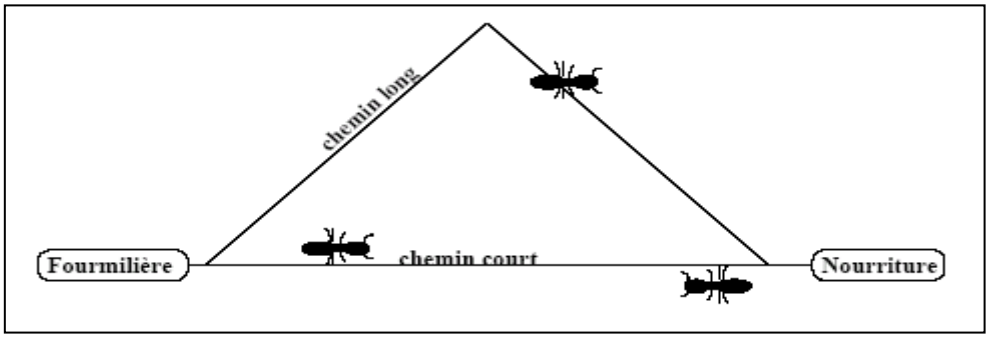


Fig. VI.2 : Recherche du chemin le plus court chez les fourmis

VI.3.3 Interaction dans les SMA

Dans les systèmes multi-agents, toute action d'un agent peut influencer les autres agents. Ferber [21] définit l'interaction comme *« une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques »*. Cette définition permet à Ferber d'établir une classification des types d'interaction selon la nature des buts, les relations aux ressources et les compétences des agents :

- **Indifférence** : les agents sont complètement indépendants, leurs buts sont compatibles, les ressources et les compétences suffisantes. Chaque agent peut donc atteindre ses buts indépendamment des autres agents.
- **Coopération** : lorsque les buts des agents sont compatibles mais que les ressources ou les compétences d'un ou plusieurs agents sont insuffisantes, les agents doivent coopérer afin d'atteindre leurs buts.
- **Antagonisme** : les buts des agents sont incompatibles. Si les ressources sont suffisantes, les agents se trouvent en situation de compétition. Sinon, la situation d'interaction devient une situation de conflit.

VI.3.4 Communication dans les SMA

La communication constitue un moyen fondamental de mise en oeuvre des méthodes de coopération telles que la coordination ou la collaboration. La communication caractérise tout échange d'information entre des agents. Sans communication, les agents se trouvent isolés les uns des autres et aucune coopération n'est possible. Goldman et Zilberstein [28] ont répertorié trois types de communications possibles entre agents suivant le mode de transmission du message :

- **La communication directe** : consiste à envoyer directement un message à un ou plusieurs agents. Lorsque le message est envoyé à un seul agent, on parle de communication point à point. Si l'information est envoyée à tous les autres agents, le message est envoyé en mode diffusion (ou *broadcasting*). Dans ce type de communication, le canal de transmission peut être un réseau informatique, un câble téléphonique, des ondes hertziennes, ...

- **La communication indirecte** : réalisée par modification de l'environnement ou par manipulation de connaissances communes. Les agents peuvent laisser des traces dans l'environnement afin d'indiquer aux autres quelle action a été exécutée, quel est leur état, etc. Lorsque les agents ont tous accès à une base de données commune, la communication peut s'effectuer par modification des connaissances stockées dans la base.

- **Observation commune de caractéristiques de l'environnement** : consiste à observer des caractéristiques de l'environnement sur lesquelles les agents n'ont aucune influence. Ces caractéristiques doivent être observables par tous les agents. Suivant les observations réalisées, les agents vont alors décider de la manière de se coordonner, d'agir. Considérons deux agents devant se déplacer dans une même direction, chaque agent ne pouvant pas observer le sens de déplacement de l'autre. Il est alors possible pour les agents de décider de se déplacer vers le nord lorsqu'il pleut et sinon se déplacer vers le sud. Dans ce cas, la caractéristique de l'environnement, observable par les deux agents, est la météo.

L'étude des langages de communication a fait l'objet de nombreuses recherches en intelligence artificielle ce qui a permis de définir des langages structurés d'échange d'information tels que KQML [22] ou FIPA-ACL [23]. Cependant, dans la suite de notre travail, nous n'aborderons pas la question du langage de communication utilisé par les agents. Nous supposerons qu'ils utilisent un même langage connu de tous.

VI.3.5 Organisation des interactions dans les SMA

Les différents types d'organisation rencontrés dans les systèmes multi-agents sont généralement inspirés des formes d'organisation humaines sociales ou politiques, ou bien des organisations biologiques. Les organisations hiérarchiques [24], prototypes des organisations militaires ou d'entreprise, sont les modèles d'organisation les plus connus. Dans ce modèle, les agents sont structurés en niveaux. Les agents d'un niveau donné assurent les fonctions de décision pour le niveau inférieur auquel ils envoient des ordres. Les réseaux contractuels présentés par Smith [68] structurent les interactions sous la forme d'appels d'offres et permettent de procéder à l'allocation de tâches. L'ensemble des agents est composé d'administrateurs gérant les appels d'offres, et d'offrants élaborant des propositions et exécutant les tâches. Il est possible de greffer une structure hiérarchique à ce type

d'organisation en élaborant des niveaux d'appels d'offres. Différents niveaux d'administrateurs sont alors créés.

VI.4 Les agents dans le cadre de notre travail

VI.4.1 Définition des agents qui participent à la composition

Nous définissons quatre types d'agents pour la composition de services sensibles au contexte :

- **Agent assistant** : assiste l'utilisateur dans ses activités quotidiennes et ses interactions avec l'environnement. Il formule des requêtes de services vers l'agent coordinateur afin de répondre aux besoins de l'utilisateur.

- **Agent contexte** : sauvegarde le contexte de l'utilisateur, et se charge de le transmettre à l'agent coordinateur afin qu'il soit exploité par les services.

- **Agent coordinateur** : un agent coordinateur Ag_c reçoit d'abord le contexte de l'agent contexte, puis la requête de l'agent assistant. Il essaie ensuite de répondre à cette requête en se basant sur les algorithmes *SDT* et *FCoSC* pour la découverte et la construction du plan de composition de services. Une fois le plan est construit, il envoie un signal à tous les agents de services concernés (ceux qui appartiennent au plan de composition). Ce signal contient la table de marquage obtenue par l'algorithme FCoSC, ainsi que les préférences de l'utilisateur. Les paramètres initiaux (contexte et entrées de la requête) sont envoyés aux agents fils de l'agent coordinateur afin d'initialiser le processus de composition.

- **Agent service** : à chaque service abstrait (AS_i) est associé un agent de service Ag_i qui se charge de sélectionner et d'exécuter le meilleur service concrets qui réalise ce service abstrait. Les services concrets représentent les actions dont dispose l'agent. Lorsqu'un agent service reçoit un signal de l'agent coordinateur, il cherche d'abord ses pères et ses fils sur la table de marquage. Ensuite, il évalue la récompense de ses actions (services) en se basant sur les préférences de l'utilisateur. Enfin, l'agent exécute le service qui maximise la récompense et la probabilité de réponse. Il s'agit en faite de trouver une action a tel que :

$$a = \arg \underset{cs \in AS_i}{Max}(P * R)$$

VI.4.2 Organisation de la communication inter agents de composition

La communication entre les agents s'effectue d'une manière directe dans une organisation hiérarchique circulaire comme le montre le schéma ci-dessous (Fig. VI.3). Chaque agent possède des pères et des fils sous forme circulaire. Le point de départ et d'arrivée du cercle est l'agent coordinateur qui initialise le processus de composition.

La table de marquage construite par l'algorithme *FCoSC* joue un rôle important pour trouver les pères et les fils d'un agent. Un agent peut avoir un ou plusieurs pères puisque ses entrées peuvent avoir une ou plusieurs sources. De même un agent peut avoir un ou plusieurs fils puisqu'il peut être la source des entrées d'un ou plusieurs agents. Les deux procédures qui cherchent respectivement les pères et les fils d'un agent i sont présentées dans la section VI.4.3.

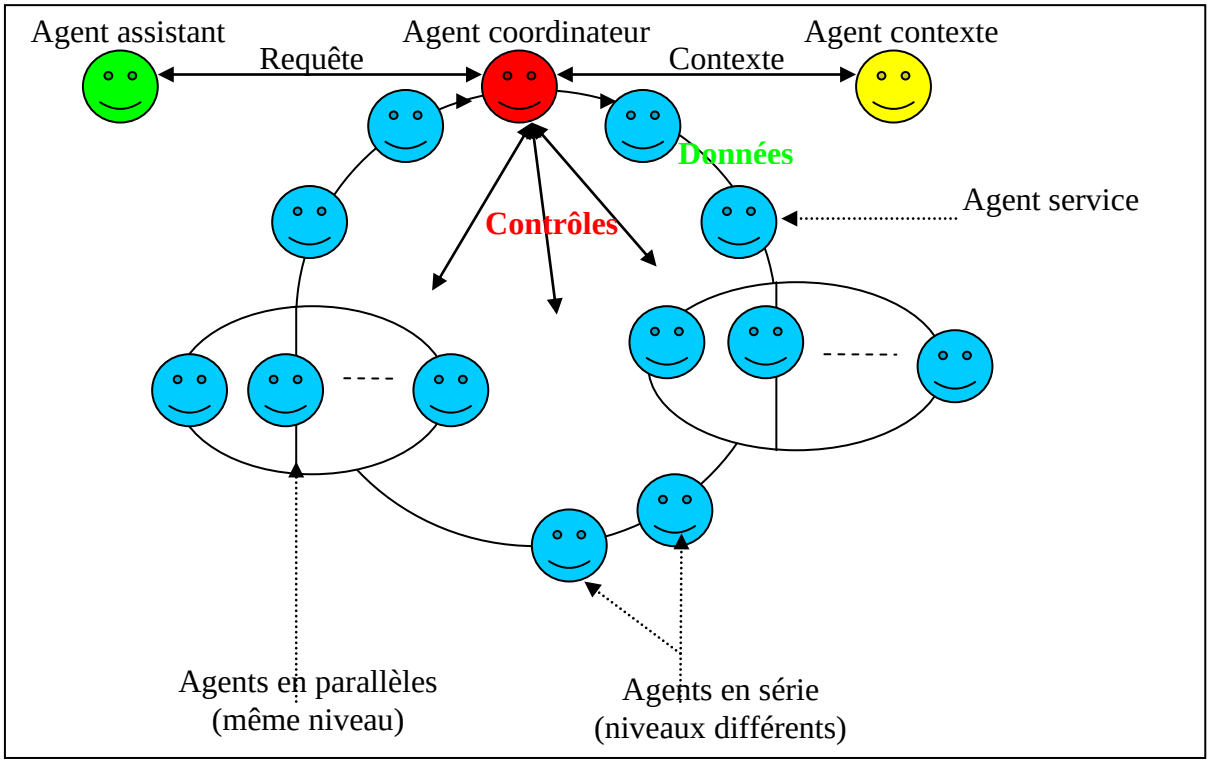


Fig. VI.3 : Organisation des agents de composition

Contrôles : table de marquage, les préférences de l'utilisateur.
Données : entrées, préconditions, sorties, effets.

VI.4.3 Algorithmes de fonctionnement des agents de composition

Les deux procédures de recherche des pères et des fils d'un agent i :

Procédure $Pères(AS_i)$:
$Pères \leftarrow \phi$
Pour tout $p \in AS_i^{in}$
  Chercher la source de p dans la table de marquage
  $Pères \leftarrow Pères \cup \{ source(p) \}$
Fin pour

Procédure $Fils(AS_i)$
$Fils \leftarrow \phi$
Pour tout $AS_j \in plan\_composition[][]$
  Si (AS_i match totalement ou partiellement AS_j) alors
    $Fils \leftarrow Fils \cup \{ AS_j \}$
  Fin si
Fin pour

Algorithme de l'agent coordinateur (Ag_c)

M : le nombre de tâches de la requête
$requete\_refusée \leftarrow faux$ // indique si la requête est acceptée ou non.
$i \leftarrow 1$ // compteur de tâches
Début
Tant que $(i \leq M) \wedge (\neg(requete\_refusée))$ faire
  Si $(\neg(SDT(T_i))) \wedge (\neg(FCoSC(T_i)))$ alors
    Demander à l'utilisateur s'il veut annuler la tâche T_i et exécuter le reste
    Si réponse = « non » alors
      $requete\_refusée \leftarrow vrai$
    Sinon
      $i \leftarrow i+1$
    Fin si
  Sinon
    $i \leftarrow i+1$
  Fin si
Fin tant que
$i \leftarrow 1$
Tant que $(i \leq M) \wedge (\neg(requete\_refusée))$ faire
  Envoyer signal à tous les agents service du $plan\_composition[i][]$
  Envoyer T_i^{in} à $Fils(T_i)$
  Attendre la réponse des agents services pendant un délai t_1
    Si (t_1 est expiré) $\vee$ (réponse d'agent = négative) alors
      Demander à l'utilisateur s'il veut annuler la tâche T_i et exécuter le reste
      Si réponse = « non » alors
        $requete\_refusée \leftarrow vrai$
      Sinon
        $i \leftarrow i+1$
      Fin si
    Sinon
      $i \leftarrow i+1$
    Fin si
Fin tant que

Si $(requete\_refusée)$ alors
  Afficher message « la requête ne peut pas être satisfaite »
Fin si

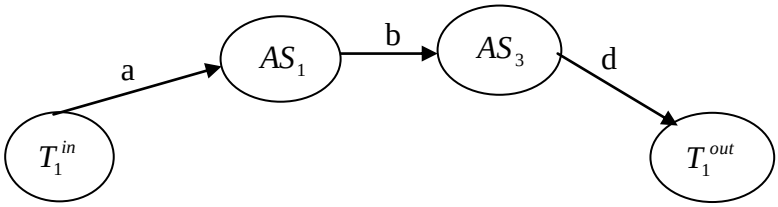
Fin

Algorithme de l'agent de service i ***(Ag_i)***

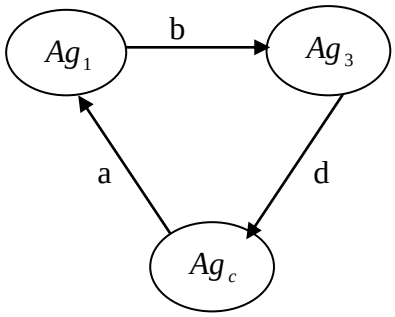
Début
Si *signal_reçu* de l'agent coordinateur alors
  Calculer $Pères(AS_i)$ et $Fils(AS_i)$
  Calculer la récompense de toutes les actions de AS_i
  Attendre les données de tous les $Pères(AS_i)$
  réponse ← négative
  // chercher la meilleur action pendant un certain délai t_2
  Tant que (réponse = négative) ∧ (t_2 n'est pas expiré) faire
    $a = \arg\underset{cs \in AS_i}{Max}(P * R)$
    Exécuter a
    réponse = réponse(a)
    Apprentissage Bayésien
  Fin tant que
  Si (t_2 est expiré) alors
    Envoyer réponse négative à l'agent coordinateur
  Sinon
    Envoyer $effets(a)$ à tous les $Fils(AS_i)$
  Fin si
Fin si
Fin

VI.4.4 Exemple d'application

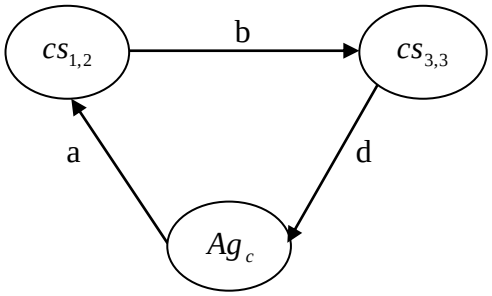
Nous reprenons toujours l'exemple précédent : nous avons le plan suivant de la première tâche obtenu par l'agent coordinateur après l'application de l'algorithme *FCoSC* :



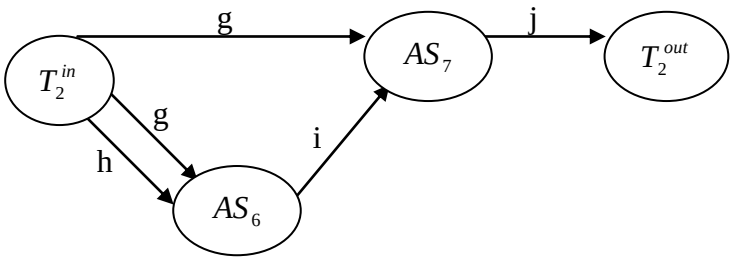
L'agent coordinateur (Ag_c) envoie un signal à tous les agent concernés par la composition (Ag_1, Ag_3). Ces trois agents effectuent alors la recherche de leurs pères et fils et se réorganisent selon le schéma suivant :



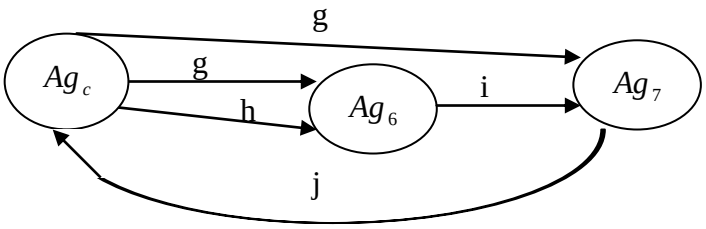
Les agents sélectionnent les meilleurs services concrets pour la composition de services. Le plan précèdent devient :



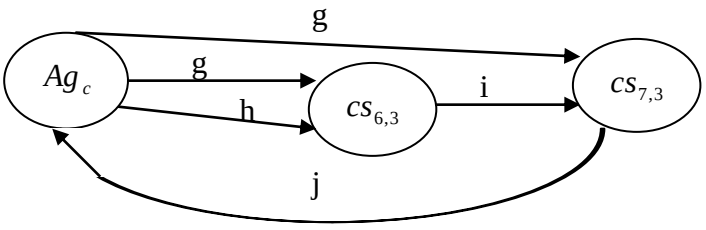
De même, le plan de la deuxième tâche :



L'organisation des agents de composition :



Les services concrets sélectionnés par les agents :



Afin de montrer l'application concrète de toutes les approches proposées pour la composition de services, nous proposons dans le prochain chapitre une étude de cas réelle qui porte sur les services pour l'assistance et la surveillance d'une personne dépendante. Nous montrons également quelques résultats des tests d'implémentation.

Chapitre VII

Implémentation et validation

VII.1 Introduction

Afin de valider notre modèle de composition de services sensibles au contexte, nous proposons d'abord un scénario qui concerne la mise en oeuvre de services pour l'assistance et la surveillance à domicile d'une personne dépendante. Il s'agit particulièrement de surveiller les mouvements de la personne et de lever le doute en cas de chute ou malaise. Cela permet d'anticiper et réduire les risques d'accidents liés au vieillissement et d'effectuer une prévention médicale suite à une évolution de l'état de fragilité.

Nous montrons également dans ce chapitre un ensemble de résultats des testes effectués sur l'algorithme *MDP-SCWL*.

VII.2 Services pour l'assistance et la surveillance d'une personne dépendante

VII.2.1 Système de capture

Le domicile de la personne dépendante est muni d'un système de capture qui est représenté sur la figure suivante :

Fig. VII.1 : Système de capteur du domicile de la personne dépendante

Capteur	Désignation	Description
●	Analyse ambiance	Température, humidité, luminosité
●	Présence (position)	Détecteur de position de la personne (peut être aussi porté par la personne)
●	Consommation d'eau	Détecteur de la consommation d'eau
●	Consommation électrique	Détecteur de la consommation électrique

	Caméra	Flux vidéo
	Accéléromètre	Détecteur de mouvement
	Ouverture porte / fenêtre	Ouverture des portes et fenêtres
	Ouverture meuble	Manipulation des meubles
	Activité cardiaque	Capteur de l'activité cardiaque de la personne (porté par la personne)
	Robot mobile	Robot mobile équipé d'une caméra de surveillance

Tableau VII.1 : Les capteurs et leur désignation

VII.2.2 Services sensibles au contexte

Les données collectées par le système de capture alimentent un ensemble de services sensible au contexte que nous envisageons afin d'analyser et d'interpréter les données contextuelles et d'agir selon le contexte de la personne. Les deux tableaux suivants décrivent les services sensibles au contexte ainsi que leur paramètres d'entrées / sorties :

Service abstrait	Entrées	Sorties	Description
AS_1	a	b	Service localisation
AS_2	c	d	Analyse ambiance
AS_3	e	f	Analyse mouvement
AS_4	g	h	Analyse de l'activité cardiaque
AS_5	b, q	i	Surveillance des mouvements robot
AS_6	b, i	j	Service caméra
AS_7	k	l	Mesure consommation électrique
AS_8	m	n	Mesure consommation d'eau
AS_9	o	p	Agenda des soins
AS_{10}	f, h, p	q	Surveillance de l'état de la personne
AS_{11}	q	r	Notification

Tableau VII.2 : Les services sensibles au contexte

Paramètres	Désignation	Description
A	Signal position	Signal indiquant la position de la personne
B	Localisation	Signal de position transformé (ex. Salon)
C	Vecteurs donnés ambiants	Température, luminosité, humidité
D	Vecteur ambiant analysé	Données analysées (ex. température élevée)
E	Signal accéléromètre	Signal des mouvements de la personne

F	Analyse mouvement	Mouvement interprété (ex. personne tombée)
G	Signal cardiaque	Signal de l'activité cardiaque de la personne
H	Analyse cardiaque	Activité cardiaque analysée
I	Position robot	Position actuelle du robot
J	Flux vidéo	Flux vidéo transmis par la caméra
K	Vecteur consommation élect.	Consommation en élect. des ≠ dispositifs
L	Total Consommation élect.	Total Consommation en élect.
M	Vecteur consommation eau	Consommation en eau des ≠ dispositifs
N	Total Consommation eau	Total Consommation en eau
O	Vecteur de soins	Historique des soins de la personne
P	Vecteur de soins analysé	Les soins prévus (ex. visites)
Q	Etat personne	Etat de santé de la personne (ex. bon, grave)
R	Notification	Signal indiquant une situation

Tableau VII.3 : Les paramètres d'entrées/sorties

VII.2.3 Système de composition de services

Notre système de composition de services sensibles au contexte reçoit les données à partir du système de capture, puis effectue la composition afin de sélectionner les services appropriés pour la situation de la personne et communiquer avec des prestataires de services externes pour une éventuelle intervention. Le système de composition occupe une place centrale comme le montre la figure ci-dessous :

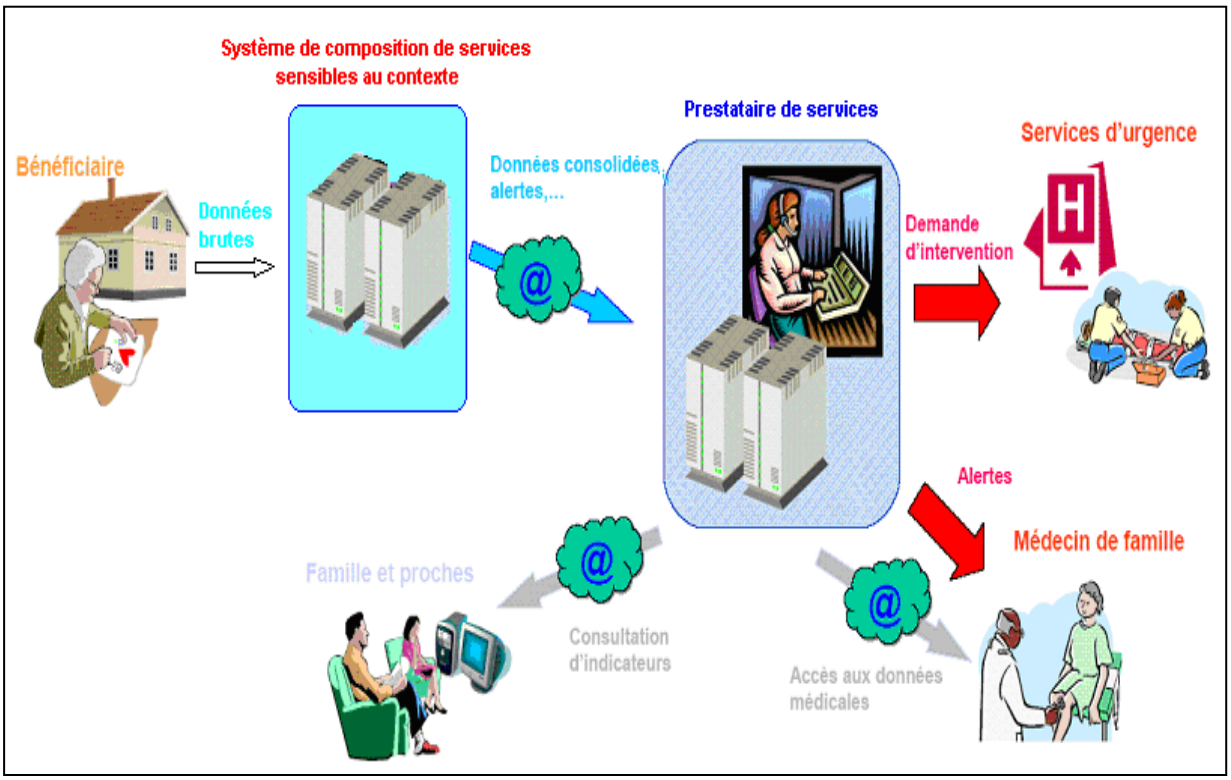
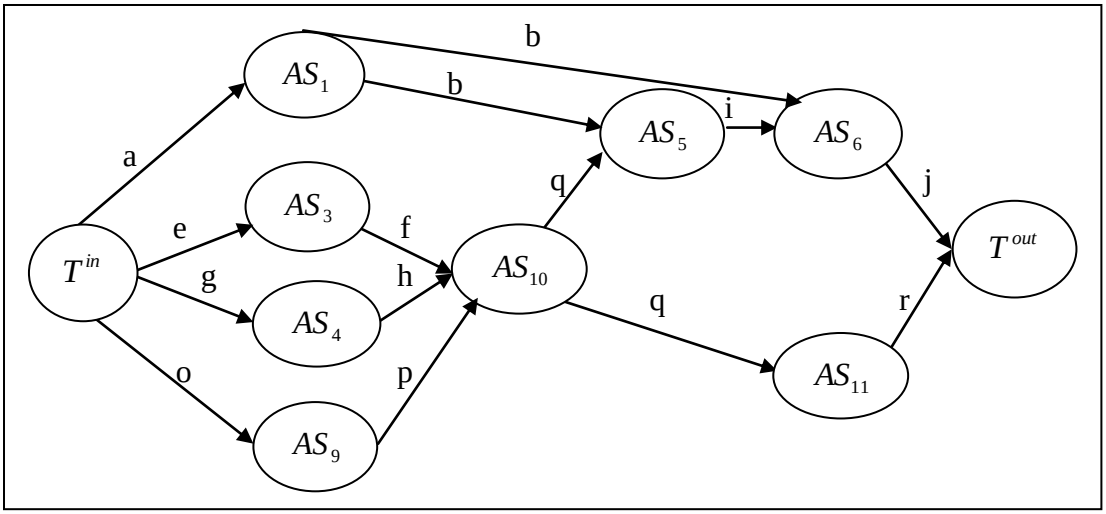


Fig. VII.2 : Composition de services sensibles au contexte pour une personne dépendante

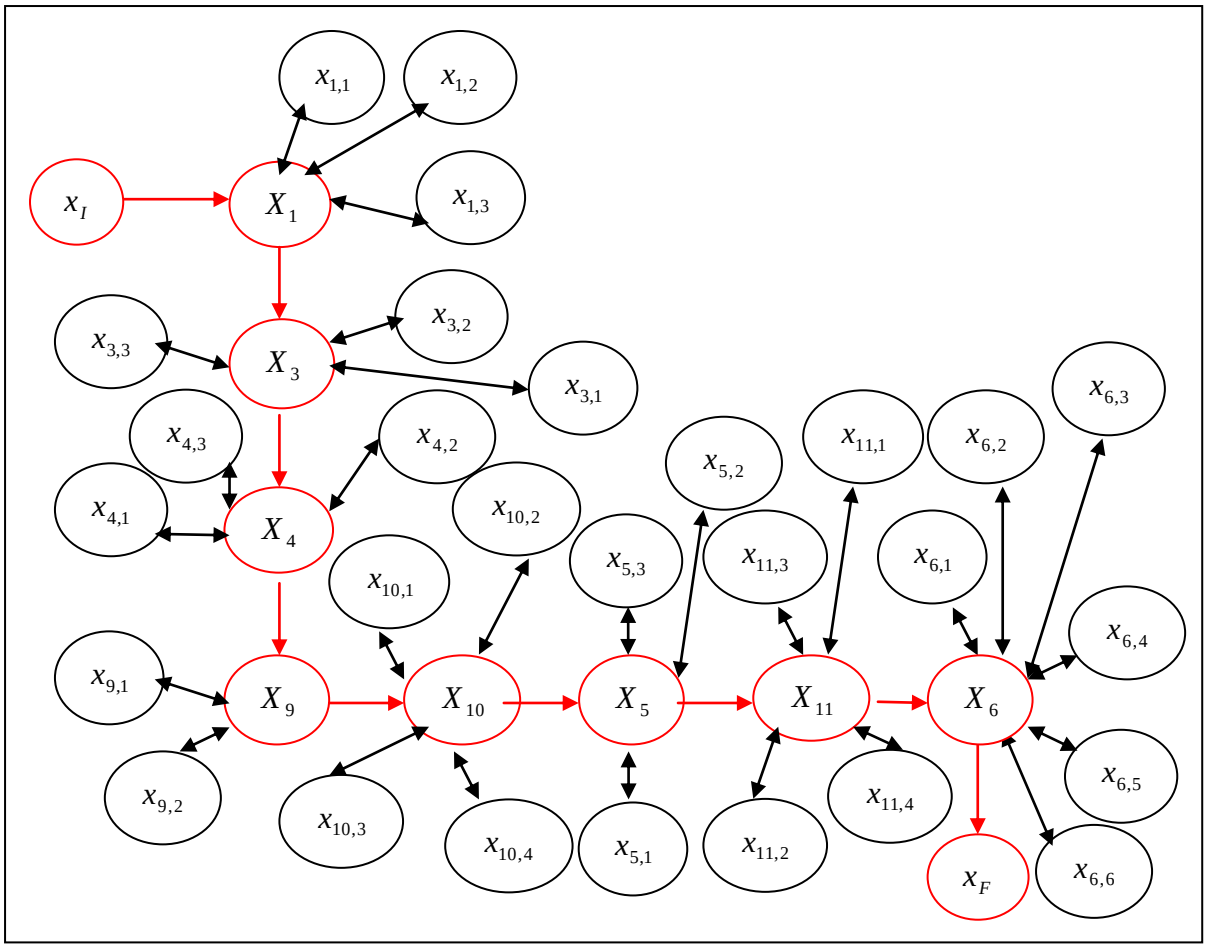
VII.2.4 Application des approches de composition

Supposant qu'à un instant donné, le système de composition reçoit les paramètres (a, e g, o) du système de capture et désire obtenir les paramètres (r, j) à transmettre pour les prestataires de services. L'application de l'algorithme donne les résultats suivants :

VII.2.4.1 Plan de composition



VII.2.4.2 Approche MDP

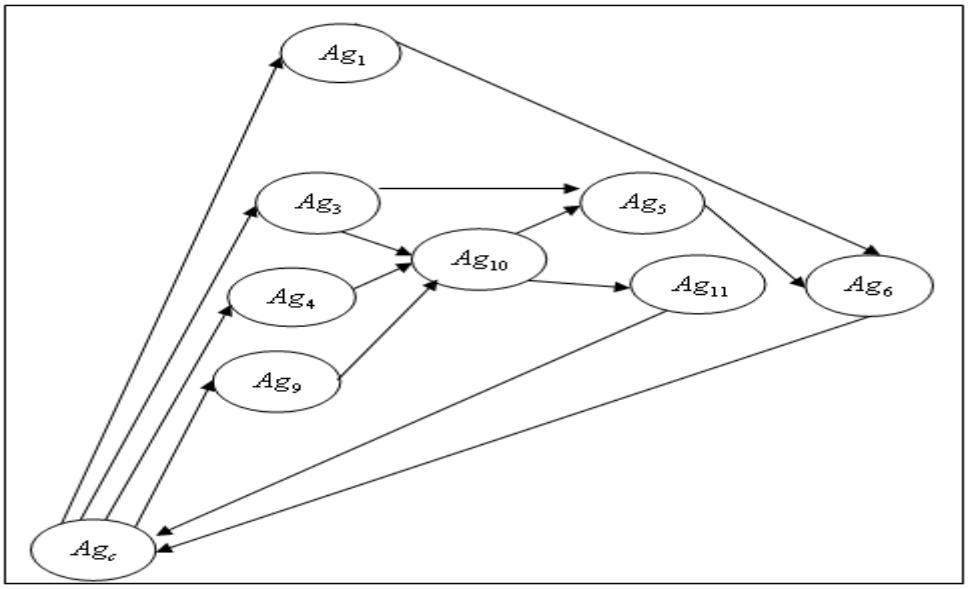


Services abstraits	**Services concrets**	**Exemple de règles de contexte**
AS_1	$cs_{1,1}$: Service localisation avec protocoles 802.11 $cs_{1,2}$: Service localisation avec Bluetooth $cs_{1,3}$: Service localisation avec 802.11 et Bluetooth	Si $signal \in E \Rightarrow localisation = chambre$ E : sous ensemble de capteurs de position
AS_3	$cs_{3,1}$: service d'analyse des mouvements sur PDA $cs_{3,2}$: service d'analyse des mouvements sur PC $cs_{3,3}$: service d'analyse des mouvements sur PC Portable	Si signal $< val1 \Rightarrow$ Personne immobile. Si $val1 \leq$ signal $< val2 \Rightarrow$ Personne moins mobile. Si signal $\geq val2 \Rightarrow$ Personne très mobile
AS_4	$cs_{4,1}$: service d'analyse de l'activité cardiaque sur PDA $cs_{4,2}$: service d'analyse de l'activité cardiaque sur PC $cs_{4,3}$: service d'analyse de l'activité cardiaque intégré	Si signal $< val'1 \Rightarrow$ Activité basse. Si $val'1 \leq$ signal $< val'2 \Rightarrow$ Activité moyenne. Si signal $\geq val'2 \Rightarrow$ Activité élevée
AS_5	$cs_{5,1}$: service de surveillance des mouvements du robot 1 $cs_{5,2}$: service de surveillance des mouvements du robot 2 $cs_{5,3}$: service de surveillance des mouvements du robot 3	/
AS_6	$cs_{6,1}$: service caméra 1 (salon) $cs_{6,2}$: service caméra 2 (cuisine) $cs_{6,3}$: service caméra 3 (chambre) $cs_{6,4}$: service caméra intégrée au robot 1 $cs_{6,5}$: service caméra intégrée au robot 2 $cs_{6,6}$: service caméra intégrée au robot 3	/
AS_9	$cs_{9,1}$: service de planning des soins 1 $cs_{9,2}$: service de planning des soins 2	/
AS_{10}	$cs_{10,1}$: service de surveillance intégré au dispositif 1 $cs_{10,2}$: service de surveillance intégré au dispositif 2 $cs_{10,3}$: service de surveillance intégré au dispositif 3 $cs_{10,4}$: service de surveillance intégré au dispositif 4	si la personne est immobile et son activité cardiaque élevé $\Rightarrow$ $état\_personne = grave$ (accident). si la personne est moins mobile et son activité cardiaque basse $\Rightarrow$ $état\_personne = mauvais$.
AS_{11}	$cs_{11,1}$: service de notification par bip sonore $cs_{11,2}$: service de notification par son et image $cs_{11,3}$: service de notification par message texte $cs_{11,4}$: service de notification par signal	Si $état\_personne = grave \Rightarrow alerte = rouge$. Si $état\_personne = mauvais \Rightarrow alerte = orange$. Si $état\_personne = normal \Rightarrow alerte = verte$.

Tableau VII.4 : Les services concrets et quelques règles de contexte

VII.2.4.3 Approche agents

Les agents se réorganisent afin de répondre au besoin de la personne. L'application de l'algorithme des agents donne le schéma de communication entre les agents présenté ci-dessous et chaque agent doit sélectionner à son tour un service concret approprié pour le contexte de l'utilisateur.



Notre modèle de composition de services sensibles au contexte permet de prendre en compte de nombreux autres services pour tout type d'occupants tels que : la surveillance des équipements domotiques, la sécurité domestique, les services personnalisés à domicile, surveillance des activités de la personne en cas de canicule, et réduction des risques d'accidents liés au vieillissement (risques de chutes, de brûlures, intoxication, dénutrition, etc.)

VII.3 Implémentation et testes

Nous avons empiriquement mesuré les performances de notre algorithme d'apprentissage **MDP-SCWL**. En particulier, nous nous sommes intéressés à déterminer la vitesse de convergence de cet algorithme vers les vraies probabilités. Cette vitesse est mesurée en terme de nombre de passes de la boucle extérieure de l'algorithme **MDP-SCWL**. Ce nombre est appelé *Episode*. Afin de mesurer la distance entre les distributions de probabilité μ, nous avons utilisé une métrique appelée KLD *(Kullbach Leibler Divergence)* définie dans [19] comme suit :

Définition (*Kullbach Leibler Divergence (KLD)*) : le KLD entre deux distributions de probabilités, p et q, est défini comme :

$$D(p//q) = \sum_x p(x) \log_2 \frac{p(x)}{q(x)}$$

$$D(p//q) = 0 \quad \text{Si} \quad p = q$$

Notre méthode de teste consiste à générer d'une façon aléatoire des probabilité initiales pour chaque service concret, et mesurer le KLD à chaque Episode de l'algorithme MDP-SCWL. Les tests ont portés sur 1000 services concrets scindés sur 10 services abstraits. Les résultats de 3 exécutions de l'algorithme MDP-SWL sont montrés sur le tableau suivant :

Episode	Run1	Run2	Run3
1	0,42174	0,9083	0,17532
2	0,068691	0,068583	0,086021
3	0,011664	0,10293	0,010076
4	0,10748	0,03054559	0,13102
5	0,000125	0,0023104	0,00061752
6	0,000052096	0,021198	0,00053492
7	0,018158	0,053983	0,000036479
8	0,00089328	0,0018799	0,000021279
9	0,0074302	0,00023791	0,024115
10	0,0035869	0,000009954	0,037181
11	0,0031537	0,011325	0,00015791
12	0,0023348	0,0029837	0,00026265
13	0,0013423	0,012868	0,0018976
14	0,0034103	0,0012931	0,0057775
15	0,00017448	0,029793	0,0003847
16	0,0021196	0,0023043	0,0046379
17	0,0019496	0,00010366	0,00058202
18	0,0014916	0,0014133	0,00022897
19	0,00029246	0,0027332	0,00025001
20	0,022412	0,0001284	0,00085563
21	0,0014799	0,0013415	0,0002273
22	0,0016854	0,00054154	0,0002381
23	0,0053121	0,0066057	0,0031537
24	0,00046783	0,022483	0,0018549
25	0,0025395	0,00095112	0,0039051
26	0,003155	0,0054498	0,00168
27	0,00066058	0,00046747	0,0024112
28	0,0031801	0,0022122	0,00075914
29	0,0022345	0,0004494	0,016286
30	0,0039133	0,0015303	0,0040761
31	0,00066511	0,00083738	0,0016412
32	0,0075498	0,0010713	0,00093528
33	0,00075093	0,00078664	0,0024779
34	0,001008	0,00013776	0,00028102
35	0,00034881	0,0015765	0,0035403
36	0,000092332	0,0060822	0,0039404
37	0,000026107	0,0038241	0,0039074
38	0,0012111	0,0018975	0,00059268
39	0,0014651	0,00024781	0,00012357
40	0,0012179	0,0002772	0,0020287
41	0,0009622	0,0013125	0,00055616
42	0,0010182	0,001833	0,00010008
43	0,0011415	0,00025138	0,0014345
44	0,0025284	0,00077809	0,00050061
45	0,0022405	0,0002349	0,0042323
46	0,0013413	0,000056924	0,0019325
47	0,00029688	0,00043533	0,00044875
48	0,0025725	0,00018688	0,00067885
49	0,00056837	0,0015629	0,00061053
50	0,000033379	0,00099174	0,001131
51	0,00072234	0,00084882	0,00071972
52	0,0010711	0,00019338	0,00067018
53	0,0025508	0,0019796	0,0013411
54	0,000444	0,0010049	0,00073386
55	0,0017353	0,0013211	0,00024554
56	0,0010494	0,00057145	0,00084978
57	0,00036932	0,00020062	0,00032162
58	0,0011057	0,00044122	0,0025472
59	0,0012361	0,000038446	0,0024377
60	0,0028952	0,0021596	0,00049521
61	0,00029103	0,00082657	0,00024143
62	0,00071246	0,0027168	0,0009626
63	0,00080078	0,0010705	0,00058616
64	0,00076955	0,0015968	0,0013366
65	0,00046865	0,0014022	0,00084085
66	0,0006262	0,00087162	0,0015726
67	0,0012589	0,00093048	0,00089437
68	0,00042421	0,00054046	0,0011227
69	0,00030345	0,00031834	0,00052303
70	0,001833	0,0012105	0,00081496
71	0,00067067	0,0010048	0,0010015
72	0,00096555	0,0008879	0,000037492
73	0,0016631	0,0014425	0,00045524
74	0,0006937	0,00011749	0,00097679
75	0,00043144	0,00067433	0,00070262
76	0,0029348	0,0011418	0,00048458
77	0,00036079	0,0013579	0,0010209
78	0,00076712	0,001314	0,00030019
79	0,00044826	0,00073186	0,0025947
80	0,00048343	0,00085833	0,00052294
81	0,0017438	0,00012949	0,0004583
82	0,00057494	0,00013472	0,0012061
83	0,00039246	0,00059772	0,00025189
84	0,00087367	0,0012859	0,00060404
85	0,0013027	0,00067958	0,00044057
86	0,0021249	0,00042042	0,00085291
87	0,00034442	0,00094861	0,00026593
88	0,0012265	0,00082298	0,00069183
89	0,0010984	0,00045095	0,0004228
90	0,00042638	0,0015808	0,00027672
91	0,0016598	0,00031216	0,00038882
92	0,0018973	0,0004268	0,00023023
93	0,00068849	0,0015981	0,0010699
94	0,00093035	0,0007918	0,0013311
95	0,00030588	0,00045062	0,0016866
96	0,00021369	0,00096693	0,0017875
97	0,0012373	0,00040637	0,00022289
98	0,0012633	0,00027479	0,00067658
99	0,00052626	0,0010058	0,00078679
100	0,0018972	0,0012641	0,0010969
101	0,0002652	0,0014826	0,00045786

102	0,00029664	0,00012277	0,00047847
103	0,00066306	0,000044108	0,00037
104	0,00072581	0,00043266	0,00094088
105	0,00040623	0,00065165	0,0011906
106	0,0013941	0,00029938	0,0010675
107	0,0022841	0,00088359	0,00034088
108	0,0022844	0,0026628	0,00066346
109	0,0016326	0,00050853	0,0017767
110	0,000377	0,0012033	0,0004182
111	0,0011035	0,00071183	0,00023648
112	0,00043199	0,00067217	0,0022242
113	0,00063061	0,0026514	0,0014519
114	0,0019941	0,00080321	0,0012813
115	0,0001636	0,0014402	0,00083477
116	0,00073027	0,00066707	0,0012476
117	0,00013649	0,00051857	0,00053629
118	0,00098496	0,00092585	0,00063794
119	0,00067838	0,00017611	0,00028819
120	0,00046534	0,00037159	0,00017111
121	0,0011524	0,00079176	0,00089455
122	0,0008241	0,00093857	0,0017251
123	0,001195	0,00035464	0,00069367
124	0,00048192	0,001111	0,00036335
125	0,00077237	0,00028883	0,0010983

126	0,0015013	0,00089034	0,00052326
127	0,00058817	0,00038164	0,000263
128	0,0012314	0,0020035	0,00047861
129	0,00048894	0,00049756	0,00024624
130	0,0011277	0,00026939	0,00071898
131	0,00027228	0,00098851	0,00017004
132	0,00074732	0,00057614	0,00044693
133	0,00035557	0,00033566	0,00096223
134	0,00081894	0,00047692	0,0010583
135	0,0011443	0,00085034	0,00067067
136	0,00037514	0,0016641	0,00064072
137	0,00027094	0,00062222	0,00036534
138	0,00033166	0,00088946	0,00059824
139	0,00031656	0,00031123	0,0018719
140	0,00012402	0,00062867	0,00061708
141	0,000020981	0,00033853	0,00029959
142	0,00023916	0,00027476	0,00034097
143	0,0022419	0,00021494	0,00025346
144	0,00014967	0,00028966	0,00072335
145	0,00052233	0,0008071	0,00064205
146	0,00045101	0,00031506	0,000082377
147	0,00083026	0,00036264	0,00026796
148	0,00053657	0,00024127	0,00088018
149	0,00028436	0,00018935	0,0013124

Tableau VII.5 : Les résultats des tests d'apprentissage sur trois exécutions

Les trois graphes associés à ces trois exécutions sont montrés sur la figure suivante :

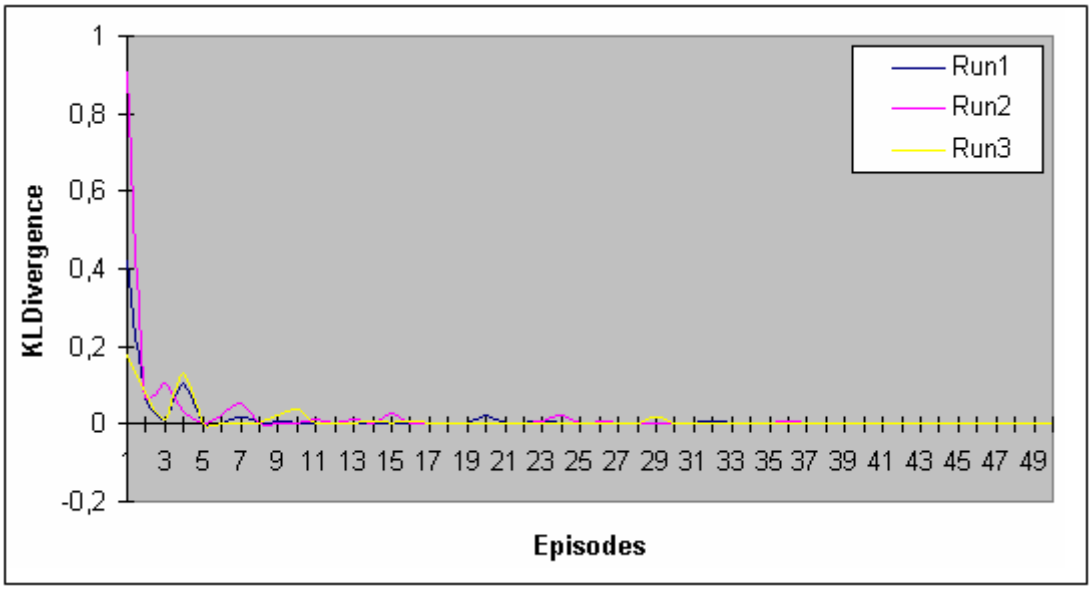


Fig. VII.3 : Graphes des performances de l'apprentissage Bayésien sur trois exécutions

Les trois graphes montres clairement qu'après quelques fluctuations initiales, la convergence vers les vraies probabilités est presque atteinte à la 30ième Episode. Les pics dans les graphes indiquent le KLD associé aux actions les plus invoquées durant le processus d'apprentissage. Puisque les probabilités de départ sont aléatoires, ces pics sont alors à leur maximum au début du processus d'apprentissage. Les fluctuations initiales s'étalent jusqu'à la 5ième Episode. A partir de cette Episode, nous constatons une nette diminution des pics en nombre et en amplitude. Cela s'explique par le fait qu'il y a moins de mises à jours sur les probabilités. De ce fait, le KLD diminue progressivement jusqu'à la 30ième Episode où ils se stabilisent. Cette partie constitue un état transitoire dans lequel la plupart des probabilités sont mises à jour. A partir de la 30ième Episode, le KLD se stabilise pour toutes les exécutions et tend vers la valeur 0. Cette partie constitue un état stationnaire dans lequel les vraies probabilités sont atteintes.

Pour mieux visualiser ces étapes d'apprentissage, nous proposons de diviser le graphe précédent en deux sous graphes. L'agrandissement ci-dessous montre les fluctuations initiales jusqu'à la 5ième Episode :

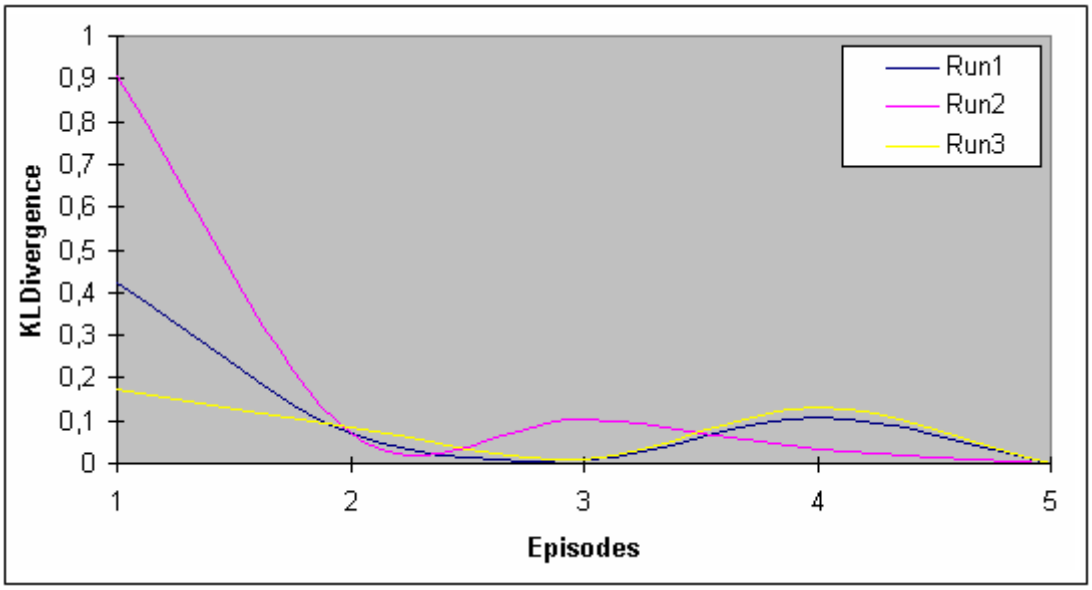


Fig. VII.4 : Graphes des premières fluctuations de l'apprentissage Bayésien

Nous remarquons qu'à partir de la 4ième Episode, l'algorithme arrive déjà à un KLD un peu moins de la valeur 0.1. Cela montre la rapidité avec laquelle l'algorithme exécute les actions et met à jour leurs probabilités.

La partie qui se situe entre la 5ième et la 30ième Episode est la partie la plus active dans le processus d'apprentissage, car elle présente un nombre important de pics mais avec des amplitudes ne dépassant pas la valeur de 0.06. Cela montre qu'il y a beaucoup de mises à jour mais à petites valeurs. Pour mieux visualiser cette partie, nous proposons l'agrandissement ci-dessous :

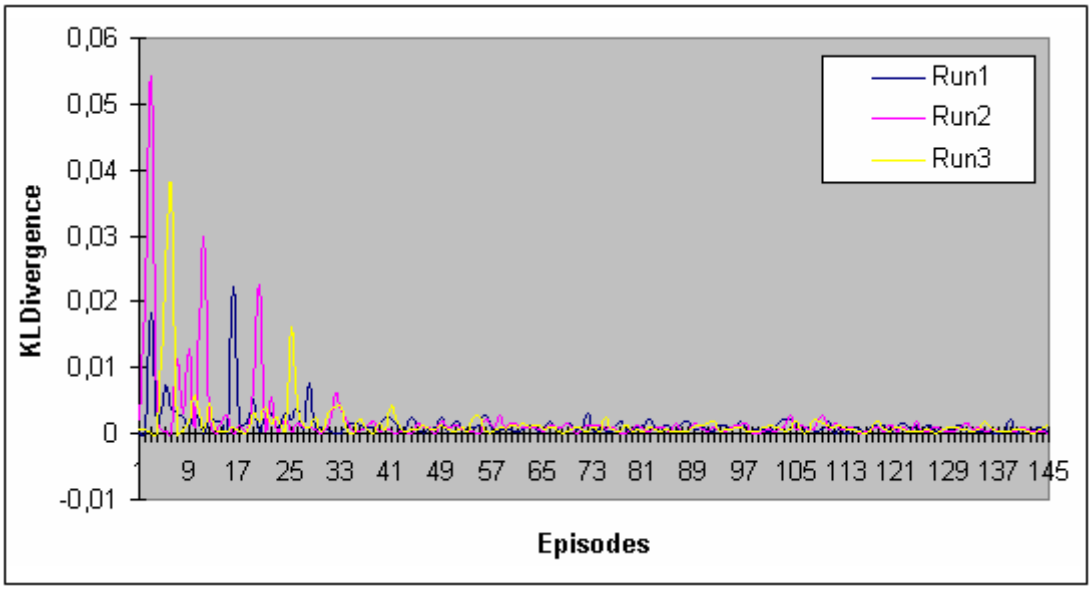


Fig. VII.5 : Graphes de la partie active de l'apprentissage Bayésien

Les graphes obtenues pour les autres testes que nous avons effectués donnent tous la même allure que celle des graphes précédents. De plus un seul teste coûte 1.3 second sur une machine Pentium 4, CPU 2.8 GHz, 256 Mo de RAM. Puisque un teste comprend 150 Episodes, et la convergence est obtenue au bout de 30 Episodes, donc l'algorithme prend environ (30/150)*1.3 = 0.26 second pour converger vers les vraies probabilités. Cette convergence est obtenue en un temps nettement supérieur à celui de l'algorithme *Execute&Learn* proposé par Mr. Doshi [19]. Ceci s'explique par le faite que cet algorithme exécute à chaque Episode l'algorithme *Value iteration* qui est gourmant en temps et en espace. De plus, l'algorithme *Execute&Learn* n'effectue l'apprentissage Bayésien qu'après avoir obtenu toutes les réponses des actions en se basant sur un autre algorithme appelé *Compose&ExecuteWorkflow*. Donc, si une seule action ne répond pas, l'algorithme doit refaire tout l'apprentissage.

Ces résultats obtenus sont applicables sur le modèle à base d'agents que nous avons proposé. En effet, un agent coordinateur attend un délais de t_1 pour avoir la réponse de tous les agents de services. Ces derniers à leur tour, attendent un délais de t_2 pour obtenir une réponse d'une action exécutée et effectuer par la suite l'apprentissage Bayésien. Nos tests ont montrés qu'au bout de 0.26 second, les vrais probabilités sont presque atteintes. De ce fait, un agent de service doit arrêter son apprentissage si aucune réponse positive n'a été obtenues pendant un délais de t_2 = 0.26 second. De même, un agent coordinateur doit attendre la réponse de tous les agents de services pendant un délais de t_1. Supposant qu'il existe 10 agents de services (services abstraits), et chacun possède 100 actions (services concrets). Puisque chaque agent prend au maximum 0.26 second, les 10 agents vont prendre un temps total de 0.26 * 10 = 2.6 seconds. Soit donc t_1 = 2.6 seconds, le temps que prendra ce modèle agent pour la composition de services. Ce temps n'inclut pas les communications inter agents.

Conclusion et perspectives

Au cours de cette thèse, nous nous somme intéressé à la modélisation et la composition de services sensibles au contexte. Un domaine de recherche très actif qui se situe à la convergence du Web et de l'intelligence artificielle. Il vise à offrir des services adaptés au contexte d'utilisation. Cependant, beaucoup de travail reste à faire dans ce domaine notamment sur le plan d'automatisation de la composition et la gestion des perturbations au niveau des services. C'est ainsi que les travaux de recherche présentés dans ce mémoire ont fait l'objet de deux contributions majeures.

La première contribution concerne la modélisation du contexte et des services sensibles au contexte. Le contexte est d'abord scindé en quatre classes, puis modélisé par les langages d'ontologie OWL et OWL-S. Cela permet aux services et applications d'interpréter le contexte et de faire un raisonnement sur ce dernier. Les services sensibles au contexte sont, quant à eux, classés en deux catégories : les services abstraits et les services concrets. Un service abstrait peut être vu comme une description d'une certaine tâche à réaliser, alors qu'un service concret représente la réalisation concrète de cette tâche. Un service abstrait est une classe de services concrets réalisant la même tâche. Cette catégorisation des services facilite leur gestion et permet de restreindre l'espace de recherche.

En s'appuyant sur un raisonnement sur les entrées et sorties des services abstraits, nous avons mis en place un algorithme qui permet de vérifier une composition de services et de construire automatiquement un plan pour cette composition lorsque cette dernière est bien vérifiée. Les changements qui surviennent par la suite dans l'environnement, tels que l'apparition et la disparition des services, n'influent pas sur le plan de composition puisqu'il est construit d'une façon abstraite, c'est-à-dire sur un ensemble de services abstraits. Un plan abstrait n'a pas une réalisation concrète bien déterminée. Cette dernière peut être changée en fonction des services concrets disponibles et leur adaptation au contexte d'utilisation. Ainsi, le plan de composition construit par notre algorithme devient flexible et adaptable aux changements.

La deuxième contribution concerne la proposition d'une approche qui permet la réalisation concrète d'un plan de composition. Il est évident que cette réalisation doit prendre en compte la pertinence des services concrets et les perturbations qui surviennent sur ces derniers. Le modèle MDP est bien placé pour répondre à cette problématique. En effet, la fonction de récompense présentée par ce modèle permet de refléter le degré de pertinence des services, alors que la fonction de transition de ce même modèle reflète les perturbations qui surviennent au niveau des services représentées comme des probabilités de réponse. Les états du MDP sont des services abstraits, alors que ses actions sont des services concrets. De cette façon, on aura pu transformer le problème de composition de services en un problème de prise de décision dans un environnement incertain. Cependant, la connaissance des vraies probabilités du modèle nécessite l'introduction des méthodes d'apprentissage. Pour cette raison, nous avons introduit l'apprentissage Bayésien dans le modèle MDP afin d'intercaler l'exécution et l'apprentissage. Les résultats obtenus, lors des différent testes, montrent clairement la performance de cette méthode adoptée.

Nous avons introduit le paradigme agent afin de donner plus de coopération, de distribution et d'intelligence au modèle de composition de services. Un agent de service est associé à chaque

service abstrait. L'ensemble des agents est organisé d'une manière hiérarchique, dans laquelle chaque agent possède des pères et des fils. Un agent reçoit d'abord des données de ses pères, puis, il sélectionne le service concret le mieux adapté au contexte de l'utilisateur. Ensuite, il envoie le résultat obtenu à tous ses fils. L'organisation des agents est aussi circulaire, puisque le début et la fin de la composition représentent les données d'une même tâche de la requête. Un agent coordinateur est alors prévu afin de lancer l'exécution de la requête et de récupérer le résultat de son exécution. Un agent contexte est aussi prévu afin de permettre à tous les agents d'accéder au contexte et l'exploiter au cours de leur exécution.

Comme perspectives, nous envisageons un ensemble de travaux futurs qui concernent plus spécifiquement les aspects suivants :

- Prise en compte de la gestion de la sécurité et de la confidentialité lors du processus de composition de services.

- Extension de la description des services sensibles au contexte afin de prendre en compte leurs comportements. C'est-à-dire l'ensemble des actions élémentaires qu'un service peut exécuter et dans quel ordre. Cette description comportementale des services est généralement adoptée dans le cadre des automates à états finis.

- Lever certaines hypothèses retenues dans le cadre de notre architecture de composition de services sensibles au contexte, notamment l'hypothèse liée à l'intergiciel sensible au contexte. En effet, nous avons supposé l'existence d'un intergiciel qui gère et fournie le contexte aux applications. Il est tout à fait possible de développer cette partie et construire un modèle complet de gestion du contexte.

- Construire un environnement d'intelligence ambiante en se basant sur le simulateur USARsim et lui intégrer les algorithmes développés dans le cadre de ce présent mémoire. USARsim permet de construire des robots intelligents et gérer leurs parties physiques, alors que nos algorithmes fonctionnent au niveau des services. Donc, il est question d'abord de développer une architecture de gestion du contexte qui ferra la jonction entre les robots construits et les services offerts. Cette architecture reçoit le contexte à partir des différents robots, et offre des services sensibles au contexte en s'appuyant sur les approches proposées dans ce mémoire.

- Améliorer le modèle agent proposé en lui procurant un peu plus de distribution. Ceci passe notamment par la suppression de l'agent coordinateur. Dans ce cas, chaque agent de service vérifie d'une manière autonome s'il doit participer à la composition ou non. Nous proposons également d'expérimenter le modèle agent en s'appuyant sur la plate forme JADE.

Bibliographie

[1] T.C. Agoston, T. Ueda, Y.Nishimura. "*Pervasive computing in a Networked Word*". In Global Distrbuted Intelligence For Everyone, INET2000: 10th Annual Internet Society Confernce. 18-21 July 2000. Yokohama, Japan.

[2] M. Aksit and Z. Choukair. *"Dynamic, adaptive and reconfigurable systems overview and prospective vision"*. In *ICDCS Workshops 2003*, pages 84–94, 2003.

[3] D. Austin. "*Web Services Architecture Requirement*". W3C Working Draft 14 Novembre 2002. disponible sur le site: « *http://www.w3.org/TR/wsa-reqs* ».

[4] Bazire and Brézillon, «*Understanding Context Before Using it*», LNAI 3554, Modeling and Using Context, 2005.

[5] R.Bellman. "*Dynamic Programming*". Princeton University Press, 1957.

[6] M. Bilenko, W.W. Cohen, S. Fienberg, J.R. Mooney, and R. Ravikumar. "*Adaptive name matching in information integration*". *IEEE Intelligent Systems*, 18(5). 2003.

[7] D. Bobrow, R. Gabriel, and J. White. *"Clos in context - the shape of the design space. In Object-Oriented Programming : the CLOS Perspective"*, MIT Press, 1993.

[8] P. Brézillon, C. Kintzig, G. Poulain, G. Privat, P.N. Favennec. *" Expliciter le contexte dans les objets communicants*". Hermès, chapitre 21, 2002, p. 295-303.

[9] P.J. Brown, **"***The Stick-e Document : a framework for creating Context-aware applications*". Electronic Publishing '96 (1996) 259-272

[10] P. J. Brown. "*Triggering information by context".* In personnal Technologies(1998). v. 2, p. 19.

[11] T. Chaari, F. Laforest et A. Flory. *" Adaptation des applications au contexte en utilisant les services WEB"*. ConferenceUbiMob'05, 3-6 June, 2005, Grenoble, France. ACM 1-59593-172-/4/05/0005.

[12] H. Chen, T. Finin, and A. Joshi. *"An ontology for a context aware pervasive computing environment".* In The Proceedings of the IJCAI Workshop on Ontologies and Distributed systems (IJCAI 03), Acapulco MX, August 2003.

[13] H. Chen et al. "*Semantic Web in a Pervasive Context-Aware Architecture*", Proceedings of PerCom 2004, Orlando FL., March 2004.

[14] A. Chibani, K. Djouani, Y. Amirat, *"Agents-middleware approach for context awareness in pervasive computing"*, ICEIS 2003, Proceedings of the 5th International

Conference On Enterprise Information Systems, Vol. 4, 23-26 April 2003, Angers, France, pp.184-189.

[15] A. Chibani, " *Intergiciel multi agents orienté web sémantique pour le développement d'applications ubiquitaires sensibles au contexte* ". Thèse de doctorat de l'Université Paris XII, soutenue le 13 décembre 2006

[16] R. Dale. *"Inside COM, Microsoft Component Object Model".* Microsoft Press, 1996.

[17] K.A Dey, D.G. Abowd. *"Towards a better understanding of context and contextawarness".* In Computer Human Interactions (CHI2000) Workshop on the What, Who, Where and How of Context Awarness. 2000.

[18] A.K. Dey, G.D. Abowd, and D. Salber. *"A conceptual framework and toolkit for supporting the rapid prototyping of context-aware applications"*. Human-computer Interaction, 16(2-4 (special issue on context-aware computing)) :97–166, December 2001.

[19] P. Doshi, R. Goodwin and R. Akkiraju, K. Verma: *"Dynamic Workflow Composition using Markov Decision Processes"*. International Journal of Web Services Research, 2(1): 1-17, Jan-March 2005.

[20] J. Euzenat, J. Pierson, F. Ramparany : *"Dynamic context management for pervasive computing"* . INRIA Rhône-Alpes, France Telecom R&D 2004 .

[21] J.Ferber. *"Les Systèmes multi-agents : Vers une intelligence collective"* . Inter Editions, 1995.

[22] T.Finin, R.Fritzson, D.McKay, and R. McEntire. *"Kqml as an agent communication language"*. In Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94), pages 456-463, Gaithersburg, MD, USA. ACM Press, 1994.

[23] FIPA (2006). http ://www.fipa.org.

[24] M.S Fox. *"An organizational view of distributed systems"*. In IEEE Transactions on Systems, Man, and Cybernetics, volume 11, pages 70-80. 1981.

[25] E. Frank."*DCOM : Microsoft Distributed Component Object Model*." Hungry Minds, Inc, 1997.

[26] T.Gardner. "*An Introduction to Web Services*". 02-October-2001. disponible sur le site: " *http://www.ariadne.ac.uk/issue29/gardner* "

[27] J.M Geib, C. Gransart, and P. Merle. "*Corba : des concepts à la pratique*". Dunod Informatique, September 1999.

[28] C. Goldman, and S. Zilberstein. *"Decentralized control of cooperative systems : Categorization and complexity analysis"*. Journal of Artificial Intelligence Research, 22 : 143-174. 2004.

[29] D. Greenwood, P. Buhler, and A. Reitbauer. *" Web Service Discovery and Composition using the Web Service Integration Gateway"*, Whitestein Technologies AG; e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. 789- 790

[30] T.R. Gruber. *"Toward principles for the design of ontologies used for knowledge sharing".* International Journal of Human-Computer Studies, 1995, vol. 43, pp.907-928

[31] T. Gu et al. "*An Ontology-based Context Model in Intelligent Environments*", Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, California, USA, January 2004

[32] A.B. Hassine, S. Matsubara, and T. Ishida. *"A Constraint-based Approach to Horizontal Web Service Composition"*. In IEEE Computer 2006.

[33] K. Henricksen, J. Indulska, and A. Rakotonirainy. *"Modeling context information in pervasive computing systems"*. In *Pervasive 2002*, pages 167–180, Zurich, Switzerland, 2002.

[34] H. Kautz and B. Selman. *"Unifying SAT-based and graph-based planning"*. In *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 318{325, Stockholm, Sweden, 1999.

[35] G. Kiczales. *"Aspect-oriented programming. Surveys",* 28A(4), December 1996.

[36] S.W. Loke. *"Logic programming for context-aware pervasive computing: Languages support, characterizing situations, and integration with the web*". Web Intelligence, pages 44–50, 2004.

[37] D. López *"An eca rule-matching service for simpler development of reactive applications"*. Middleware 2001, 2(7), November 2001.

[38] Z. Maamar, G. Alkhatib, and S. K. Mostefaoui. *"Context based personalization of web services composition and provisioning"*. In Proceedings of the 30th EUROMICRO Conference 2004.

[39] Z. Maamar, S. K. Mostefaoui, and H. Yahyaoui. *"Toward an agent-based and context-oriented approach for web services composition"*. IEEE Transactions on Knowledge And Data Engineering, 17(5):686–697, 2005.

[40] B.A. Malloy and N.A. Kraft and J.O. Hallstrom and J.M. Voas. "*Improving the Predictable Assembly of Service-Oriented Architectures"*, 2006, IEEE Software, Volume 23, IEEE Computer Society Press

[41] S. McIlraith, T. Son, and H. Zeng. *"Semantic web services".* IEEE Intelligent Systems, 16(2):46–53, 2001.

[42] S. McIlraith and T.C. Son. *"Adapting Golog for composition of semantic Web services"*. In Proceedings of the 8th International Conference on Knowledge Representation and reasoning (KR2002), April 2002, Toulouse, France, pp. 482-493.

[43] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. *"Composing Web services on the semantic web"*. The VLDB, November 2003, journal Vol.12, No.4, pp. 333-351.

[44] N. Milanovic, M. Malek. "*Current Solution for Web Service Composition*". IEEE Internet Computing, November 2004, Vol. 8, No.6, pp.51-59.

[45] S. B. Mokhtar, N. Georgantas, and V. Issarny. *"Ad hoc composition of user tasks in pervasive computing environments"*. In Proceedings of the 4th workshop on software composition, 2005 LNCS3628.

[46] T. Moran, and P. Dourish, *"Introduction to this special issue on context-aware computing"*. Human-Computer Interaction, 16, 2-3, 2001.

[47] S.K Mostéfaoui, A. Tafat-Bouzid, and B. Hirsbrunner. *"Using context information for service discovery and composition"*. In Proceedings of the Fifth International Conference on Information Integration and Web-based Applications and Services (IIWAS'2003), Jakarta, Indonesia, pages 129–138, September 15-17, 2003.

[48] G. K. Mostéfaoui, G. Pasquier-Rocha, and P. Brézillon. *" Context-aware computing : A guide for the pervasive computing community "*. In *ICPS*, pages 39–48, 2004.

[49] K. Nahrstedt, D. Xu, D. Wichadakul, and B.Li. *"Qos-aware middleware for ubiquitous and heterogeneous environments*". IEEE Communications magazine, 39(11):2–10, 2001.

[50] S. Oh, D. Lee, AND SOUNDAR R.T. Kumara "*A Comparative Illustration of AI Planning-based Web Services Composition"* . Penn State University. ACM SIGecom Exchanges, Vol. 5, No. 5, December 2005.

[51] J. Oprescu. "*Découverte et composition de services dans des réseaux ambiants* ". Thèse de doctorat de l'Institut National Polytechnique De Grenobl, soutenue le 20 décembre 2004

[52] E. Ort. *"Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools*". Sun Microsystems, April 2005.

[53] A. Pashtan. *"Mobile Web Services, chapter Ontology of mobile user context*". Combridge unversity press, 2005.

[54] C. Peltz. *"Web Service Orchestration and Choreography – A look at WSCI and BPEL4WS"*. IEEE. July 2003.

[55] R. Picard and J. Klein. *"Computers that Recognise and Respond to User Emotion Theoretical and Practical Implications"*. Interacting with Computers, 14(2) :141–169, February 2002.

[56] D. Preuveneers and Y. Berbers. *"Semantic and syntactic modeling of componentbased services for context-aware pervasive systems using owl-s".* In First International Workshop on Managing Context Information in Mobile and Pervasive Environments, pages 30–39, 2005.

[57] G. Privat. " *Des objets communicants à la communication ambiante*". Les Cahiers du Numérique, 3(4) :23–44, 2002.

[58] L. Qiu, Z. Shi, F. Lin: *"Context Optimization of AI planning for Services Composition",* IEEE International Conference on e-Business Engineering (ICEBE'06), IEEE 2006

[59] J. Rao, X. Su. *"A Survey of Automated Web Service Composition Methods*". Semantic web Services and web Process Composition (SWSWPC), First International Workshop, July 6, 2004, San Diego, CA, USA, pp. 43-54.

[60] G. Rey, J. Coutaz. *" Le Contexteur : une abstraction logicielle pour la réalisation de systèmes interactifs sensibles au contexte*". In Proceedings of Interaction Homme Machine (IHM2002), p. 105112. (2003).

[61] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. *"A Middleware Infrastructure for Active Spaces*". IEEE Pervasive Computing, 1(4) :74–83, October 2002.

[62] U. Saif, H. Pham, J.M Paluska, J. Waterman, C. Terman, and S. Ward. *"A case for goal-oriented programming semantics".* In Workshop on System Support for Ubiquitous Computing (UbiSys'03),5th International Conference on Ubiquitous Computing (UbiComp 2003), Seattle, WA, USA, October 12, 2003.

[63] B. Schilit, M. Theimer, and B. Welch. "*Customising mobile applications*". In Proceedings of USENIX Symposium on Mobile and Location-Independent Computing, pages 129–138, August 1993.

[64] B. Schilit, M. Theimer. "*Disseminating active map information to mobile hosts."* In Institute of Electrical and Electronics Engineers (IEEE) Networks. 1994. v. 5, p. 2232.

[65] E.Sirin ant B. Parsia. "*Planing for semantic web services*". In Semantic Web Sevices Workshop at 3rd International Semantic Web Conference (ISWC2004), 2004.

[66] E.Sirin, B. Parsia , D.Wu et al. "*HTN planning for web service composition using SHOP2*". Journal of Web Semantics, 2004, Vol. 1, No. 4, pp.377-396.

[67] N.A. Streitz, and P. Nixon. "*The disappearing computer – introduction*". Communications of the ACM, the Disappearing Computer (special issue), 48(3):32–35, march 2005.

[68] R.G Smith. "*The contract net protocol : High-level communication and control in a distributed problem solver*". In IEEE Transactions on Computers, volume 29, pages 1104-1113. 1980.

[69] D. Spiegelhalter, A. Dawid, , S. Lauritzen, , and R. Cowell. *"Bayesian analysis in expert systems"*. 1993.

[70] M. Uschold and M. Gruninger. *" Ontologies: Principles, Methods and Applications."* Knowledge Engineering Review, February 1996, Vol. 11, No. 2, pp. 93-155.

[71] S. Vinoski. "*Corba : Integrating diverse applications within distributed heterogeneous environments".* IEEE Communications Magazine, 35(2), February 1997.

[72] M. Vukovic and P. Robinson. *"Adaptive, planning-based, web service composition for context awareness".* In 2nd International Conference on Pervasive Computing (Pervasive 2004), Vienna, Austria, April 2004.

[73] R. Want, B. Schilit, N. Adams, R. Gold, K. Petersen, J. Ellis, D. Goldberg, and M. Weiser. *"The PARCTab Ubiquitous Computing Experiment".* Technical Report CSL-95-1, Xerox Palo Alto Research Center, March 1995.

[74] M. Weiser. "*The Computer for the 21st Century*". Scientific American, 165(3):94–104, 1991.

[75] T. Winograd .*"Architectures for context, Human-Computer Interaction*", 2001, p.402-419.

[76] M. Wooldridge, J. Wiley *"An Introduction to MultiAgent Systems"*. 2002.

[77] A. Woolrath, R. Riggs, and J. Waldo. *"A distributed object model for the Java system."* Computing Systems, 9(4) :291{312, 1996.

[78] N. Yahiaoui, B. Traverson, and N. Levy. "*Classification and comparison of adaptable platforms".* In First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04) held in conjunction with ECOOP '04, Oslo, Norway, June, 2004.

[79] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta.*" Reconfigurable Context Sensitive Middleware for Pervasive Computing*". IEEE Pervasive Computing, 1(3) :33–40, July September 2002.

[80] Z.L. Zhao, A.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. *"Qos-aware middleware for web services composition"*. IEEE Tranctions on Software Engineering, 30(5):311–327, 2003.

[81] H. Zhao and P. Doshi : *"A Hierarchical Framework for Composing Nested Web Processes"* In IEEE Computer 2006.

[82] *http://www.service-architecture.com*

[83] *Jena2. http ://jena.sourceforge.net/.*