République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université M'hamed BOUGARA de BOUMERDES

Faculté des Sciences

Département d'Informatique

MEMOIRE DE MAGISTER

Spécialité : Système informatique et génie des logiciels

Option : Spécification Logiciels et Traitement de l'Information

Ecole Doctorale

Présenté par :

CHOUIREF Zahira

Thème

Un système de programmation fonctionnelle pour la composition de services Web.

Devant le jury de soutenance composé de:

Mr. MEZGHICHE Mohamed	Professeur	UMBB	Président.
Mr. BELKHIR Abdekader	Maître de conférence	USTHB	Rapporteur.
Mr. AHMED NACER Mohamed	Professeur	USTHB	Examinateur.
Mr. HAMDANI Chaabane	Maître assistant	UMMTO	Invité.

Année Universitaire : 2007/2008

Remerciements

Tout d'abord, je tiens à remercier le bon Dieu pour m'avoir illuminée et menée jusqu'ici.

Nombreux sont ceux qui m'ont aidé, encouragé, réconforté, soulagé, tout au long de ces années et je ne saurais leur exprimer mes remerciements autant que je le souhaiterais ; leurs discussions, leurs conseils, leur présence, me furent éminemment précieux.

Je tiens tout d'abord à remercier chaleureusement Docteur A.BELKHIR d'avoir accepté d'être mon directeur de thèse. Je lui suis particulièrement reconnaissante pour sa disponibilité, sa compétence, son soutien, ses conseils judicieux et la confiance dont il m'a fait part lors de la réalisation de ce travail. Merci à vous d'avoir relu la thèse plusieurs fois, car il ne serait pas ce qu'il est sans vous.

Je souhaite adresser mes sincères remerciements au président de jury, le professeur M.MEZGHICHE notre responsable du laboratoire LIFAB pour ses efforts et l'intérêt qu'il a apporté à la recherche; au professeur M.AHMED NACER, et à Mr CH.HAMDANI les deux examinateurs de ma thèse qui ont accepté la tâche délicate de rapporter ce mémoire et qui ont eu la patience de juger ce travail, et d'apporter leurs avis éclairés sur mon travail de recherche.

Je remercie aussi le Professeur K.BADARI, le doyen de la faculté des sciences, ainsi que tous les professeurs et personnels de notre université.

Je souhaite aussi remercier mes collègues pour les moments partager en leur compagnie ainsi que tous les membres de l'équipe informatique de LIFAB. Je n'oublie pas non plus tous mes amies (amis), et tous ceux qui j'ai eu l'opportunité de connaître et que j'espère que nous continuerons toujours en contacte. Merci pour les excellents moments que nous avons passés ensemble.

Mes profonds et mes plus grands remerciements vont aux membres de ma famille pour leur patience pendant ces derniers mois difficiles, pour me soutenir, m'encourager, me faire confiance et vouloir toujours que je vole plus haut. Je tiens à remercier mes parents, qui m'ont donné l'envie d'apprendre et la valeur du travail: merci à maman pour son soutien inconditionnel et permanent, merci à mon père d'avoir toujours su me donner les bons conseils même dans les moments de doute, merci pour être des parents formidables et également pour les moments que nous avons partagé, malgré la distance et même loin des yeux, mais toujours très proche au coeur. Merci pour être toujours là, à mon côté, comme aujourd'hui. Merci du fond du coeur.

A la mémoire de notre ami Sofiane MAGRAMANE.

Table des Matières

Liste des Figures

Liste des Acronymes

ACID	Atomicité, Cohérence, Intégrité, Durabilité
API	Application Programing Interface
BPEL	Business Process Execution Language (= **BPEL4WS**)
BPEL4WS	Business Process Execution Language for Web Services
BPWS4J4	Business Process Web Services for Java4
B2B	Business To Business
B2C	Business To Consumer
CCS	Calculus of communicating Systems
CORBA	Common Object Request Broker Architecture
DAML-S	DARPA Agent Markup Language for Services
DARPA	Defense Advanced Research Project Agency
DCOM	Distributed Component Object Model
DPDL	Deterministic Proposisional Dynamic Logic
FS4SWC	Functional System For Web Service Composition
FSM	Finit State Machin
HTTP	HyperText Transfer Protocol.
HTML	HypertText Markup Language
IDL	Interface Description Language
I/O	Input/Output
Lisp	List Processor
LRT	Long-Running Transaction.
METEOR-S	Managing End-To-End Operations for Semantic Web Services
MIME	Multipurpose Internet Mail Extensions
MWSAF	METEOR-S Web Service Annotation Framework
MWSDI	METEOR-S Web Service Discovery Infrastructure
MWSCF	METEOR-S Web Service Composition Framework
OCL	Object Constraint Language
OMG	Object Management Group Ontology
OWL-S	Web Language for Services
PA	Process Algebra
PDDL	Planning Domain Definition Language
P2P	Pire to Pire
QdS	Qualité de Service
QoS	Quality of Service
RDF	Resource Description Framework
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
SOA	Architecture Orientée Services
SOAP	Simple Object Access Protocol
StarWSCoP	Star Web Services Composition Plateform
SWS	Services Web Sémantiques
TCP/IP	Transmission Control Protocol / Internet Protocol
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
URI	Unified Resource Identifier
WS	Web Services
WS-CF	WS-Coordination Framework
WSCL	Web Services Conversation Language
WSCI	Web Service Choreography Interface
WS-CDL	Web Service Choreography Description Language
WS-Discovery	Web Services Discovery
WSDL	Web Services Description Language
WSDL-S	WSDL Semantics
WSFL	Web Service Flow Language
WSMO	Web Service Modeling Ontology
W3C	World Wide Web Consortium
XLANG	eXtensible Language
XML	eXtensible Markup Language
XSRL	XML Web-Services Request Language

Résumé

La troisième génération du Web est orientée service. Cette orientation favorise l'interopérabilité des applications et des systèmes. Les services Web sont des technologies émergentes et prometteuses pour le développement, le déploiement et l'intégration d'applications Internet. Ces technologies, basées sur XML, fournissent une infrastructure pour décrire (WSDL), découvrir (UDDI), invoquer (SOAP) et composer (BPEL4WS) des services. Un des avantages majeurs des services Web par rapport à ses prédécesseurs tels que CORBA, DCOM et XML-RPC est l'apport de l'interopérabilité sur Internet.

L'accès aux services Web est défini comme étant la succession de trois étapes, à savoir : la recherche de fournisseurs du service souhaité, la sélection de l'un de ces fournisseurs et la réalisation du service par l'invocation du fournisseur choisi. Cependant, les services Web ne sont pas capables de résoudre tous les problèmes. Actuellement, de nombreuses infrastructures pour supporter des services, sont déployées par différents organismes. La diversité de ces infrastructures et des organismes qui les déploient entraîne des hétérogénéités. Plusieurs types d'hétérogénéités existent : Les différences technologiques existant entre les infrastructures provoquent une hétérogénéité technologique. L'absence d'une pensée unique et la diversité des personnes définissant des services et leurs descriptions entraînent des hétérogénéités sémantiques et structurelles dans ces définitions. Ces hétérogénéités peuvent intervenir d'une infrastructure à une autre, ou au sein d'une même infrastructure.

En effet, la réponse aux besoins complexes des utilisateurs peut correspondre à l'emploi de plusieurs services hétérogènes (simples et/ou composites) exécutés conjointement. Ce type de problème est connu comme la composition de services web. L'interprétation consistante des données échangées entre services Web composés est gênée par des différences de représentation et d'interprétation sémantiques.

Dans ce contexte, nous proposons un système FS4WSC (*Functional System For Web Service Composition*) pour composer automatiquement les services web sémantiques (SWS). Ce système traite la prise en compte des hétérogénéités sémantiques des données échangées entre les services Web engagés dans une composition. Une nouvelle architecture est proposée pour faciliter la découverte et la composition de SWS. Un système d'annotation sémantique est utilisé dans la plate-forme de services Web afin d'enrichir les services par la sémantique utilisant une ontologie du domaine commune. Ainsi, nous utilisons un module de correspondance et un moteur de composition pour l'appariement entre la requête et les services disponibles et la composition des services si c'est nécessaire. Ensuite, nous présentons notre approche "*Approche par les fonctions*" où nous proposons un formalisme pour décrire les aspects syntaxiques et sémantiques de la requête et des services Web, puis nous utilisons le langage de programmation fonctionnelle "CAML" pour implémenter les algorithmes proposés et les appliquer au domaine de réservation de voyages en ligne.

Mots clés : service Web, composition de services Web sémantique, fonction, composition de fonctions.

Abstract

The third generation of the Web is oriented service. This orientation promotes the interoperability of applications and systems. Web Services are emergent and promising technologies for the development, deployment and integration of applications on the Internet. These technologies are based on XML, providing an infrastructure to support the description (WSDL), discovery (UDDI), interaction (SOAP) and composition (BPEL4WS) of services. One of the major advantages of Web Services compared to their predecessors such as CORBA, DCOM and XML-RPC is enhanced interoperability on the Internet.

The Web services access consists in three steps: the search of service providers, the selection of one of them and the execution of the service by invoking the chosen provider. However, Web services are not able to provide solutions to all problems. Currently, a lot of organisations provide infrastructures to support services. The diversity of those organisations and their infrastructures involves heterogeneity. Several types of heterogeneity exist: Technology heterogeneity, due to the diversity of infrastructures technologies. Structural and semantic heterogeneities in the definition of services and their description, due to the lack of a unique thought (cultural diversities). These heterogeneities can intervene of an infrastructure to another, or within a same infrastructure.

Indeed, the answer to the complex needs of users may require multiple heterogeneous services (simple and/or composite) to be executed. This type of problem is known as the composition of Web services. The consisting interpretation of data exchanged between Web services composites is embarrassed by differences of representation and interpretation semantics.

In this context, we propose a FS4WSC system (Functional System For Web Service Composition) to compose semantics Web services (SWS) automatically. This system treats the hold in account of the semantic heterogeneities of data exchanged between the Web services hired in a composition. A new architecture is proposed to facilitate the discovery and composition of SWS. A Semantic annotation system is used in Web services platform in order to enrich services by the semantics using a same ontology of the domain. Thus, we use a module of correspondence and a motor of composition for the matching between the request and the available services and the composition of services if it is necessary. Then, we present our approach "*Approaches by functions*" where we propose a formalism to describe the syntactic and semantic aspects of the request and the Web services, then we use the functional programming language "CAML" to implement the proposed algorithms and to apply them in line journey booking domain.

Keywords: Web service, Web services semantic composition, function, function composition.

ملخص

إن الجيل الثالث من الواب (Web) هو ذو توجيه خدماتي، وهذا الأخير يشجع التجانس العملي بين التطبيقات والأنظمة. خدمات الواب هي عبارة عن تكنولوجيات بارزة وواعدة في مجال تطوير، نشر وإدماج تطبيقات الانترنيت. هذه التكنولوجيات المؤسسة على (XML)، تعطي قاعدة لوصف (WSDL)، واستكشاف (UDDI) وتحفيز (SOAP) وتركيب (BPEL4WS) الخدمات. إحدى الإيجابيات الهامة لخدمات الواب بالنسبة لسابقاتا مثل CORBA، DCOM و XML-RPC هي ما قدمته للتجانس العملي على الانترنيت.

إن الاستفادة أو الدخول إلى خدمات الواب معرف على أنه تتابع لثلاث مراحل، البحث عن موردي الخدمة المطلوبة، اختبار أحد هؤلاء الموردين والقيام بالخدمة عن طريق إثارة المورد المختار. بينما، مجموع خدمات الواب ليس قادرا على حل كل المشاكل، لذا تقوم منظمات مختلفة بتسخير ونشر العديد من المنشآت القاعدية بدف دعم هذه الخدمات.

إلا أن تنوع هذه المنشآت والمنظمات التي تسخرها يخلق العديد من التباينات - الكثير منها موجود - كالاختلاف التكنولوجي الموجود بين المنشآت القاعدية لهذا المال، والذي ينشأ عنه تباين تكنولوجي. كذلك، غياب فكرة موحدة واختلاف الأشخاص القائمين على تعريف الخدمات، وأوصافها يخلق تباينات دلالية وتنمـوية لهذه التعريفات. ويمكن لهذه التباينات أن تشارك أو أن تتدخل من منشأة إلى أخرى أو في نفس المنشأة.

وبالتالي فالجواب عن المتطلبات المركبة للمستعملين، يمكن أن يتناسب مع استعمال الكثير من الخدمات المتباينة (عادية و/أو مركبة) منفذة بطريقة متوازية. وهذا النوع من المشاكل معروف بـ "تركيب خدمات الواب" حيث التجانس العملي المتماسك أو المتين للمعطيات المتبادلة بين خدمات الواب المركبة مضايق بالاختلافات في التمثيل والترجمة الدلالية.

في هذه القرينة، نقترح نظاما (FS4WSC) أسميناه "النظام الوظيفي لتركيب خدمات الواب" مهمته التركيب الآلي لخدمات الواب الدلالية. ويعالج هذا النظام الأخذ بعين لاعتبار للتباينات الدلالية للمعطيات المتبادلة بين خدمات الواب المدمجة أو المستعملة في تركيبة معينة. لقد تم اقتراح هندسة جديدة لتسهيل اكتشاف وتركيب خدمات الواب الدلالية (SWS)، يستعمل فيها نظام شرح دلالي في أرضية خدمات الواب بغية إثراء هذه الخدمات بالدلالية المستعملة لوجودية أو كينونة مجال مشتركة. ونستعمل كذلك وحدة توافق، ومحرك تركيب للقرن بين العريضة (الطلب) والخدمات الموجودة أو تركيب الخدمات إن اقتضى الأمر. وبعد ذلك نعرض مقاربتنا "المقاربة بالوظائف" أين نقترح تصويرا لوصف الجوانب النحوية والدلالة للعريضة وخدمات الواب، وننتهي إلى استعمال لغة البرمجة الوظيفية "كامل" "KAML" لتجسيد الخوارزميات المقترحة وتطبيقها على مجال حجز سفر على الانترنيت.

الألفاظ المفاتيح: خدمات الواب، تركيب خدمات الواب الدلالية، وظيفة، تركيب الوظائف.

Introduction générale

L'avènement de nouveaux usages de l'informatique dite mobile et ambiante ne permet plus de concevoir des applications logicielles dédiées à des plates-formes prédéfinies, standardisées, composées d'un ensemble de dispositifs qui sont a priori connus. Les logiciels nécessitent toujours plus de capacité d'adaptation face à une multitude de contextes d'utilisation. Que dire alors de l'enjeu proposé par une informatique qui se voudrait d'adapter dynamiquement une application logicielle à un environnement d'exécution découvert dynamiquement, évoluant tout aussi dynamiquement, et partiellement connue à priori. Le paradigme qui permet de gérer une telle application par assemblage de composants se révèle alors particulièrement pertinent lorsqu'il est associé à un ensemble de composants orientés services pour la découverte dynamique de dispositifs.

La notion de SOA (Architecture Orientée Services) définit un modèle d'interactions au niveau logiciel mettant en oeuvre des connexions entre des composants logiciels «fournisseurs» à l'attention de composants logiciels «consommateurs». Cette approche a largement été adoptée pour le développement de systèmes de traitement de l'information favorisant par là même, la distribution de fonctionnalités indépendantes, leur réutilisabilité et leur intégration [SW1] [SW2] [SW3].

Les services Web sont des technologies émergentes permettant une interopérabilité entre les différents acteurs (fournisseurs et demandeurs de services) du fait de leur architecture reposant sur des technologies standard. Ils ont pour vocation de favoriser une architecture orientée services, intégrant des systèmes hétérogènes complexes, fortement distribués et pouvant coopérer sans recourir à une intégration spécifique et coûteuse **[ELF et al04]**.

D'après [ANF01] et [CNW01], l'architecture à services web apparaît pour permettre à des applications hétérogènes s'exécutant au sein de différentes entreprises de pouvoir interopérer à travers des services offerts par les applications. L'hétérogénéité des applications n'est pas seulement considérée au niveau des langages d'implémentation des applications, mais aussi au niveau des modèles d'interaction, protocoles de communication et niveaux de qualité des services.

Dans l'architecture SOA, l'accès aux services peut être décomposé en trois étapes :

1. La recherche et la localisation des fournisseurs proposant le service souhaité, en utilisant l'outil de recherche et de localisation mis à disposition par l'infrastructure sur laquelle le client se trouve.
2. Le choix de l'un des fournisseurs trouvés lors de la première étape.
3. La réalisation du service par l'invocation du fournisseur choisi en utilisant l'une des opérations de l'interface du service.

Etant donné qu'il y a plusieurs services publiés sur Internet, beaucoup d'entre eux, ne peuvent seuls satisfaire les besoins d'un client. Considérons, par exemple, le service d'organisation d'un voyage. Ce service peut se décomposer en plusieurs autres services, tels que :

- Réservation de billets d'avion ou de train ;
- Réservation de chambres d'hôtel;
- Location de voiture ;
- Payement des différentes réservations …etc.

Non seulement un seul de ces services ne permet pas d'organiser un voyage, mais seule une combinaison permet d'atteindre ce but. Il est possible évidemment d'effectuer cette combinaison manuellement, c'est-à-dire que le client choisit séparément chaque service, et organisera leurs exécutions, en prenant en compte toutes les contraintes de temps et de précédences entre chaque service. Mais si la quantité de services proposés est considérable, atteindre le but *Organisation d'un voyage* peut exiger beaucoup de temps et d'efforts de la part du client.

La composition des services web a pour objectif de déterminer une combinaison de services en fonction d'une requête d'un client. Du coté du client, cette composition semblera un unique service. La composition sera transparente au client, même si cette composition sera la combinaison de plusieurs services web.

Par ailleurs, une composition peut être composée de services atomiques ou composés. Les services atomiques sont caractérisés pour avoir une unique fonctionnalité. D'un autre côté, les services composés peuvent regrouper plusieurs fonctionnalités, c'est-à-dire plusieurs services atomiques. En conséquence, une composition peut être soit composée de services composés, soit de services atomiques mélangés.

Nous nous positionnons, dans cette thèse, sur la résolution des problèmes pour composer automatiquement les services web, où les services qui participeront à la composition ne sont pas connus à l'avance. Cela veut dire qu'une fois que la requête d'un client est déterminée, les autres services web qui en dépendent vont être répertoriés au fur et à mesure. Dans un premier temps, nous utilisons des services web très particuliers dont chacun effectue une seule opération. Notre principal objectif est de composer automatiquement les services web dédiés à la réservation de voyage, et plus particulièrement de proposer un système pour composer automatiquement les services web sémantiques (SWS) s'appuyant sur l'approche fonctionnelle prenant en compte la description sémantique des fonctionnalités des services.

L'architecture de notre système est constituée de plusieurs phases qui permettront de découvrir les services afin de proposer automatiquement des compositions. Pour composer les services, il est nécessaire de pouvoir les trouver. Or seule la sémantique permet d'effectuer des recherches sémantiquement similaires. Pour cela nous avons enrichi l'architecture de référence

(section 1.3) par deux sous systèmes (figure 10 section 5.3) : un *système d'annotation sémantique* qui identifie dans un service de demande et d'offre les éléments pertinents à annoter, exploitant une ontologie commune, et un *Module de mise en correspondance et le moteur de composition* qui permettent la recherche de services Web sémantiques adéquat à la requête et la composition des services retournés si c'est nécessaire.

Notre travail est essentiellement axé sur les langages fonctionnels, où le programmeur se soucie davantage sur la description de l'objectif à atteindre. La structure de construction de base d'un langage fonctionnel est la *fonction*, et sa structure de contrôle de base est l'*application de fonction* à ses arguments. Ces langages offrent une meilleure sécurité d'exécution, mais souvent au détriment de l'efficacité, car ils sont interprétés et non compilés. On se focalise sur le langage CAML pour représenter les services Web sémantiques (offre et demande) ainsi que leurs compositions.

Contexte et Problématique

L'infrastructure de base de services Web (section 1.3) permet d'implémenter des interactions simples entre un client et un service Web. Les services Web ont une utilité limitée puisqu'ils n'effectuent qu'une seule tâche spécifique chacun ; par exemple, pour partir en vacances une personne devra trouver un service Web pour chaque réservation qu'elle désire réaliser (avion, hôtel,…etc.).

Si l'exécution d'un service Web implique l'invocation d'autres services Web, il est nécessaire de combiner les fonctionnalités de plusieurs services en services Web plus complexes (services Web agrégés ou composites) afin de répondre à des exigences plus complexes et pour ne former, du point de vue de l'utilisateur, qu'un seul service Web. Ceci permet la définition des applications de plus en plus complexes en agrégeant progressivement des composants à des niveaux plus élevés d'abstraction. On parle alors de **composition de services Web.**

La composition de services Web s'impose dès lors comme l'application la plus naturelle qui permet de mettre à profit les composants services Web au service de l'intégration d'applications. A titre d'exemple, considérons deux services Web : le premier offre un service d'achat de billets d'avions et le deuxième un service de réservation hôtelière appartenant respectivement à une compagnie aérienne et une chaîne hôtelière. Une troisième entreprise d'agence de voyages veut mettre en place un service Web d'organisation de formules de voyages selon une liste de critères ou de préférences (vol avec réservation d'hôtel). Au lieu de mettre en place un nouveau service, elle veut combiner et utiliser un mécanisme de gestion de préférences utilisant les services des deux compagnies précédentes.

Cependant la tâche de composer des services Web est plus complexe du fait de l'autonomie et de l'hétérogénéité de ces derniers, et de la nature dynamique de la composition **[PBM02].** En effet, une composition n'est pas simplement un regroupement quelconque de services Web mais un ensemble dont les tâches sont ordonnées en fonction des relations reliant ses services Web. Les services Web sont généralement fournis par des organisations différentes et indépendamment de tout contexte d'exécution. Puisque, chaque organisation possède ses propres règles de travail, les services Web doivent être traités comme des unités strictement autonomes.

Les services Web présentent avec WSDL une sémantique trop faible pour permettre une réelle validation des compositions de services. Le respect des protocoles relatifs aux services est alors un défi dans un contexte général. Les annotations associées aux définitions WSDL ne permettent pas une validation automatique. A cause de cette hétérogénéité et de cette autonomie inhérente aux services Web, il est difficile de prévoir le comportement d'un service Web composé.

Un des problèmes majeurs, auquel nous nous intéresserons dans notre travail est : étant donné un ensemble de services Web et une requête utilisateur, comment trouver les combinaisons de services qui satisfont au mieux la requête, c'est à dire telles que le maximum d'informations de la requête se retrouve dans les combinaisons et le minimum d'informations en plus soit apporté par ces combinaisons. C'est à la fois le problème de localisation, identification et composition dynamique de services Web, dont l'enjeu est de créer des services complexes qui résolvent le problème à partir de services plus simples, avec l'étude de l'apport des descriptions sémantiques qui leurs sont associées.

Motivation

Le besoin d'automatisation du processus de conception et de mise en oeuvre des services Web rejoint les préoccupations à l'origine du Web sémantique, à savoir comment décrire formellement les connaissances de manière à les rendre exploitables par des machines. En conséquence, les technologies et les outils développés dans le contexte du Web sémantique peuvent certainement compléter la technologie des services Web en vue d'apporter des réponses crédibles au problème de l'automatisation. Par exemple, la notion d'ontologie **[RUO04]** peut jouer un rôle prépondérant pour permettre d'expliciter la sémantique des services facilitant ainsi les communications hommes-machines, d'une part, et les communications machines-machines, d'autre part.

Dans le contexte des services Web sémantiques, la motivation de recherche réside dans la possibilité d'intégrer, dans un modèle d'évaluation des performances d'un service Web, des aspects sémantiques permettant d'identifier de nouveaux critères de qualité d'un service Web et de retenir ces critères dans la mise en oeuvre effective d'un service Web pour la découverte et la composition dynamique de services Web.

Organisation

Cette thèse est scindée en *3 parties* majeures :

La Première partie est consacrée à un état de l'art relatif aux domaines abordés dans cette thèse. Nous présentons dans le Chapitre 1 la définition et les concepts des services Web. Les architectures sont classifiées et évaluées selon leurs évolutions. Nous détaillons aussi la problématique de recherche sur les services Web à savoir *la sélection, la découverte, la découverte dynamique* et *la composition de services Web*. Nous concluons ce chapitre par une conclusion. Dans le Chapitre 2, nous présentons un état de l'art de la composition des services Web sémantiques. Nous abordons en particulier la description des services Web sémantiques (SWS). Nous introduisons aussi la notion de la composition des SWS. Ensuite, nous présentons les différentes issues de la composition ainsi qu'une classification de composition de services. Nous évoquons quelques approches de la composition de services Web proposées par différents auteurs qui s'apparentent à notre approche. Nous nous focalisons sur quelques exigences de la composition pour comparer ces différentes approches. Nous concluons ce chapitre par une conclusion.

La Deuxième partie porte sur une étude complète de la programmation fonctionnelle. Les caractéristiques tant syntaxiques que sémantiques sont étudiées dans le but de mettre en relief leurs avantages et inconvénients ; et de prendre connaissances des constructions syntaxiques les plus importantes, qui serviront de modèles de base pour le travail de modélisation abordé dans la troisième partie. Un tour d'horizon des langages fonctionnels est effectué en termes de définitions, de propriétés, d'une classification générale, et des caractéristiques des langages fonctionnels dans le Chapitre 3. Le Chapitre 4 détaille la notion de fonction qui est un concept essentiel de la programmation fonctionnelle, ainsi que la composition de fonction qui est l'une des opérations de base d'un langage fonctionnel.

La Troisième partie est le cœur de notre travail. Elle est consacrée au système que nous avons développé pour la composition de services Web sémantiques. Nous présentons notre approche proposée : *Approche par les fonctions* qui a pour objectif de modéliser les services Web sémantiques comme des fonctions et leurs compositions utilisant le principe de la composition de fonctions, et de rendre explicite la sémantique des données échangées entre services Web composés afin de dépasser les limites rencontrées par les travaux existants. Cette approche aboutit à notre formalisme construit autour des notions de *fonction, composition de fonctions et des structures de données sémantiques* que nous avons proposées.

Dans le Chapitre 5, nous expliquons le principe de notre approche. Nous définissons tout d'abord plus précisément l'architecture de notre système, ainsi que son fonctionnement et l'intégration de chaque module et chaque phase. Ensuite nous présentons les définitions de base qui forment le formalisme proposé pour représenter les services. Enfin, une représentation des SWS comme une fonction est bien illustrée à travers des structures de données sémantiques que nous avons proposé. Le Chapitre 6 présente la mise en œuvre de notre architecture dans un environnement de composition de SWS, en détaillant les algorithmes proposés dans le système développé. Ce chapitre est illustré par une étude de cas, afin de démontrer que le système proposé peut être appliqué à un domaine quelconque. Dans notre travail, nous avons appliqué ce système à une "*Agence de voyage*" pour effectuer les différentes réservations, en détaillant chaque étape de la composition. Pour cela, nous avons utilisé le langage fonctionnel "CAML" qui permet à l'utilisateur d'exprimer sa perception sémantique des différents objets qu'il souhaite implémenter et d'apporter un moyen simple et naturel pour la définition de nouveaux types de données en privilégiant l'aspect sémantique des services Web (sémantique associée aux noms des services Web, aux opérations, aux entrées/sorties). Ce chapitre termine cette partie par une conclusion qui discute les apports de notre travail, ainsi qu'une comparaison de notre approche avec les approches que nous avons citée.

Pour finir, une conclusion générale reprendra nos contributions apportées dans cette thèse, et en analysera ses limites. Nous proposerons enfin quelques perspectives de recherche envisagées.

Chapitre 1
Les services Web

1.1 Introduction

L'une des tendances historiques qui a conduit à l'apparition des services Web est l'utilisation de l'architecture par composants comme approche d'intégration des applications **[PFI96]**. *Les composants sont des entités logicielles indépendantes fondées sur une interface et une sémantique bien définie.*

Après l'avènement du **B2C** (Business To Consumer), où les entreprises mettent en ligne leurs services pour leurs consommateurs à travers des applications Web, celles-ci souhaitent augmenter leur productivité à l'aide du paradigme **B2B** (Business To Business). Le **B2B** repose sur l'échange de produits, d'informations et de services entre entreprises. Ceci implique l'utilisation de services et la collaboration avec des systèmes proposés par d'autres concepteurs et par conséquent une maîtrise de l'hétérogénéité. L'interopérabilité est ainsi devenue une nécessité pour l'entreprise dans le monde du **B2B**. C'est justement ce que les services Web apportent par rapport aux solutions dites monolithiques. D'une certaine façon, le modèle des services Web est une évolution du modèle des composants distribués rendu nécessaire par l'utilisation intensive de l'Internet.

Cependant la solution des services Web repose sur l'ubiquité de l'infrastructure d'Internet alors que les autres architectures reposent chacune sur sa propre infrastructure. L'interopérabilité est donc une caractéristique intrinsèque aux services Web.

Définir les services Web nécessite alors d'introduire à la fois leurs architectures et leurs infrastructures. Dans ce qui suit, nous proposons une définition des services Web puis nous exposons l'architecture ainsi que l'infrastructure nécessaire.

1.2. Définition et Concepts

1.2.1 Origines

Les Services Web sont considérées comme étant l'évolution naturelle du Web. Ils s'inscrivent dans la continuité d'initiatives telles que CORBA (Common Object Request Broker Architecture, de l'OMG) en apportant toutefois une réponse plus simple, s'appuyant sur des technologies et standards reconnus et maintenant acceptés de tous.

B2B et **B2C** désignent respectivement des plates-formes de gestion des échanges commerciaux entre entreprises ou d'une entreprise vers ses clients par l'intermédiaire de l'Internet.

On peut distinguer trois phases de développement dans l'histoire du Web :

- **Le Web du Document**, le phénomène Internet originel, utilisé principalement par des organisations et des particuliers pour publier des informations sur leur travail, leurs produits, etc.
- **Le Web Applicatif**, le progrès grâce auquel les entreprises ont commencé à utiliser le Web à des fins commerciales. Les sites Internet sont devenus plus interactifs, et plus complexes, gérés par des serveurs d'applications, capable de distribuer leur charge avec d'autres serveurs, en fonction des besoins.
- **Le Web des Services** est la phase émergente, dans laquelle les serveurs d'application précédents communiquent désormais entre eux. Cette évolution a été poussée par le désir de pouvoir réaliser des échanges interentreprises dans un environnement automatisé et ouvert tel qu'Internet. L'échange de données informatisées entre deux applications nécessite une normalisation des messages échangés.

Une approche Web Services du système d'information vise à transformer chaque composant, base de données, applicatif métier, application de bureautique, en noeud s'exposant sur des standards de l'Internet, pour soit consommer des Web Services, soit pour en fournir. Ainsi, on passe d'une interopérabilité applicative à une interopérabilité entre services.

Cette approche propose une API (Application Programing Interface) universelle qui ne requiert pas d'autres protocoles que ceux d'Internet : HTTP (HyperText Transfer Protocol) [W3C06] sur TCP/IP[1] (Transmission Control Protocol / Internet Protocol) principalement. De plus, ils normalisent l'appel, l'échange et l'organisation de services applicatifs.

1.2.2. Définition des services Web

Dans la littérature, il existe de nombreuses définitions des Web services. Cette prolixité montre que la notion de service Web a besoin d'être éclaircie, et motive des travaux de recherches.

En 2001, Nagy et al. écrivent dans [CNW01] : "Un service Web est une application accessible à partir du Web. Il utilise les protocoles Internet pour communiquer et utilise un langage standard pour décrire son interface."

1 Les deux protocoles de communication qui forment les fondements de l'Internet

Les services web peuvent être définis comme des applications (programme) auto descriptives, modulaires, indépendantes et faiblement couplées qui fournissent un modèle simple de programmation et de déploiement d'applications, basé sur des normes s'exécutant à travers l'infrastructure web [DAN03], et qui peuvent être découvertes et invoquées dynamiquement via Internet ou un intranet par d'autres services. Ils sont définis par un ensemble de standards qui permettent aux applications de faire appel à des fonctionnalités (décrites par les services web) à distance (soit sur le même réseau, soit sur Internet) en simplifiant ainsi l'échange de données et de dialoguer à travers le réseau, indépendamment de leur plate-forme d'exécution et de leur langage d'implémentation. On peut simplement dire qu'un service Web est un service offert par l'intermédiaire du Web. Autrement dit, le Web est le média de communication utilisé, et tous les services accessibles via ce média peuvent être qualifiés de services Web [DAN03].

Grâce aux services Web, les entreprises peuvent encapsuler leurs procédés métiers et les publier comme des services, chercher et souscrire à d'autres services et échanger des informations au-delà des frontières des entreprises [BHI05].

Les services Web sont un paradigme naissant qui vise à la transposition des architectures par composant dans le cadre du Web. Un service Web est un composant logiciel qui offre des services à travers une interface standardisée. La particularité des services Web réside dans le fait qu'elle utilise la technologie Internet comme infrastructure pour la communication entre les composants logiciels (les services Web) et ceci en mettant en place un cadre de travail basé sur un ensemble de standards.

La notion de « service Web » désigne essentiellement une application (un programme) mise à disposition sur Internet par un fournisseur de service, et accessible par les clients à travers des protocoles Internet standards [FBM02; CAS01].

Des exemples de services actuellement disponibles concernent les prévisions météorologiques, la réservation de voyage en ligne, la vérification en ligne du solde d'un compte bancaire, des services d'achat de livres, etc.

L'avantage des services web, par rapport aux autres approches de systèmes distribués tel que RMI (pour Remote Method Invocation), réside dans leur support des pare-feux, mais surtout dans leur articulation autour d'XML (standard promulgué par le W3C), ce qui leur procure l'avantage d'être non propriétaire et ainsi multi plateforme.

Le consortium W3C[1] définit un service Web comme étant une application ou un composant logiciel qui vérifie les propriétés suivantes :

- Il est identifié par un URI ;
- Ses interfaces et ses liens (*binding*) peuvent être décrits en XML ;
- Sa définition peut être découverte par d'autres services Web ;
- Il peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant des protocoles Internet **[KEL03].**

L'objectif ultime de l'approche services Web est de transformer le Web en un dispositif distribué de calcul où les programmes (services) peuvent interagir de manière intelligente en étant capables de se découvrir automatiquement, de négocier entre eux et de se composer en des services plus complexes **[FBM02, MCI01].** En d'autres termes, l'idée poursuivie avec les services Web, est de mieux exploiter les technologies de l'Internet en substituant, autant que possible, les humains qui réalisent actuellement un certain nombre de services (ou tâches), par des machines en vue de permettre une découverte et/ou une composition automatique de services sur l'Internet. L'automatisation est donc un concept clé qui doit être présent à chaque étape du processus de conception et de mise en oeuvre des services Web. Elle est essentielle pour intégrer les facteurs suivants **[FBM02; CAS01]**:

- Passage à l'échelle : il faut être capable de traiter un nombre important de services Web (annuaire de services au niveau mondial).
- Forte réactivité dans un environnement hautement dynamique.
- Réduction des coûts de développement et de maintenance des services Web.
- Forte adaptabilité facilitant la maintenance et l'évolution des services Web.
- Prise en compte de critères de qualité de services aussi bien d'un point de vue qualitatif que quantitatif **[MAX01]**.

1.3. Architecture des services Web

Besoin en architecture : Les architectures en informatique proposent des schémas directifs et une vision sur le fonctionnement et la dynamique des systèmes. Pour promouvoir l'interopérabilité et l'extensibilité du paradigme des services Web, **une architecture de référence** est nécessaire afin de préserver les objectifs initiaux visés par les services Web lors des évolutions technologiques successives.

1 World Wide Web Consortium : http ://www.w3.org/2002/ws/

Une architecture orientée services SOA : (Service Oriented Architecture) : L'architecture des services Web comme l'architecture du Web sont des instances d'architectures orientées services (SOA) **[BOO03]**. Elle propose une perspective globale sur le développement, la gestion et le fonctionnement des services Web **[BAR03].** L'architecture SOA est un modèle (abstrait) qui définit un système par un ensemble d'agents logiciels distribués qui fonctionnent de concert afin de réaliser une fonctionnalité globale préalablement établie **[HEA01].**

Elle consiste à diviser le logiciel répondant à un problème, en un ensemble d'entités proposant des services. Chacune de ces entités peut utiliser les services proposés par d'autres entités. On obtient ainsi un réseau de services interagissant entre eux. Cette architecture s'appuie sur une architecture à composants (implémentation « réelle » des services **[HUM04]**) et suit l'évolution logique des architectures logicielles **[END et al 04]** (**figure 1**) :

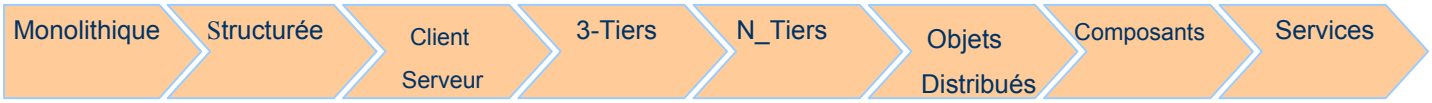


Figure 1 - Evolution des architectures logicielles.

Le choix d'une architecture SOA entre dans la perspective de transformer le Web en une énorme plate-forme de composants faiblement couplés et automatiquement intégrables. L'architecture SOA vise trois objectifs importants **[SW4]** : (i) identification des composants fonctionnels, (ii) définition des relations entre ces composants et (iii) établissement d'un ensemble de contraintes sur chaque composant de manière à garantir les propriétés globales de l'architecture.

Modèle de fonctionnement

Les interactions entre les services Web impliquent trois participants (acteurs) **[KEL03]**: le fournisseur de services (prestataire), l'annuaire de services et le client (utilisateur du service) (voir ***figure2***).

- Le fournisseur de services correspond au propriétaire du service (i.e. entité responsable du services Web). D'un point de vue technique, il est constitué par la plate-forme d'accueil du service.
- Le client correspond au demandeur du service. D'un point de vue technique, il est constitué par l'application de recherche et d'invocation d'un service. L'application client peut être elle même un service Web.

- L'annuaire des services représente l'entité logicielle qui joue le rôle de l'intermédiaire entre les clients et les fournisseurs de services, il correspond à un registre de descriptions de services offrant des facilités de publication de services pour les fournisseurs ainsi que des facilités de recherche pour les clients et le moyen de localiser leurs besoin en terme de services.

La dynamique de l'architecture se décompose ainsi : Le fournisseur de services définit la description de son service et la publie dans un annuaire de service (dans des registres) en vue d'être localisé par des clients; Le client utilise les facilités de recherche disponibles au niveau de l'annuaire pour retrouver et sélectionner un service donné. Il examine ensuite la description du service sélectionné (extrait sa description du registre) pour récupérer les informations nécessaires lui permettant de se connecter au fournisseur du service et d'interagir avec l'implémentation du service considéré (entreprend une interaction) **[BHI05].**

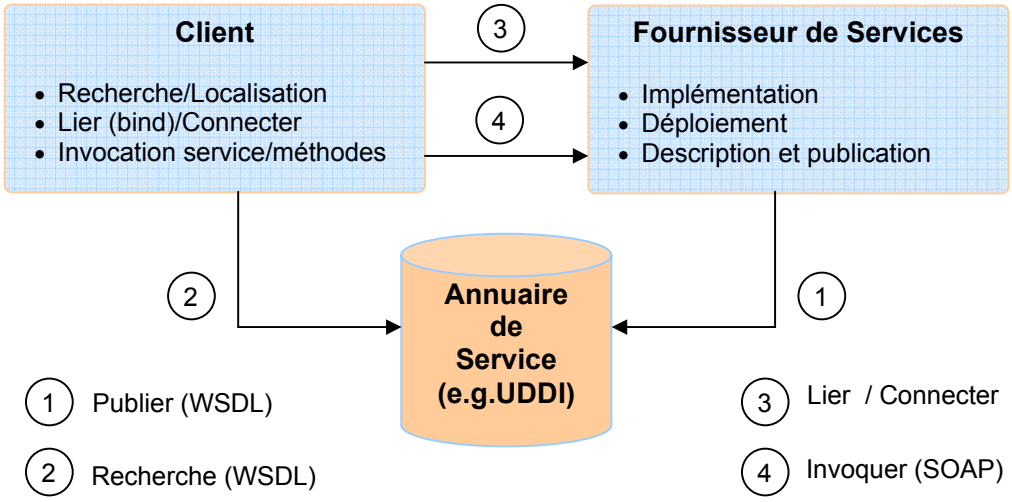


Figure 2 - Les interactions entre les services Web.

Les interactions entre les services Web impliquent trois intervenants: le fournisseur de services, l'annuaire de services et le client.

Le modèle de fonctionnement de l'architecture des services Web se base sur un cadre technologique qui constitue l'infrastructure de l'architecture par composants des services Web. Cette infrastructure offre les services nécessaires pour la réalisation des différentes étapes du cycle de vie d'un service Web.

1.4. L'infrastructure des services Web : les standards mis en jeu dans l'architecture

L'originalité de l'infrastructure des services Web consiste à mettre en place ces services en se basant exclusivement sur les protocoles les plus répandus d'Internet.

Pour garantir l'interopérabilité des trois opérations précédentes (publication, recherche et lien), l'infrastructure services Web s'est concrétisé autour d'un ensemble de spécifications considérées comme des standards pour chaque type d'interactions:

Un protocole abstrait de description et de structuration des messages, SOAP[1] **[SOA00]**, une spécification XML qui permet la publication et la localisation des services dans les annuaires, UDDI[2] **[UDD02]** et un format de description des services Web publiées dans les annuaires, WSDL[3] **[WSD01]**.

1.4.1. Rôle de XML dans l'infrastructure services Web

La technologie XML (*eXtensible Markup Language*), standardisée par le W3C (*World Wide Web Consortium*) en 1998 est aujourd'hui largement reconnue, acceptée et utilisée par de nombreuses entreprises comme format universel d'échange de données. On retrouve dans XML une généralisation des idées contenues dans HTML et SGML (*Standard Generalized Markup Language*). Reposant sur un système de balises au sein d'un fichier texte, XML peut être employé pour exprimer n'importe quel type d'information. On le retrouve par exemple aussi bien comme élément de sauvegarde de documents au sein de fichiers ou de bases de données ou encore comme format d'échange de données. **[DAN03]**

Dans HTML, on n'utilise les balises que pour décrire l'aspect graphique que doit revêtir la page dans le navigateur Web. Dans XML, les balises permettent d'associer toutes sortes d'informations au fil du texte.

XML est aujourd'hui un standard qui permet de décrire des documents structurés transportables sur les protocoles communs d'Internet. Il constitue la technologie de base des architectures services web, il est un facteur important pour contourner les barrières techniques. En effet, il apporte à l'architecture l'extensibilité et la neutralité vis à vis des plates-formes et des langages de développement. De plus, grâce à la structuration, XML permet la distinction entre les données des applications et les données des protocoles permettant ainsi une correspondance facile entre les différents protocoles.

1. Simple Object Access Protocol.
2. Universal Description, Discovery and Integration.
3. Web Services Description Language.

L'interopérabilité entre les systèmes hétérogènes demande des mécanismes puissants de correspondance et de gestion des types de données des messages entre les fournisseurs et les clients. Pour les services Web, on utilise systématiquement XML avec les Namespaces et la spécification XML Schema, tous deux indispensables pour exprimer les structures des données habituellement complexes figurant dans les messages échangés.

1.4.2. La communication : SOAP

Le formalisme XML étant basé sur du texte, il est tout à fait adapté pour être véhiculé par le protocole de communication HTTP. Pour assurer la communication entre les participants d'un service Web, le couple HTTP/XML est donc naturellement utilisé. Les messages exprimés à l'aide de XML respectent un format standard définis par le W3C que l'on appelle le protocole SOAP (*Simple Object Access Protocol*).

Ainsi, dans le cadre des services Web, une application communique avec une autre application par l'intermédiaire du protocole SOAP/HTTP. Cela signifie qu'une application envoie un message SOAP (donc un message exprimé en XML) vers une autre application, que celle-ci traite la demande effectuée par ce message et renvoie un message de réponse à l'application appelante, basant sur le mécanisme RPC (*Remote Procedure Call*: appel de procedures localisées sur des machines distantes) qui simplifie la coopération entre applications en fournissant un mécanisme pour transmettre un appel de procédure vers une application distante offertes sur le Web. **[DAN03]**

Le protocole de communication HTTP

Le choix de HTTP est un élément très important car il illustre parfaitement la capacité de SOAP à s'adapter aux échanges sur Internet réutilisant une technologie largement déployée et acceptée. Le protocole HTTP est un protocole simple capable de s'accommoder à la fois de la qualité de service et des temps de latence très variables de l'Internet; ce que n'est pas le cas par exemple des protocoles d'environnements répartis tels que DCOM ou CORBA.

De plus, en s'appuyant sur HTTP, on ne se heurte pas aux problèmes de pare-feu (*firewall*) ou de configuration de réseau IP qui rendent parfois difficile le déploiement d'applications à objets répartis au-delà du périmètre du réseau d'entreprise. **[CHA02]**

Les messages SOAP

La communication par message constitue un point crucial dans toute architecture SOA. SOAP est au coeur de l'échange des messages entre applications sur le Web. Du fait qu'il est basé sur XML, il permet l'échange de données structurées indépendamment des langages de programmation ou des systèmes d'exploitation. Le protocole SOAP est une spécification XML qui définit un protocole léger d'échange de données structurées entre services Web dans un

environnement totalement distribué et hétérogène. Il est indépendant du contenu du message, à proprement parler, il laisse la responsabilité de l'interprétation aux couches de communication supérieures **[GHM00]**. Il se contente d'offrir la possibilité de structurer des messages destinés à des objectifs particuliers allant d'un simple échange de données jusqu'à l'appel de procédures à distance.

N'imposant aucun modèle de programmation spécifique, SOAP peut être employé dans tous les styles de communication : synchrone (s'appuyer sur HTTP) ou asynchrone (s'appuyer sur SMTP[1]), point à point ou multipoint, intranet ou Internet.

Modèle d'échange de messages en SOAP

Les messages SOAP sont des transmissions fondamentalement à sens unique (unidirectionnel) d'un expéditeur à un récepteur **(figure3)**. Lorsqu'une transmission d'un message commence (e.g. invocation d'un service *web)*, un message SOAP (ou document XML) est généré. Ce message est envoyé à partir d'une entité appelée le SOAP *sender,* localisé dans un SOAP nœud, celui-ci permet d'appeler une ou plusieurs fonctions du service. Les paramètres des fonctions invoquées se situent à l'intérieur du message. Le message est transité par zéro ou plusieurs noeuds intermédiaires (SOAP *intermediates*) et le processus fini lorsque le message arrive au SOAP *receiver*. Le message est alors traité par le service Web avec les paramètres correspondants. Un message SOAP, de même structure, est retourné. Mais les paramètres transportés représentent les résultats de la fonction invoquée. Le chemin suivi par un message SOAP est nommé *message path.* **[RIC04]**.

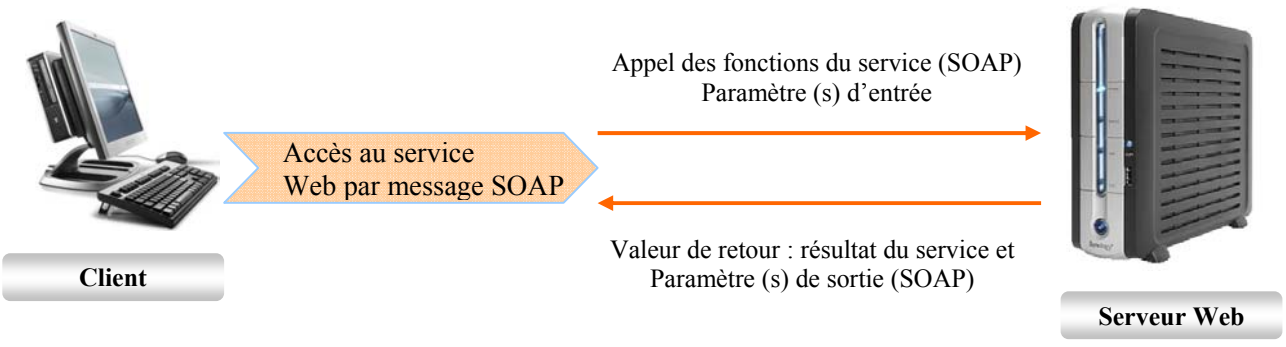


Figure 3 - Schéma de fonctionnement du système unidirectionnel SOAP

1. SMTP Simple Mail Transfer Protocol protocole de la famille TCP/IP utilisé pour le transfert de courrier électronique

Enveloppe SOAP: Un message SOAP est un document XML composé d'un élément *envelope* qui est appelé "envelope SOAP", contenant un en-tête (*header*) facultatif et un corps (*body*) du message obligatoire. (**figure 4**).

L'enveloppe définit l'espace de noms (*namespace* : URI permettant de connaître la provenance de chaque balise) précisant la version supportée de SOAP. Cet espace de noms est standard et permet de différentier les éléments du schéma. e.g:

"http://schemas.xmlsoap.org/soap/envelope/ ".

La seconde partie inclut l'espace de noms concernant les conventions de codage. Cette partie est facultative. Le document devra alors suivre le schéma défini dans cet espace de noms. Ce namespace peut être : " http://schemas.xmlsoap.org/soap/encoding/ ". La définition type d'une enveloppe est donnée par l'**Extrait de code 1** :

```
<soap :Envelope xmlns :soap=http://schemas.xmlsoap.org/soap/envelope/
xmlns :enc="http://schemas.xmlsoap.org/soap/encoding/">

<soap :Header>…</soap :Header>
<soap :Body>…</soap :Body>
                                        </soa
                                            .
```

Extrait de code 1 Définition type d'envelope SOAP

SOAP Header : l'en-tête de message SOAP, est le premier fils de l'élément *envelope*. Même s'il peut être vide, il doit impérativement être écrit. Cet élément apporte des données supplémentaires à SOAP. Ces données peuvent être des informations d'authentification, de gestion de transactions, de paiement, etc.

SOAP Body : l'élément *Body* contient l'information destinée au receveur. Il faut que cet élément encadre une balise contenant le nom de la méthode invoquée (pour une requête), ou le nom de la méthode suivie de *Response* (pour la réponse). Cette balise doit aussi contenir l'espace de noms correspondant au nom de service.

Un message SOAP peut aussi contenir un ou plusieurs attachements (document, images, etc.) à transmettre avec le message. Ces données ne sont pas représentables en XML.

De ce fait, SOAP utilise un mécanisme d'inclusion appelé MIME (Multipurpose Internet Mail Extensions), cette méthode est répandue pour transmettre des documents autres que du texte dans des courriers électroniques.

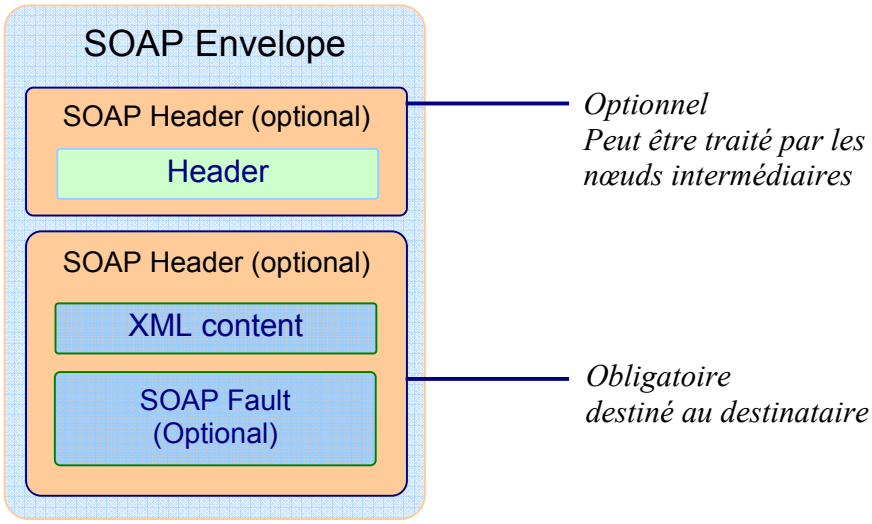


Figure 4 - La structure des messages SOAP

Pour résumer, un message SOAP en cours de transit est composé de deux parties indépendantes :

- une structure XML qui constitue le message SOAP;
- un en-tête du protocole de transport où le message SOAP est encapsulé en vue d'être livré à l'application destination;

Nous pouvons cependant ajouter que le protocole SOAP n'intègre pas la sémantique du message. Pour remédier à cela, le langage RDF technologies du Web sémantique, permet de supporter les échanges de messages.

1.4.3. Description de service : WSDL

Le protocole SOAP met à la disposition des services Web un moyen standard de structuration et d'échange de messages XML. Il ne fournit aucune indication sur la structure que le message doit respecter vis à vis du service Web sollicité. C'est toujours dans le but de rendre les services Web faiblement couplés et autonomes, que la spécification WSDL a vu le jour. Contrairement aux architectures monolithiques où la description des composants ainsi que les moyens de les invoquer dépendent fortement de l'infrastructure utilisée, la spécification WSDL offre une grammaire qui décrit l'interface des composants services Web de telle façon qu'ils se suffisent à eux-mêmes.

WSDL est un langage de description des capacités de services Web basé sur XML. Un document WSDL décrit essentiellement le nom de la méthode utilisé, son nombre de paramètres, et leur type, ce qu'un service Web offre, où il réside et comment on peut l'invoquer. Nous pouvons dire que les services Web sont auto descriptifs. **[BHI05]**

La spécification du service est composée de deux parties : une définition abstraite des services en terme de messages échangés entre les différents types de port et la définition des mécanismes de liaison entre les définitions abstraites et un ensemble de techniques de déploiement (généralement des protocoles Internet).

Un document WSDL est devisé en sept sections distinctes (**figure 5**) :

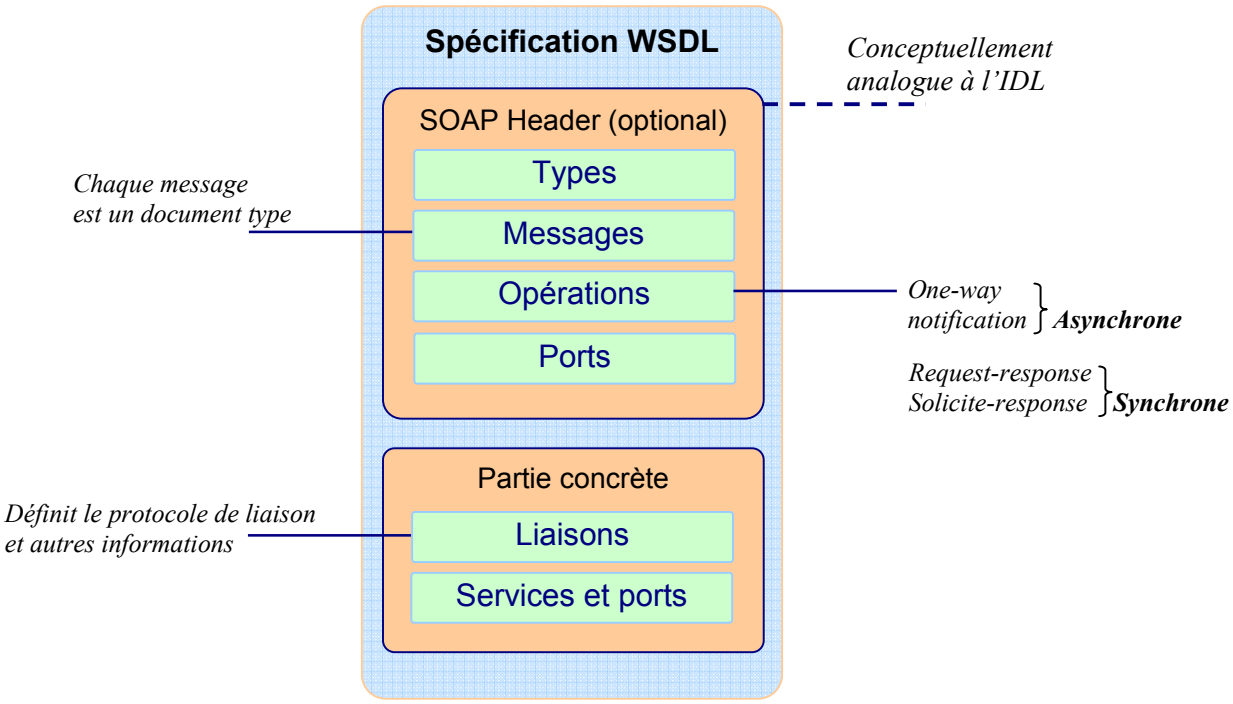


Figure 5 - *Structure d'une interface WSDL*

types : Fournissent des définitions de types de donnée afin de décrire les messages échangés.
message : Par l'intermédiaire de cette section, on définit le format des messages échangés, Un message correspond aux données qui seront véhiculées selon les méthodes invoquées. Chaque méthode du service possède deux éléments message, le premier correspond à la requête et le second correspond à la réponse. Chacun est constitué d'une ou plusieurs parties (sous-éléments **part**) décrivant un élément du message. La description contient le nom de l'élément en paramètre d'entrée ou de sortie selon le message et son type.

type de port (*portType*) : Ensemble d'opérations abstraites. Chaque opération se réfère à un message entrant et à des messages sortants. Chaque opération est identifiée dans le document WSDL par l'élément operation déclarant le nom de la méthode et les données passées en paramètres. Ces opérations sont regroupées dans un élément portType. Le document WSDL est constitué d'autant d'élément portType que le service contient de groupe de méthodes.
opération : Décrit les opérations invoquées à l'aide des messages reçu, émit par le service et éventuellement des messages d'erreur.
port : Ce qui spécifie une adresse pour un rattachement, déterminant ainsi un seul noeud branché, ou point terminal [1] (*endpoint*).
rattachement (*binding*) : Ce qui spécifie les aspects concrets de protocole de communication et le format des données pour les opérations et messages définis par un type de port particulier.
service : Représente une collection de points d'entrée (endpoint) relatifs, il sert à regrouper un ensemble de ports.

La spécification WSDL joue un rôle important dans l'interopérabilité des composants services Web. Moyennant un schéma uniforme obéissant à une sémantique bien définie, elle permet aux composants de définir ce qui est nécessaire à leur invocation. La spécification WSDL est définie selon une sémantique totalement indépendante du modèle de programmation de l'application. Elle sépare clairement la définition abstraite du service (échange de messages) de ses mécanismes de liaison (définition des protocoles applicatifs).

Cette dernière caractéristique permet au composant d'interagir même si l'application a été modifiée ce qui est un point important pour assurer l'interopérabilité des services.

La complétude de la spécification WSDL permet l'automatisation du processus d'invocation.

1.4.4. Service recherche et publication : UDDI (Universal Description, Discovery and Integration)

Les deux standards exposés précédemment définissent ensemble l'aspect le plus basique de développement de l'infrastructure des services Web. Toutefois, dans un environnement ouvert comme Internet, le modèle de description des services Web n'est d'aucune utilité s'il n'existe pas un moyen de localiser aussi bien les services que leurs descriptions WSDL. Un troisième standard a été conçu pour réduire l'écart entre les applications clientes et les services Web, appelé UDDI **[BCR02].**

---

1 Tout emplacement pouvant être rejoint au moyen d'une adresse distincte dans un réseau (e.g: http ://services.xmethods.net :80/perl/soaplite.cgi)

UDDI est un annuaire mondial d'entreprises s'appuyant sur le réseau Internet. Il permet d'automatiser les communications entre prestataires, clients, etc. Dans ce but, celui-ci propose plusieurs entrées : nom, carte d'identité des sociétés, description des produits et services. Et, si ces derniers sont des applicatifs invocables à distance, il fournit les références des connexions permettant de communiquer avec eux.

UDDI est une spécification qui définit les mécanismes qui permettent aux entreprises de publier leurs services et de découvrir et interagir avec d'autres services via le Web. UDDI se base sur SOAP et suppose que les requêtes et les réponses sont des objets UDDI envoyés comme des messages SOAP **[BHI05].**

L'enregistrement des *services Web* dans un annuaire *UDDI* s'effectue auprès d'un opérateur en accédant au site Web de ce dernier à partir d'un navigateur ou d'un outil intégré à un environnement de développement. Des recherches précises peuvent s'effectuer dans l'annuaire par catégories d'entreprise en utilisant des standards taxinomiques d'identification d'entreprise et par mots clés.

La spécification UDDI constitue une norme pour les annuaires des services Web. Les fournisseurs disposent d'un schéma de description permettant de publier des données concernant leurs activités, la liste des services qu'ils offrent et les détails techniques sur chaque service. Elle offre aussi une API aux applications clientes, pour consulter et extraire des données concernant un service et/ou son fournisseur (Interrogation de service).

Les registres UDDI sont des services Web qui offrent deux fonctionnalités de base : la publication des différents types d'information sur les services et leurs fournisseurs selon un schéma de description, et la consultation du contenu des registres.

La publication d'un service chez un opérateur, donne lieu automatiquement à un processus de propagation des informations aux différents registres UDDI. L'accès à l'ensemble d'informations des registres peut se faire de n'importe quel opérateur UDDI. La recherche se fait grâce à un moteur de recherche intégré au site de l'opérateur UDDI choisi. Ce moteur de recherche permettra d'affiner la recherche selon plusieurs critères : Nom de l'entreprise, la localisation de l'entreprise, identifiant de l'entreprise, le nom du service Web …etc.

Chaque registre UDDI stocke trois sortes de données : des données concernant les fournisseurs de services telles que le nom de l'entreprise, ses coordonnées et des descriptions de l'entreprise consultable par l'utilisateur appelées *pages blanches*, des données concernant l'activité ou le service métier des fournisseurs, la description des *services Web* déployés par les entreprises appelées pages jaunes et les données techniques de chaque service publié qui constituent les pages vertes. La spécification UDDI offre un schéma qui structure d'une manière uniforme les différents types concernant les trois niveaux de description.

Le fonctionnement d'un UDDI est décrit par les modèles suivants:

- Modèle de données (**figure 6**): Des différents composants sont des documents XML : *PublisherAssertion*, *Business Entity*, *Business services*, *BindingTemplate*, et *Tmodel*. **Publisher Assertion** : Cette partie est facultative. Elle permet de décrire l'organisation dans son intégrité si elle est divisée en plusieurs divisions. **[DAN03]**
  BusinessEntity: Décrivent les organisations ayant publié des services dans le répertoire BusinessKey;
  BusinessService: Décrivent de manière non technique les services;
  BindingTemplates Spécifient les coordonnées des services;
  Tmodel : Décrivent de manière technique les services;

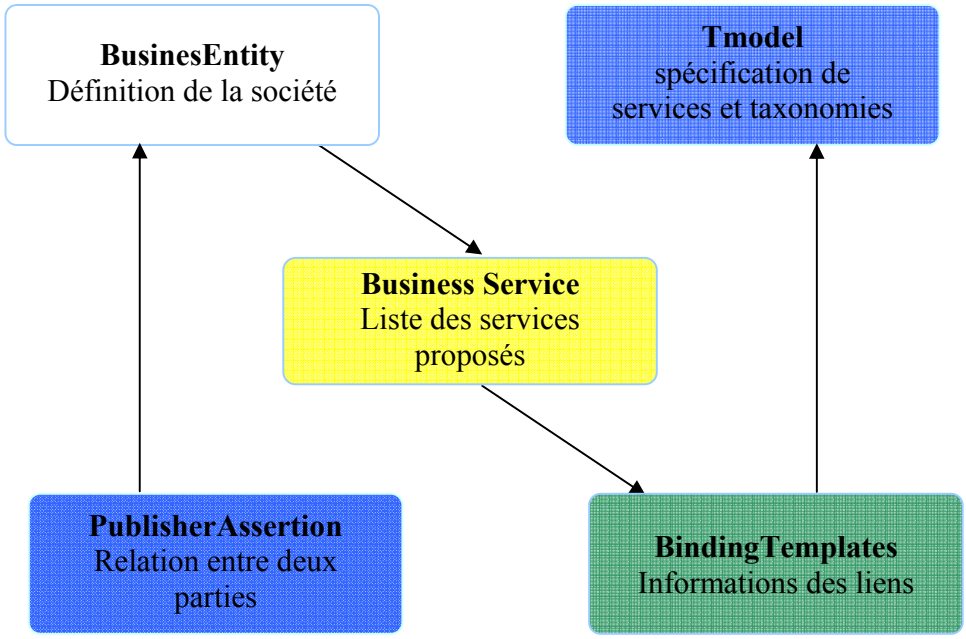


Figure **6** *Architecture de UDDI (Entités composant un annuaire*).

- Modèle de programmation: L'API UDDI est divisée en une interface de programmation pour l'enregistrement de services Web dans un annuaire UDDI et une interface de programmation pour la recherche d'informations. Cette API est composée de 2 grandes bibliothèques :
  - API de requête (interrogation du Registre);
  - API de publication;
- Modèle d'usage: Publication de service, Découverte de service, Description d'utilisation de service.

1.5. Conclusion sur les standards

En conclusion, les standards proposés sont de bas niveau sans véritable sémantique. Cependant, ces standards permettent un certain nombre de tâches automatiques qui induisent une sémantique: diagnostic et découverte de protocoles pour SOAP et composition automatique pour WSDL. Comme il n'y a pas de véritable interaction entre ces trois standards, des couches ont été ajoutées afin de remédier à ce problème. Si ces couches sont bien utilisées elles peuvent contribuer à ajouter de la sémantique en permettant d'effectuer des tâches automatiques. En pratique WSDL et SOAP sont seulement utilisés. En pratique aussi d'autres outils correspondant à des couches de plus haut niveau sont utilisés même s'ils ne sont pas considérés comme des standards.

1.6. Architecture étendue

Actuellement, SOAP, WSDL et UDDI sont les trois standards qui constituent l'architecture des services Web. Ensemble, ils résolvent les problèmes de l'hétérogénéité des systèmes pour l'intégration d'applications en ligne **[TEA01].** Cependant, une application **B2B** nécessite d'invoquer un ensemble de services dans un ordre précis et selon une logique bien définie. Or SOAP, WSDL et UDDI ne s'intéressent pas à ce problème et se situe plutôt au niveau transport et données. Toutefois, il existe d'autres aspects essentiels à mettre en oeuvre avant de parler d'automatisation de processus de découverte et d'intégration des services Web.

A ce propos, plusieurs technologies ont été proposées. Ces technologies s'intéressent à différents problèmes et à différents niveaux, des propriétés non fonctionnelles (comme la sécurité et la fiabilité) au niveau transport à la qualité de services au niveau procédé.

L'architecture étendue (avancée) est constituée de plusieurs couches se superposant les unes sur les autres, d'où le nom de pile des Web services. La ***figure 7*** décrit un exemple d'une telle pile.

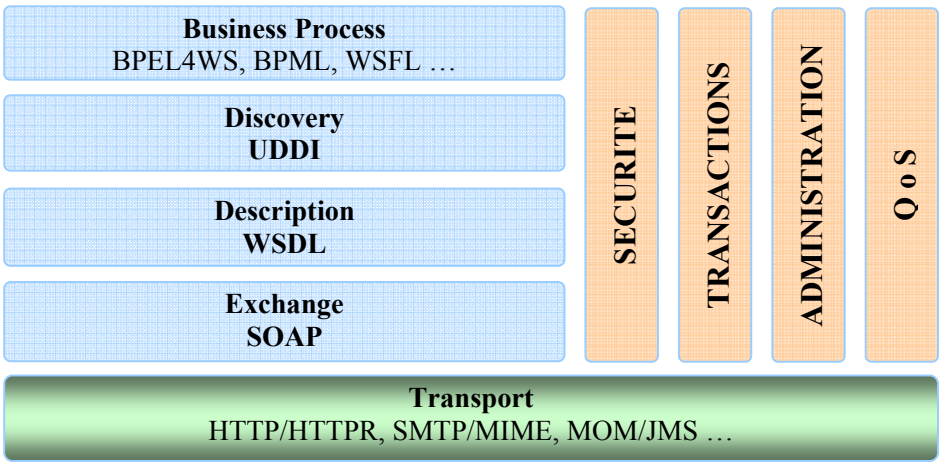


Figure 7 - Architecture en pile (étendue).

La pile est constituée de plusieurs couches, chaque couche s'appuyant sur un standard particulier. On retrouve, au-dessus de la couche de transport, les trois couches formant l'infrastructure de base décrite précédemment. Ces couches s'appuient sur les standards émergents SOAP, WSDL et UDDI. Comme mentionné précédemment, l'infrastructure de base définit les fondements techniques permettant de rendre les business processes accessibles à l'intérieur d'une entreprise et au-delà même des frontières d'une entreprise. Dans ce contexte deux types de couches permettent de la compléter : (i) les couches dites **transversales [GOT et al02]** (*e.g:* sécurité, administration, transactions et qualité de services (QoS)) rendent viable l'utilisation effective des services Web dans le monde industriel ; (ii) une couche **Business processus** permet l'utilisation effective des services Web dans le domaine du e-business.

1.7. Problématique de recherche sur les services Web

Des recherches faites sur les services Web par rapport à SOA couvrent plusieurs problèmes intéressants : la Sélection des services Web, la Découverte et Découverte dynamique des services Web, la Composition des services Web… etc.

1.7.1. Sélection des servies Web

Avec la sélection des services web, on cherche à choisir le meilleur fournisseur d'un service web, étant donné un ensemble de fournisseurs de ce service. Un nouveau modèle de registre et découverte de services web a été définit basé sur la Qualité de Service (i.e. QdS en anglais *Quality of Service - QoS*) **[SHU03]**. La plate-forme de cette proposition est constituée de quatre rôles: Fournisseur de services web; Consommateur de services Web; *Certificateur* de la Qualité de Service et le nouvel registre.

Le fournisseur du service offre le service Web en publiant ce dernier dans le nouveau registre; le consommateur a besoin du service web offert par le fournisseur ; le nouvel registre UDDI est un lieu de stockage de services web enregistrés avec facilités de recherche. Le certificateur QdS sert à vérifier les revendications de qualité de service pour un service Web. Le nouveau registre est différent du actuel modèle UDDI en ayant information sur la description fonctionnelle du service Web, et sur la qualité de service enregistré dans le stockage. La consultation pourrait être faite selon la description fonctionnelle du service Web désirable, avec la qualité exigée de service attribue comme des contraintes de consultation.

1.7.2. Découverte de services Web

Dans le contexte d'une application qui a besoin d'exécuter une fonctionnalité implémentée comme un service web par plusieurs fournisseurs, la découverte fait référence au processus de recherche des services web implémentant la fonctionnalité souhaitée. Les registres UDDI sont des entités qui servent d'appui à la découverte de services web pour les

applications client. De cette façon une application interroge un registre UDDI pour les fournisseurs d'un service web.

OWL-S est un langage qui définit une ontologie de services web [MAR03]. Il est basé sur le langage OWL [MVH03]. Les intérêts liés à l'utilisation de OWL-S sont que ce langage inclut la sémantique et contient des fonctions indispensables à la mise en oeuvre de services Web : la description, la recherche et l'invocation de services. Il permet de réaliser les deux tâches suivantes :

1) Découverte automatique de services Web : Cette tâche est possible parce que OWL-S permet d'exprimer et de résoudre des requêtes avec contenu sémantique. Par exemple une requête comme : « Trouver des services Web qui vendent des tickets d'avion entre deux villes spécifiques et qui permettent de payer avec une carte de crédit particulière ».

2) Invocation automatique de services Web : OWL-S fourni un ensemble d'APIs pour que l'invocation à un service Web soit automatique.

OWL-S est composé de trois parties (figure.8) [MAR03]. Principales: *Le profil du service* : Pour faire de la publicité et découvrir des services; *Le modèle du processus* qui donne une description détaillé d'une opération du service; Le «*grounding*» qui fournit les caractéristiques techniques pour établir la communication avec le service au moyen de messages.

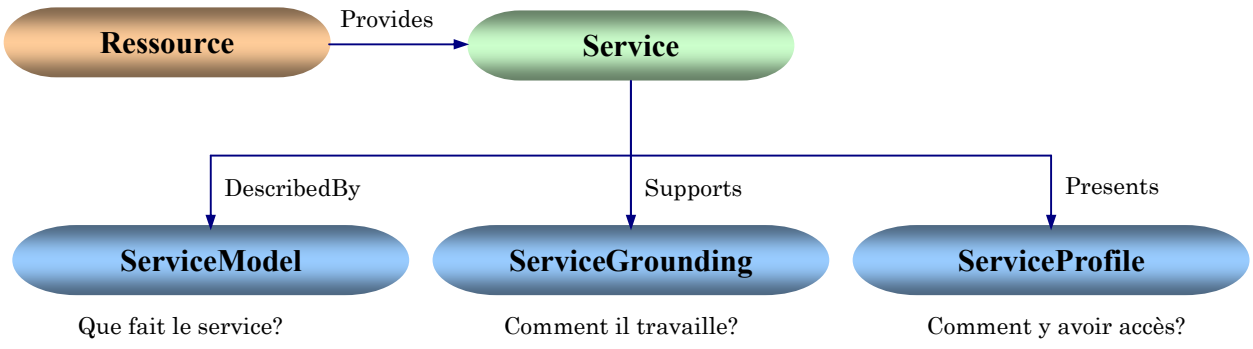


Figure 8 - Niveau supérieur de l'ontologie OWL-S

1.7.3. Découverte dynamique

La découverte dynamique d'un service Web s'effectue d'une façon différente. Au lieu de s'enregistrer dans un registre connu, les services Web découverts dynamiquement annoncent explicitement leur arrivée sur le réseau et leur départ. WS-Discovery définit des protocoles pour annoncer et découvrir des services Web via des messages de multi diffusion.

Lorsqu'un service Web se connecte à un réseau, il annonce son arrivée en émettant un message "Hello". Lorsqu'un service quitte un réseau, WS-Discovery précise qu'un message "Bye" doit être envoyé au réseau ou au proxy de découverte. Ce message informe les autres services du réseau que le service Web concerné n'est plus disponible.

La découverte dynamique étend aussi l'architecture de services Web à des équipements, comme les imprimantes et les systèmes de stockage qui peuvent être intégrés dans un système sous la forme de services Web, sans avoir besoin d'outils ou de protocoles spécialisés. **[CKB04]**

1.7.4. Composition de Services Web

«*Les composants sont destinés à être composés*» **[BRO96].** La *réutilisation* est un avantage important de l'approche par composants **[SZY97] [PFI96].** Les services Web, tels qu'ils sont présentés, sont conceptuellement limités à des fonctionnalités relativement simples qui sont modélisées par une collection d'opérations. Toutefois, pour certains types d'application, il est nécessaire de combiner un ensemble de services Web WSDL (services Web basiques) en services Web plus complexes (services Web agrégés ou composites) afin de répondre à des exigences plus complexes **[CSW01] [YAN01] [NAR02 a].**

La technologie "services Web" est la technologie clé permettant l'intégration des applications via le Web. L'objectif de la composition de service est de créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services existants, composés ou non en vue d'apporter une valeur ajoutée. 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. La création d'une application distribuée complexe peut être obtenue par la composition de services Web. Cependant, la création d'un service à partir d'autres services est loin d'être une tâche triviale. Pour aider les développeurs à créer des services composites, l'intergiciel de composition de services Web doit fournir une abstraction et une infrastructure qui facilitent la définition et l'exécution d'un service composite.

La composition de services constitue l'un des champs de recherche les plus actifs dans le domaine des services Web sur lequel nous nous concentrons dans notre travail.

1.8. Conclusion

L'essor incontestable du Web et plus particulièrement des services web a permis la naissance d'une nouvelle génération de systèmes caractérisés par une meilleure intégration d'applications hétérogènes et une meilleure communication entre ses différents composants. Les services Web regroupent tout un ensemble de technologies bâties sur des standards (SOAP, WSDL, UDDI, XML). Ils permettent de créer des composants logiciels distribués, de décrire leur interface et de les utiliser indépendamment de la plate-forme sur laquelle ils sont implémentés. La recherche dans ce domaine est très active et s'intéresse entre autres à la composition, l'orchestration, la sécurité, la sémantique des services web…etc.

La technologie des Services Web est aujourd'hui de plus en plus incontournable et se présente comme le nouveau paradigme des architectures logicielles. Cette technologie englobe de nombreux concepts et tend à s'imposer comme le nouveau standard en terme d'intégration et d'échanges B2B.

En résume, les services Web sont dotés de systèmes de communication (avec plusieurs couches) et ils sont basés sur des standards Internet et en dernière instance sur le langage XML qui permet de bien distinguer les données des protocoles, avec en l'occurrence, SOAP. L'apport principal des services Web est la solution au problème d'hétérogénéité par rapport aux plates-formes et langages des applications, et d'avoir un partage des fonctionnalités et facilite grandement le développement.

Les services qui sont utilisés dans la composition manuelle sont connus à l'avance et sa description est faite en accord avec ses paramètres. Dans un contexte automatique, les services qui appartiendront à la composition ne sont pas connus, ou pas complètement. Le chapitre 2 est dédié à ce type de composition, les compositions automatiques.

Chapitre 2

Composition de services Web sémantiques

Dans ce chapitre, nous présentons le problème de la composition automatique des services Web sémantiques (SWS). Nous évoquons plus particulièrement la notion de composition de SWS ainsi que ses différentes issues et classifications (types), puis nous présentons un ensemble d'approches de compositions et une étude comparative de ces différentes approches.

2.1. Introduction

Les services Web offrent une architecture par composants permettant à des applications d'offrir leurs fonctionnalités sous forme de services à travers des protocoles Internet universels. Ceci marque une évolution significative dans l'histoire d'Internet qui jusque là a été destiné à jouer le rôle d'un vecteur d'échange de données. Avec les services Web, Internet se transforme en une plate-forme de composants auto descriptifs, facilement intégrables et faiblement couplés **[CNW01]**. Comme toute innovation, l'apparition des services Web donne lieu à un ensemble d'opportunités et de nouvelles applications. Dans ce travail une motivation anime notre intérêt pour les services Web à savoir la composition des services Web.

Le problème de la composition de services Web se réfère à la construction des nouveaux services Web « services Web composite » à partir des services individuels existants. Dans cette partie nous présentons d'abord le besoin de la composition de services Web, puis nous faisons une synthèse précise des différentes approches de composition basées sur quelques plateformes et frameworks de composition actuellement existants. Les approches de composition seront particulièrement étudiées par leur capacité à garantir des propriétés de fiabilité (description syntaxique, protocole d'utilisation d'interface, sémantique comportementale, propriétés de synchronisation, définition de qualité des services…etc).

2.2. Services Web Sémantique

De nombreux langages **[MAR et al04]**, **[ARR04]**, **[PEE05]**, **[KKM05]** ou extensions de langages **[PAO et al02]**, **[MIL et al04]**, **[SW9]** ont pour objectif de résoudre les problèmes de sémantique rencontrés par les services Web. Ces langages intègrent la sémantique des données dans les descriptions des services Web, qui deviennent des services Web sémantiques. Dans cette partie nous expliquons l'apport de la sémantique dans la composition des services Web.

2.2.1. Description comportementale de services Web

Les services Web, tels qu'ils sont présentés dans le Chapitre 1, sont conceptuellement limités à des fonctionnalités. Une telle vue peut être décrite seulement soit par une perspective d'entrée sortie (Input/Output), ou en termes de description comportementale.

La perspective d'I/O est supportée par des standards tels que WSDL, et elle consiste à décrire un service en termes de ses exécutions exportées (opérations), sans spécifier des contraintes sur l'ordre d'exécution de ses opérations d'où la nécessité d'une description de son comportement qui est une liste de toutes les séquences d'exécution d'opérations supportée par le service Web.

2.2.2. Sémantique, Ontologie et services Web

Dans un environnement ouvert et dynamique, tel que Internet, il est évident qu'une composition à la demande sera très préférable, en plus il sera judicieux de prendre en charge la sémantique durant le matching (appariement) des services, afin de minimiser les fausses réponses, et d'améliorer la qualité globale des résultats.

Lors de la réalisation d'une composition, des conflits sémantiques apparaissent quand les services sont décrits selon des vocabulaires différents, ou lorsque la signification du vocabulaire utilisé diffère d'un service Web à un autre. Une description de service Web sémantique repose sur un vocabulaire commun décrit dans une **ontologie**, qui est définie comme "*une conceptualisation partagée des connaissances d'un domaine*" **[GRU00]**. Les ontologies ont été introduites pour résoudre le problème du manque d'expressivité, de complétude et de cohérence des descriptions précédentes (WSDL, UDDI). Elles doivent permettre à l'utilisateur de décrire des domaines explicitement et rigoureusement.

Une ontologie décrit le vocabulaire d'un domaine de connaissance, incluant les concepts utilisés dans le domaine et les relations entre ces concepts. Elle repose sur un langage de description, qui fournit les structures pour décrire les concepts et leurs relations. L'adoption d'une ontologie suppose un accord de tous les utilisateurs sur une représentation commune des connaissances d'un domaine, garantissant ainsi une interprétation identique des données. Dans **[ANT03]**, les auteurs citent les cinq principaux objectifs suivants des ontologies :

- une syntaxe bien définie ;
- une sémantique bien définie ;
- un support de raisonnement efficace ;
- une expressivité suffisante ;
- et une facilité d'expression.

Pour que l'interprétation des opérations WSDL soit possible, on peut annoter sémantiquement les services Web. Les outils d'annotation visent à améliorer l'appréhension des services web ainsi que l'échange, la communication et l'interopérabilité sur le Web.

Les annotations sémantiques permettent une meilleure recherche de services et facilitent le traitement intelligent d'une requête en utilisant des mécanismes d'inférence et des grandes ontologies. Elles consistent à associer une *sémantique*[1] à certains éléments de services en référence à une ontologie du domaine. Cette interprétation ontologique joue plusieurs rôles dans le processus global de découverte (recherche de service) : elle impose des contraintes sémantiques qui permettent de désambiguïser l'analyse syntaxique.

En faisant référence à quelques travaux de recherche **[COL et al99], [MOR06], [HER05]**, dans le domaine de la recherche d'informations, on peut inspirer l'idée du mécanisme d'annotation sémantique automatique de pages Web à partir d'une ontologie, pour annoter les services Web. L'annotation sémantique repose sur deux modules :

L'étiqueteur sémantique est utilisé pour attacher des étiquettes sémantiques aux éléments essentiels des services fournis en entrée, en projetant chaque élément qu'on veut annoter sur une ontologie du domaine. *Le module d'acquisition d'ontologies*, permet d'acquérir une ontologie spécialisée adaptée à chaque service. Ce dernier prend principalement en entrée des concepts de l'ontologie dont sont extraits certains types de relations de dépendance uniquement. Il fournit en sortie une ontologie sous la forme d'une hiérarchie multiple de classes sémantiques, c'est-à-dire des regroupements de mots sous un concept commun.

2.3. Composition de services Web sémantiques

Les services Web ont une interface qui indiquent leurs fonctionnalités et qui peuvent être appelées par d'autres systèmes par des interactions basées sur des messages. Dans certains cas, un besoin fonctionnel (et/ou non fonctionnel) ne peut pas être satisfait par un service Web simple, mais pourrait être probablement satisfait convenablement en intégrant et en composant un ensemble de services élémentaires et/ou composés.

Informellement, soit G le besoin d'un utilisateur (but), et $W = \{ W1, W2.... Wn \}$ l'ensemble des services Web disponibles. La composition de ces services Web revient soit à produire un nouveau service Web composé WG en réutilisant et en combinant un sous-ensemble des services disponibles $Wi...\otimes...Wj...\otimes...Wk$, où $\otimes$ est l'opérateur de composition, c-à-d développer un médiateur qui permettra la communication entre le client (un être humain ou un agent logiciel) et les services participants, ou bien établissant une structure de linkage $L = \{ Li\ j\ ,\ Lik, \ldots \}$ entre les services participants $Wi\ ,\ \ldots Wj\ ,\ \ldots Wk$, leurs permettant de communiquer directement par l'intermédiaire des échanges de messages.

1 Le terme « sémantique » est utilisé pour décrire ce qui a trait à la signification véhiculée par un objet, généralement un mot.

Cela se réfère à la composition basée sur la chorégraphie où les canaux ou liens d'échange de messages sont établis entre les services participants, ainsi que le client communique directement avec les services appropriés.

Le problème de la composition des services Web peut être réduit à trois problèmes fondamentaux: le premier est de faire un plan qui décrit comment les services Web interagissent et comment les fonctionnalités qu'ils offrent peuvent être intégrées pour fournir une solution à un problème. Le deuxième est la découverte des services Web qui exécutent les tâches exigées dans le plan. Le troisième est la gestion de l'interaction entre services Web. Le planning, la découverte et les interactions sont interconnectées: un plan spécifie le type de services Web à découvrir, mais il dépend aussi des services Web qui sont disponibles. De la même façon, l'interaction dépend des caractéristiques du plan, mais le plan lui-même dépend des exigences de l'interaction. Ces trois sous problèmes dictent ainsi un ensemble de défis que toute infrastructure de composition de service Web doit surmonter.

Pour découvrir un service Web, l'infrastructure devra être capable de représenter les capacités fournis par un service Web et doit être capable de reconnaître les ressemblances et les similarités entre les capacités fournies et les fonctionnalités requises. Le deuxième challenge pour cette infrastructure est de supporter l'interaction entre les services Web. En particulier, elle devrait permettre la spécification des informations q'un service Web exige et fournit, le protocole de l'interaction et les mécanismes de bas niveau exigé pour invoquer le service Web **[SYC et al03].**

La composition des services implique aussi la synthèse de la composition et l'orchestration .La synthèse de la composition donne la spécification sur la manière de coordonner les services disponibles afin de répondre à la requête du client. Cette spécification peut être automatisée par des algorithmes ou bien manuellement par un humain .Une fois la spécification du service composite est faite l'orchestration a pour rôle de coordonner entre les services composants et de contrôler les flux de données entre les composants afin de garantir la bonne exécution du service composite.

La composition de services peut être simplement définie comme étant "le procédé consistant à combiner des services existants pour former de nouveaux services". La composition de services web est fondée sur deux activités différentes : l'Orchestration et la Chorégraphie.

Une **Orchestration** comprend un élément central responsable de la composition dans son ensemble. Le processus devient alors la somme de ses sous processus et l'orchestrateur gère, seul, les échanges de messages. L'orchestration de services fait référence à l'activité de création de processus, exécutables ou non, qui utilisent des services web. Une **Chorégraphie** décrit les séquences de messages échangés par un service Web lors de son interaction avec d'autres services (typiquement l'échange public de messages entre des services web) plutôt

qu'un processus spécifique exécuté par un acteur en particulier. Il s'agit donc d'une manière décentralisée de gérer une composition puisque chaque service Web est responsable d'une partie du workflow.

Actuellement il existe différents standards permettant de réaliser l'orchestration et la chorégraphie, parmi lesquels **WSFL** d'IBM et **XLANG** de Microsoft. Ces efforts ont ensuite été fusionnés pour former une spécification commune nommée "Business Process Execution Language for Web Services" **(BPEL4WS) [ISM05].**

2.4. Issues de Composition

Dans ce qui suit nous allons introduire Six différentes issues qui ont un grand impact sur la composition des services web ainsi que quelques concepts, quelques efforts de standardisation et travail de recherche sur la composition des services web **[DUS05].**

2.4.1 Coordination

Les interactions entre les services web lors de la composition exigent la coordination de séquences d'opérations, pour assurer la convenance (correctness) et la cohérence (consistency).

De nouveaux protocoles sont apparus afin de fournir des modèles d'abstractions, et simplifier le développement des services web. Différent efforts de standardisation, tel que WS-Coordination **[CAB02]** par IBM ou WS-CF par Sun **[BUN et al03]** ont été proposées.

2.4.2. Transaction

Le modèle des transactions **ACID [DUS05]** des services web permet d'encapsuler une séquence d'opérations comme une unité atomique de traitement, appelée transaction, et qui vérifie les propriétés suivantes : l'*Atomicité*, la *Cohérence* (*consistance*), l'*Intégrité* et la *Durabilité.*

L'intérêt de l'approche transactionnelle est qu'elle assure (i) une exécution correcte (en terme de fiabilité) d'une transaction prise individuellement et (ii) des exécutions correctes (en terme de cohérence) de plusieurs transactions concurrentes. Les transactions sont utilisées pour accroître la fiabilité des interactions entre les services Web et pour simplifier le traitement des erreurs dans les applications de taille importante. Un protocole de transaction **[PAP03]** est ajouté au framework de coordination afin de fournir des transactions de courte durée appelée aussi des transactions atomiques, ainsi que des transactions de longue durée.

Dans le cas des Web Services les transactions sont distribuées et souvent de longue durée. Les Transactions distribuées implique plusieurs gestionnaires de ressources sur plusieurs machines. La validation (commit) des mises à jours doit être faite de manière

à garantir leurs abandons complets en cas d'incident sur l'un des gestionnaires de ressources. Les transactions atomiques sont de type "tout ou rien". Les actions qui sont entreprises avant le commit ne sont que des tentatives, c'est-à-dire non persistantes et non visibles aux autres activités. Quand une application se termine, elle interroge le coordinateur pour déterminer le résultat de la transaction. Le coordinateur détermine si des erreurs d'exécution ont eu lieu en demandant à chaque participant de voter. Si tous les participant votent qu'ils ont réalisés leurs tâches avec succès, le coordinateur valide (commit) toutes ces actions. Si l'un des participants vote qu'il a échoué ou s'il ne répond pas, le coordinateur annule toutes les actions entreprises (abort). Une LRT (Long-Running Transaction) est la capacité de gérer et de garantir qu'une transaction distribuée a été accomplie (ou annulé) dans un environnements faiblement couplés.

2.4.3. Contexte

D'après **[KKB et al04] [ALV et al03]** le contexte des services Web peut être inféré comme l'ensemble d'informations utilisés par le service Web pour ajuster l'exécution et les outputs afin de fournir au client un comportement adapté et personnalisé à ses besoins. Le contexte peut être enrichi en y ajoutant de nouveaux types d'informations à n'importe quel moment, sans aucun changement de l'infrastructure fondamentale. Le contexte peut contenir des informations telles que le nom du consommateur, adresse, et l'emplacement courant, le type de dispositif que le client utilise (matériel et logiciel) ou tous genres de préférences du client concernant la communication. Le standard WS-Context **[BUN et al03]** proposé par Sun, spécifie le contexte, le partage de contexte, et la gestion du contexte.

2.4.4. Modèle de conversation

Benatallah et autres **[BEN et al04]** modélisent le comportement des services Web en termes de ses conversations avec les clients, c à d, comme un arbre de messages échangés, sur lesquels des contraintes d'ordre sont imposées. Le comportement d'un service Web est représenté comme une machine à états finis déterministe où les transitions sont étiquetées par des opérations et les états avec le statut de la conversation, par exemple l'effet de l'opération qui mène à cet état est clairement défini. L'état initial est étiqueté par « start ». Quelques transitions ne sont pas étiquetées avec une opération puisqu'elles modélisent ces situations quand l'évolution des conversations à partir d'un état à un autre n'est pas causée par une opération (explicitement invoquée), mais par un événement (prédéfini) par exemple le time-out.

Les auteurs **[BEN et al04]** ont introduit un framework qui aide les développeurs à définir des modèles et des abstractions plus riches de services Web. Ces abstractions peuvent être classifiées comme des abstractions d'achèvement ou d'activation. Les abstractions d'achèvement contiennent des opérations compensatrices en cas ou une opération avancée soit annulée et des opérations de verrouillage qui verrouille les ressources d'un client. Les abstractions d'activation décrivent implicitement et explicitement les transitions entre états.

Un type important de transitions implicites est les transitions chronométrées qui se produisent automatiquement après un certain temps. Pour modéliser les transactions, de multiples propriétés sont utilisées:

Les propriétés d'activation décrivent le déclenchement des transitions ,des propriétés qui expliquent de manière précise l'effet d'une transition sur l'état du client, des propriétés de verrouillage de certaines ressources pour le client durant un temps donné.

Le modèle de conversation facilite la découverte et la composition dynamique des services, la validation du modèle de composition de services, la génération du squelette de composition de service, l'analyse de la génération du modèle de conversation, de la composition ainsi que les conversations. Le modèle de conversation inclus la spécification WSCL (Web Services Conversation Language), et WSCI (Web Service Choreography Interface), ainsi que WS-Coordination et WS-Transaction.

2.4.5. Supervision de l'exécution (Exécution monitoring)

Les services Web composites peuvent être exécutés soit d'une manière centralisée ou distribuée. L'exécution centralisée est semblable au paradigme client -serveur, le serveur est le programme central qui contrôle l'exécution des composants du service Web composite. A l'inverse, dans le paradigme distribué, les services Web participants partagent leur contexte d'exécution. Chacun des hôtes exécutant le service Web a son propre coordinateur qui doit collaborer avec les coordinateurs des autres hôtes pour garantir l'exécution des services web dans l'ordre correct.

2.4.6. Infrastructure

Dans l'infrastructure de base qui est décrite dans la partie I, plusieurs entrées UDDI sont inutilisables, des pointeurs (ou indicateurs) sont absents, ou contient des informations inexactes et imprécises, de plus la découverte de services dans UDDI est limitée seulement aux exigences fonctionnelles. Afin de spécifier les informations concernant la qualité de service (QoS) une nouvelle approche a été adoptée **[RAN03].** Cette approche ne touche pas au modèle de service Web existant mais elle l'étend en y ajoutant une autre entité appelée ' QoS certifier. Le rôle du certificateur permet de vérifier les demandes de QoS du fournisseur de services avant que l'enregistrement du service dans le registre UDDI puisse avoir lieu. Le certificateur peut approuver ou refuser la demande du fournisseur.

Il se peut qu'il y ait beaucoup de services qui correspondent aux exigences fonctionnelles de l'utilisateur, les contraintes de QoS peuvent aider à filtrer les services recherchés. Le client peut ajouter quelques contraintes sur la qualité de service, et donc, exiger une recherche beaucoup plus fine. Si aucun service ne correspond, l'utilisateur peut réduire les contraintes.

La structure de UDDI contient un élément supplémentaire appelé *'qualityInformation'*. Cette nouvelle structure fournit la description des différents aspects de QoS, tel que la disponibilité, la fiabilité ou la performance.

```
<SOAP-ENV:Envelope>
        <SOAP-ENV:Body>
                <find service>
                ...
                        <qualityInformation>
                        <availability>0.9</availability>
                        </qualityInformation>
                </find service>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Les auteurs **[RAN03]** définissent différents aspects de QoS qui est relative :

- *A l'exécution* comme scalabilité, capacité, performance (temps de réponse, latence, débit), fiabilité, disponibilité, flexibilité, gestion d'exception, et l'exactitude.
- *Aux supports de transaction* comme régulateur, standards supportés, stabilité, coût, et perfection.
- *A la sécurité* comme authentification, autorisation, confidentialité, capacité de trace et d'audit, cryptage de données et la non répudiation.

2.5. Classification de composition de services Web (type)

Etant donné un ensemble de services web disponibles et une requête d'un client, le problème de la composition de services Web est de synthétiser un nouveau service composé qui réalise la requête du client, donc spécifier comment coordonner les services Web disponibles pour réaliser cette requête. Une telle spécification peut être obtenue selon plusieurs types de classifications :

2.5.1. Composition statique

Ce type de composition peut être appliqué dans des environnements « stables » où les services Web participants sont toujours disponibles et où le comportement du service composite est le même pour tous les clients. La composition des services web prend place durant la période de conception. Les composants sont choisis, relient entre eux et enfin compilés et déployés. Le service composite ainsi obtenu fonctionnera bien tant que son environnement et les services qui le compose ne changent pas ou ne changent que rarement. Les stratégies de récupération en cas d'échec des services, tel que substitution du service ou la gestion des erreurs, sont prédéfinis. Microsoft BizTalk et Bea WebLogic sont des exemples de moteurs de

composition statique [SUN et al03]. Cependant, l'environnement de services Web est un environnement fortement flexible et dynamique, de nouveaux services deviennent disponibles et leurs fonctionnalités peuvent changer fréquemment, ainsi que le nombre de fournisseurs de service progresse constamment.

Des problèmes peuvent survenir si un fournisseur actualise un service ou si l'un d'entre eux est substitués par un autre. Dans cette situation il est inévitable de changer l'architecture logicielle, et dans le pire des cas changer la définition du processus et reconcevoir le système.

Dans ce cas, la composition statique peut être trop restrictive et les composants doivent s'adapter automatiquement aux changements inattendus ainsi qu'aux exigences du client avec l'intervention minimale de l'utilisateur [SUN et al03]. La composition sous des conditions statiques considère que les fonctions du service ne changent pas et que les services sont toujours disponibles. Pour cela de nouveaux types de compositions ont été considérés.

2.5.2 Composition dynamique

Dans ce type de composition, les services Web à composer sont déterminés lors de l'exécution de la requête d'un client. Ils peuvent être déterminés selon les contraintes de chaque client, la disponibilité des services Web, …etc. La composition dynamique apparaît la plus intéressante d'une part, elle promet d'être capable de faire face à un environnement très dynamique dans lequel des services apparaissent et disparaissent rapidement. D'autre part, elle permet de mieux satisfaire les besoins de chaque client en minimisant son intervention. La composition des services web implique : i) La découverte des « bons » services à composer selon des contraintes et des besoins. ii) La dynamicité et la flexibilité dans la composition de services Web.

Il existe deux approches pour la découverte des services web. La première approche est basée sur le langage de description syntaxique des services Web WSDL (Web Service Description Language) qui permet de décrire les fonctionnalités, les données (sous forme de messages) et également le protocole d'accès aux services Web. Le standard UDDI (Universel Description, Discovery and Integration) est un registre de service Web permettant d'identifier et de localiser un service Web en se basant sur le langage WSDL. La deuxième approche est basée sur les langages : DAML-S (DARPA Agent Markup Language) et OWL (Ontology Web Language) qui permettent de décrire la sémantique des services Web en se basant sur des ontologies.

StarWSCoP (Star Web Services Composition Plateform) [SUN et al03] est l'une des plateformes de composition dynamique de services Web. Il inclut plusieurs modules: un système intelligent qui décompose les exigences de l'utilisateur en des descriptions abstraites du service, un registre de service qui est un entrepôt du service web; un moteur de découverte pour trouver des services adéquats qui satisferont les exigences de l'utilisateur dans le

registre de service, un moteur de composition qui supervise les services composés afin qu'il soit exécuté dans le bon ordre, un wrapper (adaptateur) qui assure l' interopérabilité des services hétérogènes qui ont été développés séparément par des fournisseurs différents, une bibliothèque qui stocke les traces d'informations sur l'exécution des service composite, une estimation en temps réel des QoS du service composite et un moniteur de l'événement qui contrôle les événements et notifie le moteur de composition.

Cette plateforme ajoute à l'UDDI une couche basée sur les ontologies pour définir une sémantique pour les services Web. Pour assurer la composition dynamique des services web basé sur la QoS, WSDL est étendu avec des attributs de QoS, comme le temps, le coût, ou la fiabilité.

```
<portType name="GetTotalFilmCost">
        <operation name="getTotalFilmCost"
                cost="20"
                time="50"
                reliability="90%">
         ...
        </operation>
</portType>
```

2.5.3. Composition vision industrielle

L'industrie s'intéresse au développement d'outils communs pour la composition manuelle des services Web où tous les composants du service sont représentés par des fichiers WSDL, et suppose un mode d'interaction atomique. Comme on a cité ci-dessus (section 3), la composition de services Web est fondée sur deux activités différentes: L'Orchestration et la Chorégraphie. En effet, l'industrie se concentre plus sur l'orchestration (obtenue manuellement) et la spécification des services Web composés, exprimé dans des langages. Différents standards permettant de réaliser l'orchestration et la chorégraphie, par exemple BPEL4WS, WSCI et BPEL.

L'orchestration de services Web est basée sur une technique de Workflow. Elle exige une spécification complète du service composite, en terme de la façon dont les divers services sont liés, et du flux du processus interne du service composite.

BPEL4WS (Business Process Execution Language for Web Services) est un langage de composition proposé par IBM, Microsoft et BEA qui correspond à une grammaire XML qui décrit des processus métiers qui peuvent être interprétés et exécutés par un moteur d'orchestration (BPWS4J4 par exemple.). Un processus métier correspond à une séquence d'opérations ou plus exactement à un flux d'activités **[WVD02]**. Ces activités peuvent faire intervenir de un à plusieurs services Web. On peut distinguer les activités de base des activités

structurées. Les activités de base de ce langage permettent : d'invoquer une opération d'un service Web tiers (activité ***invoke***) ; de présenter la composition comme un nouveau service Web avec l'activité ***receive*** pour décrire la réception d'une requête ; et l'activité ***reply*** pour générer une réponse. Les activités structurées structurée gèrent tout le flot du processus et spécifient à quel moment ont lieu les activités de base. Elles peuvent définir des variables qui peuvent contenir des valeurs retournées par un service Web et qui peuvent ensuite être employées comme des paramètres dans l'appel d'un autre service Web. Elles utilisent les activités de base pour décrire : des séquences ordonnées (***sequence***) et des exécutions en parallèle (***flow***) ; des branchements (***switch, if***) et des boucles (***while***). des chemins alternatifs (***pick***).

Le langage intègre également un mécanisme de gestion des exceptions (***throw, catch***), ainsi qu'un mécanisme de compensation (scope) qui permet d'annuler une transaction dans son intégralité lorsque celle-ci échoue. Il constitue une couche supérieure au langage de description WSDL. Il utilise, en effet, WSDL pour définir les opérations de services Web élémentaires à appeler et pour présenter le processus métier comme un nouveau service Web. BPEL4WS considère de plus la possibilité de créer des compositions de services Web hiérarchiques, c'est à dire d'utiliser le processus comme l'implémentation d'une interface de service.

2.5.4. Composition vision académique

Plusieurs approches pour la composition de services Web vision académique ont été proposées. Dans cette partie nous décrivons ce type de composition selon plusieurs auteurs.

McILraith et son groupe [MCI02] [MSZ01] présente un outil pour la composition des services Web représentés comme actions de Situation Calculus et représenté par le langage OWL-S (Web Ontology Language for Services) qui est un langage de description de services basé sur XML utilisant le modèle des logiques de descriptions (une proposition d'ontologie des services Web). Il est aussi un langage de haut niveau pour la description et l'invocation des services Web dans lequel la sémantique est incluse et interfacé avec UDDI/WSDL/SOAP.

Cet outil fonctionne comme suit : un client présente sa requête au système exprimé sous forme de procédure générique en associant ses contraintes et ses préférences. Cette requête ne peut pas être exécutée directement mais exécutée par un agent en exploitant les ontologies OWL-S des services Web, l'agent instancie automatiquement les spécifications avec les contraintes des services Web dans cette ontologie, (c.-à-d, un genre de flux de processus dans la théorie de Situation Calculus), qui présente une spécification partielle du comportement du service composite désiré. Chaque noeud de la situation d'arbre présente une situation, c-à-d une configuration instantanée du service à chaque point de ces exécutions. Quand un composant d'un service Web est exécuté, due à la connaissance incomplète sur ses effets on ne connaît pas la nouvelle situation résultante puisque plusieurs situations peuvent aussi être présentées.

Pour la modélisation du comportement global de composition des services Web une structure a été définie par **Hull et son groupe [BUL et al03]**. Les services échangent des messages selon la topologie de la communication prédéfini, exprimée comme un ensemble de canaux parmi les services Web, les séquences de messages échangés désignés sous le nom de conversation. Dans cette structure les propriétés de conversation sont étudiées afin de caractériser le comportement des services Web, modélisé comme des machines de **Mealy**. Les auteurs abordent le problème de la composition de services Web comme suit : les données entrées au problème de composition sont : i) Un comportement global désiré ou le service Web cible (c-à-d, un ensemble de toute conversation désirée qui est le résultat de cette composition) spécifié comme un langage régulier, et ii) Une infrastructure de la composition qui est un ensemble de canaux, un ensemble (nom) de services Web et un ensemble de messages.

Le résultat de la composition est une spécification de machine de Mealy, et ce en utilisant DPDL (Deterministic Proposisional Dynamic Logic) **[HKT00]**, qui est une logique dynamique permettant d'exprimer des changements dans l'exactitude des formules logiques durant l'exécution d'un programme, elle est basée sur un ensemble de propositions (vraies ou fausses).

2.6. Approches de composition de services web

Les approches de composition de services web ont commencé à émerger pour remédier à l'incompatibilité des langages de composition de première génération Web Service Flow Language (WSFL) d'IBM et Web Services Choreography Interface (WSCI) du Système BEA. Les chercheurs ont développé des langages de seconde génération, tels que le Business Process Execution Language for Web Services (BPEL4WS), qui combine WSFL et WSCI avec les spécifications du XLANG de Microsoft.

Nous présentons dans cette section un aperçu des approches de composition de services Web proposées par différents auteurs, que nous comparons en respectant quatre conditions principales à savoir *la Composition Automatique, la Connectivité et Propriétés non fonctionnelles de qualité de service, la Scalabilité et l'exactitude (Correctness) de la composition.*

2.6.1. Composant Web

Afin de supporter le besoin de la réutilisation, la spécialisation et l'extension de services, ainsi que la flexibilité et la scalabilité de la composition de services, le concept du composant Web a été introduit **[YAN02]**.

Les composants Web sont basés sur un mécanisme de paquetage pour développer des applications réparties basées sur le Web en combinant des services Web élémentaires ou complexes existants (publiés). Ils présentent leurs interfaces et opérations d'une manière

uniforme et consistante sous la forme de définitions de classe. Ils ont une nature récursive du fait ils peuvent composer des services Web publiés qu'ils sont également à leurs tours considérés comme des services Web quelques soit leurs nature de complexité.

Les services Web sont représentés comme des composants pour supporter les principes de base du développement logiciel cités ci-dessus (la réutilisation, la spécialisation, et l'extension). L'idée principale est d'encapsuler les informations de la logique de composition à l'intérieur de la classe de définition qui représente un composant Web. L'interface publique du composant Web peut être alors publiée et peut être utilisée pour la découverte et la réutilisation. La logique de composition comprend le type de composition et la dépendance des messages.

Le type de composition signifie la nature de composition, il peut prendre deux formes:

- ***Ordonné*** (Order) : détermine si un composant peut exécuter des services composants séquentiellement ou en parallèle.
- ***Exécution Alternative*** : indique si un composant peut invoquer des services alternatifs jusqu'à ce que l'un d'entre eux réussis.

La dépendance des messages définit la correspondance (mapping) de messages d'inputs et d'outputs. Il y a trois types de dépendance:

- ***la Synthèse*** : Cette construction combine les messages d'output des services composants pour former les messages d'output du service composé.
- ***la Décomposition*** : relie les messages d'inputs des services composants pour générer les messages d'inputs du service composé. Cette construction décompose le message d'input du service composé pour générer les messages d'output des services composants.
- ***La correspondance de messages*** (message mapping) : permet de faire la correspondance entre les Inputs et les outputs des services composants. Par exemple, l'output d'un service composant est l'input d'un autre service.

Le standard actuel de définition de services Web (WSDL) comprend les constructions suivantes : messages, portTypes, liens, ports. Un service se compose d'un ensemble de ports qui définissent les liens utilisés dans le service. Pour demander ou découvrir un composant Web, il faut définir ses propriétés (messages) et opérations en WSDL. Une fois la classe du composant Web est définie, elle peut être réutilisée, spécialisée, et extensiée. Le même principe s'applique à l'activité de composition de service où un service composé est vu comme un service Web spécial qui contient des constructions et la logique de composition. Cette dernière représente un aspect fondamental de la composition car elle permet de définir comment les services composants peuvent être combinés, synchronisés, et coordonnés qui est modélisée comme une séquence d'interactions entre deux services.

Les composants Web offrent la compatibilité et la vérification de conformité. Deux services, S1 et S2, sont compatibles quand S1 est au moins aussi capable que S2, et quand S1 peut remplacer S2. Le service S1 se conforme au service S2 quand nous pouvons combiner S1 et S2 de sorte que l'output de S1 puisse être pris comme input de S2. L'approche par composant supporte plusieurs constructions de la composition de base: séquentielle, alternative séquentielle, parallèle avec synchronisation du résultat, alternative parallèle. Les possibilités de construction sont augmentées par l'utilisation de condition while-do **[MIL04]**.

Le concept de composant est utilisé comme moyen pour créer des services web composites ainsi que pour la réutilisation, la spécialisation et l'extension de services Web. Il propose un langage de composition de services exprimé en XML qui peut être utilisé comme un script pour contrôler l'exécution des séquences de compositions de services Web existants, et peut être facilement échangé à travers le réseau. Il fournit aussi un framework complet pour la composition de services Web qui peuvent être organisés, définis et invoqués sur la base des composants Web. Cela permet une grande quantité de réutilisation et flexibilité de compositions des services.

2.6.2. Composition algébrique et mathématique des services web

La composition algébrique des services a pour but d'introduire des descriptions beaucoup plus simple que les autres approches, et permet de modéliser des services comme des processus mobiles pour assurer une vérification de certaines propriétés tel que la sécurité, la vivacité 'liveness' (la terminaison correcte), et la gestion des ressources. La théorie des processus mobiles est basée sur π- calcul, dans laquelle l'entité de base est un processus qui peut être un processus vide, un choix entre plusieurs opérations I/O et leurs continuations, une composition parallèle, une définition récursive, ou une invocation récursive.

Les opérations I/O peuvent être des inputs (receive) ou bien des outputs (send). Par exemple, x(y) dénote la réception du tuple y sur le canal x; $\bar{x}$[y] dénote l'envoie du tuple y sur le canal x. une notation pointillée qui spécifie une séquence d'action par exemple $\bar{c}$ [1,d].d(x,y,z). $\bar{c}$ [x+y+z], dans laquelle un processus envoi le tuple [1,d] sur le canal c,reçoit dans le canal dont les composants son reliés aux variable x,y,z puis envois la somme x+y+z de ces dernières sur le canal c. La composition de processus parallèle est dénotée A|B. Plusieurs processus peuvent s'exécuter en parallèle et communiquent en utilisant des canaux compatibles.

Décrire un service d'une manière abstraite implique la vérification de la convenance de la composition. En utilisant le π- calcul en peut décrire l'exemple précédent par :

A(processInput).B'[AOutput].C'[AOutput]|B(BInput).out'[BOutput]|
C(CInput).out'[COutput] |out (processOutput).

On peut aussi vérifier d'autre propriétés pertinentes en assignant des types de comportements au processus, il existe deux manières de typer un processus : on peut typer seulement un sous ensemble de port ou bien la totalité du processus. Dans le premier cas nous pouvons proscrire le type ou le format de données qui peuvent être échangées via deux ports. Pour le second cas on type le processus en entier. La notion de type devient l'image homomorphique du processus. *Avec la composition en algèbre de processus, la question qui se pose est quelles sont les informations à typer* [MIL04].

Dans le contexte du calcul mathématique la composition de services web paraît être essentiel. Le service web mathématique offre des fonctions spécifiques pour résoudre des problèmes atomiques. L'utilisateur doit être capable de composer ces fonctions dans son algorithme pour résoudre son problème. Cela peut être accompli en utilisant un langage à plan"*plan langage*" pour produire un document plan. Un plan est un document qui décrit comment utiliser les différents services web mathématique pour résoudre un problème particulier. Un plan est comme un programme dans lequel la plupart des appels aux fonctions doivent être manipulés par le service web. Un plan est un document chorégraphique à état multiple qui peut être soit abstrait, non résolu ou résolu, selon le nombre de service connu et impliqué dans la chorégraphie. Un tel plan peut être instancier dans un langage de composition tel que BPEL ou dans une routine mathématique (Maple) pour l'exécution [CHI05].

2.6.3. Approche déclarative

Les approches courantes de la découverte, composition et exécution de services exhibent les limitations des technologies UDDI et WSDL. UDDI manque d'informations sur ce que fait le service, il supporte la recherche de services basés sur les métadonnées de services et non pas sur les descriptions actuelles qui caractérisent les services. Les registres de services ont besoin d'inclure les descriptions déclaratives des conditions d'utilisation de services et les effets secondaires d'usage du service ont besoin d'être fourni pour sélectionner un service.

Une fois un service est localisé, les descriptions WSDL ne décrivent pas en détail les effets des actions de services (opérateurs ou méthodes), et elle ne fournissent pas d'une manière explicite des informations sur les pré-conditions pour exécuter une action du service (méthode). De plus les informations contextuelles nécessaires pour faire face aux issues telles que l'annulation de service, l'autorisation et les paiements de service ne sont pas explicitement décrits.

L'exécution des services Web composite requiert la maintenance d'un état du service pour faciliter l'échange de données et le contrôle entre les services. Pour cela, une représentation explicite est exigée pour assurer une bonne interaction entre les services élémentaires.

Ambroszkiewicz [ABS03] a défini une approche qui est quelque peu différente des plateformes de composition typique, où le concept de description du service et du registre de service définit dans l'architecture de cette approche ont été révisé. Le but de cette approche est de créer un langage de description qui va pâlir aux inconvénients des standards actuels, exprimer les requêtes du client et permettre un usage distribué pour permettre aux utilisateurs d'introduire de nouveaux types de ressources, des fonctions et des relations avec des noms uniques, à savoir URLs. Les requêtes des clients sont exprimées d'une manière déclarative en utilisant des langages formels.

L'approche déclarative est constitué de deux phases: la première phase prend une situation initiale et le but désiré comme point de départ, et construit un plan générique pour atteindre le but souhaité. Elle est réalisée en utilisant PDDL (Planning Domain Definition Language) [GER06] l'un des langages de description les plus connu dans le domaine de planning qui a influencé le développement de nombreux langages de description de services Web comme OWL-S (Web Ontology Language for Services) [MBH et al04] et estime la régression du plan en utilisant XSRL (XML Web-Services Request Language) qui doit fournir une sémantique exploitable par la machine et spécifier le comportement du service abstrait. La seconde phase choisit un plan générique pour découvrir les services appropriés, et construire à partir de ses derniers un workflow, qui peut être réalisé en utilisant des langages de modélisation de processus existant, tel que BPEL.

McDermott [MCD02] a étendu PDDL en introduisant la notion de "valeur de l'action", représentant certaines informations qui sont créés ou appris comme une conséquence d'exécuter une action particulière. L'intention principale d'introduire cette extension est d'avoir la capacité de capturer les informations et le contenu de messages qui sont échangés entre les services.

Dans La plate forme SELF-SERV [SHE et al02], les services web sont déclarées puis composés, enfin exécuté dans un environnement dynamique (pair -à -pair). SELF - SERV définit trois types de services: les services élémentaires, composés, et les communautés de service. Les communautés de service peuvent être vu comme un conteneur de services alternatifs. L'exécution du service est supervisée par un composant logiciel appelé coordinateur qui initie, contrôle et supervise l'état des services composés. Les coordinateurs recherchent les informations pertinentes du plan de l'état du service et le représente dans une table de routage qui contient les pre-conditions et les post-traitements (post-processings).

Le framework SELF - SERV est constitué d'un service manager et de plusieurs pools de services, tous sont implémentés en Java et communiquent par échanges de documents XML. Les informations Inputs et outputs sont tout deux structurés comme des documents XML. Le service manager est un moteur de recherche et de découverte de services qui facilite la publication et la localisation de services, un éditeur de services qui facilite la définition de nouveaux services et l'édition des services existants, et un service déployer qui génère des

tables de routage pour chaque état d'un service, ces tables sont transférés aux hôtes du service composite correspondant.

2.6.4 Composition de services dirigée par les modèles

Orriens et al **[ORR et al03]** ont introduit une approche de composition de services dirigée par les modèles qui est basée sur la composition dynamique des services web afin de pâlir aux inconvénients des standards courants de la composition des services Web, tel que BPEL. Cette approche doit faciliter la gestion et le développement de la composition dynamique de services. Les processus métiers peuvent être construit dynamiquement en composant des services Web dans un mode dirigé par les modèles où le processus de conception est contrôlé et régi (gouverner) par une série de règles métier.

UML (Unified Modelling Language) est utilisé pour fournir un niveau élevé d'abstraction, et permettre de créer un lien direct aux autres standards, tel que BPEL4WS. L'OCL (Object Constraint Language) est utilisé pour exprimer les règles métiers et décrire le flux des processus. Les règles métiers peuvent être utilisées pour structurer et programmer (schedule) la composition des services, et décrire la sélection des services, ainsi que leur enchaînement. Le processus de développement de composition de services peut être subdivisé en quatre phases : définition, planification (scheduling), construction, et exécution de service. Pour permettre la représentation de toutes les compositions possibles des services, les auteurs ont introduit une abstraction du méta modèle qui modélise les composants et leurs relations et définis les éléments et les règles de composition des services.

La composition des services web est souvent associée aux workflow **[GEO et al02] [PRE02] [HAN et al02] [PIC et al03]** , les éléments de la composition du service sont basés sur l'activité des éléments du processus métier , en représentant précisément les fonctions métier (condition), conditionnant le comportement de la composition en y ajoutant des pré et post-conditions (événement), décrire des occurrences des événements pendant le processus de composition du service (flow), définir un bloc d'activités (message), sauvegarder les inputs et les outputs (fournisseur), décrire la partie qui offre des services concrets et la classe abstraite (rôle), décrire toutes les parties qui participent à la composition du service (processus).

Les règles de composition peuvent être modélisées en utilisent OCL et peuvent être structurées comme suit:

- Des règles de structurations qui guident le processus de structuration ;

- Des règles de données qui contrôlent les relations entre les messages et l'usage de données ;
- Des règles de comportement qui contrôlent les occurrences des événements et assurent le respect des contraintes d'intégrité ;
- Des règles de ressources qui contrôlent l'usage des ressources, tel que la sélection de service, des fournisseurs… etc.
- Finalement les règles d'exceptions, pour gérer les comportements exceptionnels dans le processus de composition de service.

La composition de services dirigée par les règles métier [ORR et al03 a] [ORR et al03 b] est similaire à celle dirigée par les modèles. Les auteurs utilisent les mêmes éléments de la composition du service et définissent quatre composants pour l'architecture de composition: le définisseur, le programmeur, le constructeur, et l'exécuteur pour représenter les quatre phases du processus de composition de service. Le framework qui a été défini compose d'un coordinateur de composition de services SCM, qui assiste l'utilisateur dans le développement, l'exécution, et la gestion de la composition des services, et un entrepôt de composition de service qui stocke les éléments et les règles de composition.

Le SCM détermine les exigences de l'utilisateur et les passent au définisseur afin d'établir une composition abstraite de service. Pour déterminer si toutes les informations sont déjà disponibles, le définisseur demande au moteur de composition d'interroger l'entrepôt de règles de composition. Dans le cas où les règles de composition correspondante existent déjà, l'entrepôt des éléments de composition renvoi les activités trouvées au définisseur. Les nouvelles règles sont stockées dans l'entrepôt de règles de composition et dans l'entrepôt des éléments de composition. Le programmeur développe un ensemble de compositions concrètes et laisse l'utilisateur sélectionne une alternative qui est passée au constructeur pour générer le logiciel exécutable. L'exécuteur supervise l'exécution du service.

Un plan de classification des règles métier a été introduit: des règles de structuration qui facilitent la spécification de la manière dans laquelle la composition de services peut être effectuée, des règles reliées au rôle qui gouvernent les participants impliqués dans le processus de la composition du service, des règles concernant les messages qui régulent l'usage des informations, des règles concernant les événements qui contrôlent le comportement du service web composite dans le cas où un événement attendu ou inattendu surviens et enfin des règles concernant les contraintes qui représentent les conditions de la composition.

2.6.5 Approche formelle (Une Spécification formelle pour la composition et la vérification des services Web)

Plusieurs propositions, tel que BPEL4WS et WSCI ont été utilisées pour décrire le processus opérationnel de la composition ou pour décrire le flux de messages dans la collaboration, mais ils manquent de quelques mécanismes de raisonnement pour vérifier

l'interaction entre services web. Il est possible qu'un processus prédéfini se comporte anormalement dû aux interactions inexactes entre les services web. D'où apparaît le besoin de fournir une méthode formelle pour vérifier la solidité du service web composite. Les méthodes formelles est un domaine d'étude qui fournit un langage pour décrire la spécification, la conception et le code source d'un logiciel. Des preuves formelles sont possibles pour prouver des propriétés exprimées.

Pistore et al. **[PIS et al05 a] [PIS et al05 b]** représentent les services Web utilisant des systèmes de transition **[MIL82]** qui communiquent en échangeant des messages. Cette approche se fonde sur les techniques de contrôle du modèle symbolique pour déterminer une composition parallèle de tous les services disponibles, et alors générer un contrôleur qui contrôle les services composés afin qu'ils satisfassent les besoins spécifiés par l'utilisateur.

Informellement, si $W = \{ W1 , W2 ,... Wn \}$ est l'ensemble de services disponibles, p est le besoin spécifié par l'utilisateur (c.-à-d., p décrit le but G), et le || est l'opérateur de composition, le but est de déterminer un "contrôleur" Wc, tel que : $Wc \triangleright (W_1 || ... || W_n) \models p$.

De même, le framework Colombo **[BER et al05]** modélise les services Web en utilisant les systèmes de transition marqués. Cependant, cette approche se fonde (compte) sur la spécification des liaisons pour établir les canaux de communication entre les services qui ont des signatures identiques et les exploite pour déterminer une composition faisable qui répond aux exigences du but en réduisant les problèmes de composition pour la satisfaisabilité d'une formule propositionnelle déterministe approprié de la logique dynamique.

Gwen Salaün et autre **[SBS04]** ont modélisés les services Web d'une façon légèrement différente appliquant l'Algèbre de Processus **[HEN88]** (PA) en deux manière différentes: (i) en temps de conception, l'algèbre de processus peut être utilisée pour décrire une spécification abstraite du système à développer qui peut être validé et utilisé comme une référence pour la mise en oeuvre; (ii) en appliquant l'ingénierie inverse, les descriptions d'interface du service Web existantes peuvent être traduites à des algèbres de processus. Ce qui a adopté au CCS **[MIL82]** comme une algèbre de processus et a démontré des techniques pour traduire les processus BPEL **[AND03]** en CCS, qui peut être alors vérifié pour raisonner au sujet des propriétés spécifiées dans la logique temporelle. Hamadi et Benatallah **[HAM03]** ont appliqués une algèbre basée sur les réseaux de pétri pour modéliser le flux de contrôle et capturer les sémantiques des compositions des services Web complexes. Le framework qu'ils ont défini fournit plusieurs constructions de flux de contrôle comme la séquence, l'alternative, l'itérative et l'arbitraire. Des constructions qui peuvent être utilisées pour déterminer et vérifier une composition sont aussi montrées. Cependant, au lieu d'une composition faite (semi-) automatiquement ou manuellement, SELF-SERVE **[BSD03]** a étendu ce travail en fournissant les capacités pour composer et exécuter dynamiquement les services Web représentés comme diagrammes d'état, et pour adopter un environnement de calcul pair à pair (P2P) pour exécuter

les services composites, ce qui a dans la pratique des avantages multiples (en termes de scalabilité, défaut-tolérance etc…) comparé aux architectures centralisées.

2.7. Méthode de comparaison

Comparant maintenant les différentes approches selon les quatre exigences de la composition de service. Une discussion des possibilités de composition automatique de services est montrée dans la *Table 1*.

2.7.1. Connectivité et Propriétés Non fonctionnelles

Toutes les approches présentées offrent la connectivité des services. Bien que les services eux-mêmes soient modélisés de plusieurs façons, la connection consiste à faire la correspondance et l'orchestration des messages d'input et d'output entre les ports services des services partenaires. La plupart des approches négligent la spécification des propriétés de qualité de service (QoS) non fonctionnelles tel que la sécurité, la dépendance, ou encore la performance.

2.7.2. La Correction (Correctness) de la composition

La vérification de la convenance dépend du service et des spécifications de la composition. Cependant le degré de vérification de la convenance varie. L'approche par composant fournit une façon simple de vérifier la conformité et la compatibilité. π-Calcul offre une vérification algébrique puissante pour déterminer la vivacité, la sécurité, et la qualité de service. Cependant, Pétri utilise une algèbre compliquée pour la vérification.

Les méthodes de vérification des modèles de vérification sont comparables avec π-Calcul. Plusieurs méthodes sont disponibles pour prouver que la spécification de service composé est conforme au modèle. D'où l'importance de décider quels besoins doivent être spécifiés pour la vérification du modèle afin de produire des résultats utiles. Il est aussi important de calculer et de vérifier les ressources disponibles (tel que le temps CPU ou l'espace de stockage).

2.7.3. Composition Automatique

Plusieurs approches de composition ont pour but d'automatiser la composition ce qui promet un développement plus rapide d'applications, une réutilisation plus sûre et facilitera

l'interaction de l'utilisateur avec un ensemble de services complexes. L'utilisateur final ou l'application cliente spécifie un but (exprimé dans un Langage de description ou une notation mathématique) et un moteur "intelligent" de composition qui sélectionne les services adéquats et fournis une transparence de composition à l'utilisateur. Les principaux problèmes sont comment identifier les services candidat, les composés et vérifier si le service composite résultant répond à la demande.

2.7.4. La Scalabilité de la Composition

Toutes les approches de composition supportent la connectivité des services Web à travers des messages qui passe par les ports. Les utilisateurs finaux voudront interagir avec beaucoup de services. Par conséquent, l'une des issues critiques est comment proposer des approches évoluent selon le nombre de services impliqué.

L'approche par composant requière un temps supplémentaire pour la correspondance (mapping) et la synchronisation entre les définitions de classe et XML. Le π-Calcul offre une notation concise avec des mécanismes de réduction puissants qui facilitent la spécification des services complexes.

Le tableau suivant résume la comparaison de quelques propriétés intéressantes des différentes approches présentées.

Table 1. Comparaison des exigences de la composition de service

	Connectivité des services	Propriétés Non fonctionnelles	vérification de la composition	Composition Automatique	Scalabilité de composition
Composant Web	√		√		mauvaise
Algébrique	√		√		bonne
Dirigée par les modèle	√	√		√	bonne
Déclarative	√		√	√	moyenne
Formelle	√		√	√	moyenne

2.8. Conclusion

La composition des services web est un point crucial qui a un grand impact sur plusieurs domaines de recherches. Beaucoup d'efforts ont été fourni afin de permettre une composition utilisable et acceptable de services Web. Ces efforts ont été concrétisés par plusieurs standards et approches de compositions qui varient de celles qui aspirent à devenir des normes de l'industrie à celles qui sont beaucoup plus abstraites .Une approche idéale couvrirait chacune des quatre conditions principales qui ont été cité.

Le problème principal avec les approches "industrielles" est la vérification d'exactitude (correctness). De l'autre coté il est souvent difficile d'appliquer les approches formelles dans les environnements réels d'entreprise, et quelques problèmes de Scalabilité demeurent présents. Du point de vue d'exactitude, il est avantageux d'analyser les propriétés des services web en utilisant des mathématiques. Cependant, il faudrait pouvoir passer de WSDL et de SOAP aux solutions mathématiques.

Le but à long terme doit être d'incorporer des mécanismes de vérification qui permettent une bonne Scalabilité et permet aux utilisateurs et aux développeurs d'accomplir des tâches sans qu'ils ne se préoccupent pas de l'espace mémoire, ou si le processus aboutira à une impasse, ou encore de la sécurité et de la confidentialité…etc.

Après une étude des services Web, de la composition de services et des différentes approches, nous allons maintenant consacrer la partie suivante à un état de l'art de la programmation fonctionnelle, que nous utilisons dans notre approche pour la modélisation des services Web ainsi que leurs composition.

Chapitre 3

La programmation fonctionnelle

3.1. Introduction

Les premiers langages de programmation étaient généralement constitués d'une suite d'instructions s'exécutant de façon linéaire. Dans ce contexte, le lancement d'un programme débutait par l'exécution de la première instruction du fichier source et se poursuivait ligne après ligne jusqu'à la dernière instruction du programme. Cette approche linéaire, bien que simple à mettre en oeuvre, a très rapidement montré ses limites. En effet, les programmes monolithiques de ce type ne se prêtent guère à l'écriture de grosses applications et ne favorisent absolument pas la réutilisation du code. En conséquence sont apparus d'autres langages qui proposaient une approche radicalement différente : l'approche fonctionnelle.

Il est, certes, évident que l'objectif initial de pouvoir imiter les mathématiques n'est pas tout à fait atteint, mais néanmoins des avancées tant techniques que théoriques ont été réalisées durant un demi siècle de recherche. L'introduction même d'une nouvelle façon de voir les choses (fonctionnelle) et surtout de les représenter (fonctions) sur machine peut en témoigner.

Le premier langage fonctionnel à voir le jour est Lisp (List Processing) dans le cadre des travaux menés dans les années 50 en intelligence artificielle. Depuis, les langages fonctionnels ont connu tant d'évolutions et de nouveautés qui font qu'aujourd'hui une pléiade de langages revendiquant le caractère fonctionnel (ML, Miranda, Haskell, Scheme, CAML,…) exprime toute la vigueur et la richesse du monde des langages fonctionnels.

La programmation fonctionnelle prend progressivement une place de plus en plus importante dans le développement des logiciels depuis plusieurs années. A l'opposé des langages impératifs qui sont basées sur des considérations d'implémentation (machine Von Neuman), les langages fonctionnels s'appuient sur des concepts mathématiques simples qui font d'eux un outil de choix dans la maîtrise et le développement du logiciel **[BAC78].**

Les milieux de recherche en informatique furent donc les premiers à utiliser ces langages dans leurs travaux, dans le domaine du Génie Logiciel en particulier, et exploiter leurs qualités de base : modularité, réutilisabilité, facilité de transformations et de preuves de programmes.

L'objectif de cette partie est de présenter un survol des langages fonctionnels en termes de définitions, historique, et propriétés ainsi que les notions théoriques sur lesquelles se base le paradigme fonctionnel.

3.2. La programmation fonctionnelle

La programmation fonctionnelle représente un domaine d'étude qui suscite un intérêt croissant de la part des milieux de la recherche et plus récemment du monde industriel, grâce à l'apparition des langages fonctionnels évolués (tels que ML). Un programme fonctionnel est une application d'un domaine d'entrée vers un domaine de sortie. Exprimé autrement, un programme fonctionnel est un ensemble de fonctions elles-mêmes formulées à partir d'autres fonctions par compositions.

La programmation fonctionnelle s'appuie donc sur des concepts mathématiques simples et puissants : fonctions, application, composition et abstraction.

Un programme fonctionnel est une séquence d'expressions dont l'exécution dépend des valeurs prises par certains paramètres et qui a pour effet de calculer une valeur. L'exécution d'un programme consiste en l'évaluation de toutes les expressions.

La programmation purement fonctionnelle, dite aussi applicative, se distingue des autres modes de programmation (impérative, à objets) par son absence d'effet de bord au sens ou nous l'entendons. Une fonction a toujours le même comportement : elle n'a pas de mémoire d'un appel à l'autre et ne modifie jamais ses arguments ni une quelconque variable globale. Le seul effet d'un appel de fonction est la production de son résultat et aucune modification d'une zone mémoire "interne" ou "externe" (effet secondaire) n'est possible.

Une donnée structurée (arbre, liste) peut donc être construite par une fonction et rendue en tant que résultat (à partir de ses paramètres d'entrée), mais elle ne peut jamais être modifiée par une autre fonction.

Par conséquent et c'est un point important, le programmeur ne s'occupe jamais du "stockage" des valeurs manipulées par le programme, de l'allocation des cellules mémoires nécessaires, ni de leur libération : ces fonctions sont réalisées par le système.

Les avantages de la programmation fonctionnelle généralement évoqués peuvent être synthétisés par l'expression de trois qualités :

La première concerne la modularité des programmes. La programmation fonctionnelle facilite la conception des programmes dans la mesure où chaque fonction implémente naturellement une brique de base qui peut être écrite et testée individuellement. Mais le plus important, qui différencie nettement la programmation fonctionnelle des autres styles de programmation, ce sont les facilités d'abstraction et d'assemblage offertes par les fonctions d'ordre supérieur [HUG89]. Une fonction d'ordre supérieur est une fonction prenant une autre fonction en paramètre ou fournissant une fonction en résultat.

La deuxième qualité concerne la manipulation du programme entant qu'objet mathématique. Un programme (i.e. une fonction) peut être étudié de façon formelle pour prouver certaines propriétés (de terminaison par exemple) ou encore être transformé de façon tout aussi formelle pour obtenir un programme équivalent possédant des propriétés meilleures. **[STO77].**

Enfin, la troisième qualité, concerne la facilité de mise en œuvre du parallélisme due, entre autre, à l'absence d'effet de bord qui rend délicates les opérations d'entrées-sorties et de traitement des exceptions.

3.3. Les langages fonctionnels : principes & théorie

Les langages fonctionnels font partie des langages déclaratifs les plus utilisés dans plusieurs domaines tels que l'intelligence artificielle, le calcul symbolique, etc. Ils se basent sur la notion mathématique de fonctions qui recevant leurs arguments retournent un résultat. Ces langages sont donc proches des mathématiques, par conséquent le passage d'une spécification de problème vers un programme fonctionnel est plus direct et sa correction est relativement plus facile à prouver.

3.3.1. Définition des langages fonctionnels

Un langage est dit fonctionnel s'il respecte les deux conditions suivantes :

1. La structure de base pour la construction d'expression est la fonction.
2. La structure de contrôle de base est l'application de fonction.

Le sens donné au mot fonction est celui habituellement utilisé dans le domaine mathématique.

$$f : \mathbf{E} \rightarrow \mathbf{F}$$
$$x \mapsto f(x)$$

L'implémentation des fonctions dans les langages fonctionnels est réalisée de telle sorte qu'elle soit la plus proche possible de la notion de fonction mathématique.

Une fonction véhicule les informations suivantes :

- Liaison lexicale : le sens de la fonction est déduit du sens des constructeurs qu'elle utilise.
- L'abstraction fonctionnelle : qui exprime la sémantique du corps de la fonction en termes de paramètres formels et de constantes.
- En plus de ces informations, un langage fonctionnel typé véhicule la notion de type : domaine et codomaine de $f : A \rightarrow B$, A est le domaine ; B est le codomaine.
- Un programme écrit dans un langage fonctionnel est une fonction définie n termes d'autres fonctions. Ces fonctions sont dites "*objets de première classe*".

Un objet est dit de "*première classe*" s'il satisfait les conditions suivantes :

1. Peut être stocké dans des structures de données (ou variables) et transféré d'une structure à une autre.
2. Peut être passé en paramètres à des fonctions.
3. Peut être retourné comme résultat d'une fonction.

3.3.2. Classification des langages fonctionnels

L'approche de la programmation fonctionnelle, à travers les différents langages existants, fournit une vision différente de la programmation. Elle exprime à la machine comment résoudre un problème et non pas les étapes techniques de mise en œuvre.

Les langages fonctionnels peuvent être vus comme des variations syntaxiques et peu sémantiques les uns **[PEY 90]**. Malgré ces traits communs, ils présentent certaines caractéristiques distinctives. Plusieurs classifications existent. Pour notre part, on les distingue à travers trois caractéristiques essentielles :

- La nature pure ou impure du langage,
- Le typage statique ou dynamique du langage,
- Le caractère strict ou non strict (paresseux) du langage.

Cette section introduit brièvement ces notions.

3.3.2.1. Langages fonctionnels purs, langages fonctionnels impurs

Un langage fonctionnel pur est un langage où l'affectation (ou tout autre débordement externe) n'existe pas, ***Miranda*** à titre d'exemple. L'ordre d'évaluation des expressions n'a aucune influence sur le résultat final.

Les langages fonctionnels impurs, comme ***ML*** ou ***Scheme***, intègrent des traits impératifs comme l'affectation, permettant ainsi l'écriture de programme dans un style plus proche de la programmation impérative où l'ordre d'évaluation des expressions est important.

3.3.2.2. Typage dynamique versus Typage statique

On peut répartir les langages fonctionnels en ceux typés dynamiquement (les noms d'objets ou "variables" n'ont pas d'attribut type, mais les valeurs sont typées) tels que **Lisp, Scheme,…** et ceux typés statiquement (chaque "variable" se voit attribuer un type lors de la compilation) : ***Haskell, CAML,*** etc. Les langages typés statiquement sont d'habitude plus efficaces (rapides), car on peut éviter beaucoup de tests durant l'exécution du programme. Ils permettent aussi aux utilisateurs la définition des types abstraits de données.

La plupart des langages de la famille *ML* possèdent une discipline de typage statique et implicite. Statique car le typage des expressions manipulées se fait avant exécution. Mieux encore, une expression dont le typage échoue est tout simplement rejetée et non évaluée. Implicite car le programmeur n'a pas à préciser le type de chaque identificateur introduit dans son code, c'est à la charge de l'inférence de type **[MEN 94]** de le déterminer.

3.3.2.3. Caractère strict versus caractère non-strict [PEY 90, MAE 02]

Lors de l'implémentation d'un langage fonctionnel, un choix de conception est effectué afin de préciser le mode d'évaluation des expressions, qui à son tour détermine la sémantique opérationnelle du langage. En effet, selon le choix entrepris, un même programme peut dans un cas terminer et dans l'autre échouer.

On présente, les caractères strict et non strict des langages fonctionnels.

- **Caractère strict :** Une fonction est dite stricte si elle exige de connaître la valeur de ses arguments. Associé souvent à l'évaluation stricte ou appel par valeur, ce choix impose l'évaluation des arguments avant d'évaluer la fonction. L'échec de l'évaluation de l'un des arguments assure que la fonction concernée ne soit jamais évaluée. *ML* et ses dialectes (*CAML, CAML Light*), *Hope*, … sont des langages stricts.

- **Caractère non strict :** Ce caractère est associé à l'évaluation paresseuse ou l'appel par nom ou encore par nécessité. L'évaluation d'un argument est retardée jusqu'à ce que sa valeur soit nécessaire. Cette évaluation ne se fait qu'une seule fois (cas d'une évaluation paresseuse), les autres utilisations de l'argument auront accès à la valeur initialement calculée.

Ce programme est tout à fait logique dans le cadre des langages fonctionnels purs, où effectivement, un objet ne change pas de valeur tout le long du programme. *Miranda, Haskell,…* sont des langages non stricts (paresseux).

Remarque : Il existe des langages fonctionnels hybrides qui supportent à la fois les deux modes d'évaluation vus ci-dessus. Les arguments simples (constantes entières, caractères,…) sont évalués avant la fonction, tandis que les arguments complexes (liste, structure de données récursive,…) sont évalués selon le besoin

3.3.3. Propriétés des langages fonctionnels

Certaines propriétés des langages fonctionnels purs, telles que l'absence des effets de bord et la transparence référentielle, ont pour conséquences :

- L'absence de la notion de variable c'est-à-dire celle mise à jour du continu d'un objet typiquement par l'instruction d'affectation ; elle facilite la détermination de la

sémantique de programmes, car ce dernier revient à l'application de fonctions à ses arguments sans aucun effet de bord.

- La valeur d'une expression ne dépend que de la valeur de ses sous-expressions. Ce comportement minimise l'importance de l'ordre d'évaluation des sous-expressions en augmentant ainsi les possibilités de parallélisation.

- Comme ce modèle est plus proche des mathématiques, l'application des mêmes techniques de preuves et de raisonnements est possible.

Une étude des notions d'effets de bord et de transparence référentielle est présentée dans cette section, mais aussi le polymorphisme des types dans les langages fonctionnels en général.

3.3.3.1. Absence des effets de bord

Les langages fonctionnels purs ne définissent pas de variables, mais ils utilisent des identificateurs pour nommer les différents objets manipulés dans le programme. Ces identificateurs sont par la suite liés une bonne fois pour toute à des valeurs qui peuvent être des constantes ou des valeurs fonctionnelles.

Définition : Un effet de bord (*side effect*) est l'opération qui permet à un programme de communiquer avec son monde extérieur, essentiellement via des tâches d'entrées/sorties et des modifications de cases mémoires.

Les effets de bord qu'on peut citer sont :

- La modification du continu d'une variable par l'affectation.
- Les entrées à partir des fichiers, du clavier, périphériques d'acquisition de données, etc.
- Les sorties vers des fichiers, l'écran, l'imprimante, périphériques à réglage automatique, etc.

3.3.3.2. Transparence référentielle

Définition 1 : C'est le principe sur lequel "le résultat du programme ne change pas si on remplace une expression donnée par une autre de même valeur".

Définition 2 : [HAR93] Toute occurrence d'un même nom dans une expression désigne la même valeur.

Remarque : Ce principe permet la preuve de correction du programme et aide à améliorer les performances par transformation.

Exemple :

*f(x) + f(x) → 2 * f(x)*

Cette transparence est inexistante dans les langages impératifs à cause des effets de bord (affectation).

Exemple : soit l'expression *y = f a ;*

Dans un langage fonctionnel pur, les expressions suivantes sont toutes équivalentes :

*y + y ⇔ 2 * y ⇔ y + (f a) ⇔ (f a) + (f a).*

Par contre, dans un langage impératif (exemple le langage C) :

Soit le programme suivant :

```
int w = 0 ;
int f ( int x ) { return x + w ++ ; }
```

La valeur de *f (0) + f (0) ≠ 2* f (0)* :

f (0) + f (0) = 0 + 1 = 1
2 * f (0) = 2 * 0 = 0

3.3.3.3. Polymorphisme

Les langages fonctionnels permettent une programmation générique plus simple et plus accessible, grâce au typage polymorphe dans lequel les types des expressions peuvent rester, à priori, indéterminés, et qui peuvent être associés à n'importe quel type défini.

Donc, la programmation de fonction peut se faire sans se soucier du type des objets sur lesquels le traitement sera effectué, par exemple une fonction de tri peut s'écrire telle qu'elle puisse traiter les tableaux d'éléments de n'importe quel type.

Chapitre 4

Fonction et composition de fonctions

4.1. Définition d'une Fonction

Le concept de programmation fonctionnelle repose sur la notion mathématique de fonction, d'où la nécessité de définir cette dernière.

Considérons l'expression : $(x + 1)$

Chaque fois qu'on remplace x par un nombre entier et qu'on évalue l'expression, on obtient un nombre unique. Cette expression peut être assimilée à la machine de la figure 9 qui, pour tout élément x soumis en entrée, en produit le successeur.

Figure 9 - Machine à successeurs

On peut constater que x et son successeur sont tous deux des entiers naturels. Dans le langage mathématique, on parlerait d'une fonction *Succ* de l'ensemble des entiers naturels vers lui-même, notée :

$$\begin{aligned} \text{Succ}: \quad & \mathbb{N} \rightarrow \mathbb{N} \\ & x \rightarrow (x + 1) \end{aligned}$$

Où x est la variable et $(x + 1)$ la valeur de la fonction. Dans les langages de programmation, on emploie plutôt le terme paramètre en lieu et place de variable. De façon plus concise, on note :

$$\text{Succ}\ (x) = (x + 1)$$

Une fonction est une relation qui, à un ensemble de variables d'entrée appartenant à son domaine de définition, fait correspondre une unique valeur appartenant à son domaine image, noté *f(x)*. Cette unique valeur, appelée valeur de la fonction ou bien résultat de l'application de *f* à l'élément *x*, est alors désignée par le nom de cette fonction. **[PIE91]**

Plus formellement, on peut définir une fonction F par la notation suivante :

$$\begin{aligned} \text{F}: \quad & \text{A} \rightarrow \text{B} \\ & x \rightarrow \text{F}(x) \end{aligned}$$

Ce qui se lit : F est une fonction définie dans A à valeur dans B, qui à tout x pris dans A fait correspondre un et un seul F(x) pris dans B. Intuitivement, une fonction est une manière

d'associer de façon unique des éléments d'un ensemble (de départ) à des éléments d'un autre ensemble (d'arrivée).

Autrement dit, une fonction *f* à *n* variables est une correspondance notée:

f : *E*1 × … × *En* → *F* qui à un élément du produit cartésien des *n* ensembles *E*1, …, *En* fait correspondre un élément de l'ensemble *F*. *E*1 × … × *En* est l'**ensemble de départ** de la fonction *f* et *F* est son **ensemble d'arrivée**. Par exemple :

- la fonction à une variable *partie entière* : **D** → **Z** qui à un nombre décimal *d* fait correspondre sa partie entière, qui est un entier relatif ;
- la fonction à deux variables *min* : **Z** × **Z** → **Z** qui à une paire de nombres entiers relatifs fait correspondre le plus petit de ces deux nombres.

➢ **Fonction identité**

Il existe une fonction particulière, appelée *fonction identité* et notée I, qui possède la propriété de laisser inchangée la variable qui lui sert d'argument. On peut la définir de la manière suivante :

$$I(x) = x$$

➢ **Fonction réciproque**

On dit que deux fonctions F et F' sont inverses l'une de l'autre *(fonctions réciproques)*, si et seulement si:

$$F(F'(x)) = F'(F(x)) = I(x).$$

À titre d'exemple, la fonction prédécesseur, notée *Pred*, est l'inverse de la fonction successeur *Succ*. Car :

$$Pred(Succ(x)) = Succ(Pred(x)) = x$$

➢ **Récursivité**

Les fonctions récursives **[FIL 06]** jouent un rôle clé en programmation fonctionnelle. Elles permettent notamment d'exprimer la répétition sans itération et l'appel de fonction coûte très peu cher,

Lorsque l'on définit des fonctions, il est souvent utile de faire appel à la fonction elle-même dans sa propre définition. On parle alors de **définition récursive.** Il est nécessaire également de pouvoir définir des fonctions **mutuellement récursives** correspondant par exemple à des **récurrences croisées**. Plusieurs fonctions sont dites **mutuellement récursives** lorsqu'elles s'appellent de façon croisée c.-à-d. lorsque le nom de l'une peut apparaître dans le corps de l'autre et inversement.

L'appel d'une fonction récursive déclenche une succession d'appels de cette fonction jusqu'à atteindre le cas de base à partir duquel il est possible de terminer l'évaluation de ces appels dans l'ordre inverse de leur déclenchement. C'est la structure en pile de l'environnement qui permet de conserver la mémoire des appels successifs de la fonction.

- **Curryfication**

Considérons la fonction mathématique :

$$f: \; Z \times Z \rightarrow Z$$
$$(x, y) \rightarrow f(x, y) = x \times y$$

Cette fonction a deux arguments peut se représenter à l'aide de la fonction unaire F définie par :

$$F : \; Z \rightarrow (Z \Rightarrow Z)$$
$$x \rightarrow F(x) = f\,x$$

Où la fonction $f\,x$ est définie par

$$f\,x : Z \rightarrow Z$$
$$y \rightarrow f\,x(y) = x \times y$$

La fonction F est une fonction d'ordre supérieur puisque pour tout x, $F(x)$ est une fonction. F définit en fait une famille de fonctions $(f\ x)$, indexée sur Z. Elle est appelée **"curryfiée"** de f (du nom du mathématicien Haskell Curry).

Les avantages principaux de curryfication sont :

- la curryfication aide à diminuer le nombre de parenthèses nécessaires dans une expression, et
- les arguments d'une fonction curryfiées sont attribués l'un après l'autre.

La programmation fonctionnelle permet d'exprimer des fonctions sur des données et de passer ces fonctions en argument ou de les recevoir en résultat.

Dans un langage purement fonctionnel, nous ne manipulons que des valeurs. Un programme désigne une fonction qui calcule une valeur à partir de ses opérandes. Chaque pas d'exécution d'un programme fonctionnel, consiste en une application d'une fonction sur les valeurs de ses opérandes déjà calculées. L'ordre dans lequel sont calculés tous les sous termes du terme qui constitue le programme fonctionnel, est appelé "ordre de réduction". **[GUE96]**

En programmation on s'intéresse aux fonctions dont l'élément d'arrivée est calculé par l'**application** d'une opération à l'élément de départ. La définition de cette opération, appelée **corps** de la fonction, est exprimée en termes du langage de programmation utilisé.

Les ensembles mis en correspondance par les fonctions sont les ensembles de valeurs de chaque type ; les variables d'une fonction sont appelées ses **arguments formels** et les valeurs de ces variables auxquelles s'applique la fonction sont appelées ses **arguments effectifs.**

4.2. Appel d'une fonction

L'appel de la fonction est réalisé en faisant intervenir, dans une expression, le nom de cette fonction suivi de la liste de ses paramètres effectifs. Comme dans la notation usuelle en mathématique, les paramètres sont situés entre parenthèses:

...← ... × <Nom_fonction> (<liste de paramètre effectifs>) + ...

Le calcul s'y effectue par appel d'une fonction avec ses arguments qui peuvent eux-mêmes être des appels de fonctions et ainsi de suite, jusqu'à trouver des données.

Une fonction se termine en retournant ou non une valeur. La valeur d'un appel de fonction est la valeur retournée par l'exécution de son corps à partir d'un état dans lequel les arguments effectifs sont affectés aux arguments formels. Cette valeur est unique. Le retour de la valeur signifie l'arrêt de la fonction.

4.3. L'environnement des données

En programmation fonctionnelle, une variable est une liaison entre un nom et une valeur. Les variables peuvent être globales, et elles sont alors connues de toutes les expressions qui suivent la déclaration, ou bien locales à une expression, et dans ce cas elles ne sont connues que de l'expression pour laquelle elles ont été déclarées.

L'ensemble des variables connues d'une expression est appelé environnement de l'expression.

Les paramètres :

Le programme appelant doit donner à certaines fonctions des valeurs pour effectuer ses calculs. La fonction associe à ses valeurs des variables afin de les manipuler : ce sont les paramètres de la fonction. Un paramètre est une variable locale à une fonction. Il possède dès le début de la fonction la valeur passée par le programme appelant.

Les données d'une fonction

Dans l'utilisation et l'écriture d'une fonction, la plus grande difficulté est de comprendre l'ensemble des données auxquelles la fonction a accès.

Un environnement de données, appelé aussi espace d'adressage, correspond à l'ensemble des variables associées exclusivement à un programme ou à une fonction.

Il est tout à fait possible que deux variables, l'une déclarée dans le programme appelant et l'autre déclarée dans la fonction, portent le même nom. Elles peuvent être du même type ou non, peu importe, puisqu'elles sont utilisées de manière différente dans des environnements de données différents.

Le programme et la fonction sont des "boîtes indépendantes" représentant les espaces d'adressage indépendants. Il n'existe aucun moyen pour le programme d'avoir accès aux variables de la fonction, ni à la fonction d'avoir accès aux variables du programme. Le seul échange se fait:

- du programme vers la fonction, en passant des valeurs grâce aux paramètres;
- de la fonction vers le programme en retournant une seule et unique valeur.

Les paramètres et les variables

La plupart du temps, l'exécution d'une fonction est paramétrable grâce à des valeurs qui lui sont passés. Les paramètres sont des variables de la fonction : il est donc faux de vouloir les redéfinir dans la zone de déclaration des variables.

Une fonction peut accéder à deux types de données:

- Les paramètres, dont les valeurs sont connues dès le début de la fonction. Les valeurs sont passées en paramètres. Il est inutile de nommer les paramètres avec le même nom que les variables utilisées lors de l'appel de la fonction.
- Les variables (appelée variables locales) définies dans le bloc de déclaration des variables.

Au cours de l'exécution d'une fonction, les variables définies dans le programme appelant sont inconnues: aussi bien leur nom que leur valeur.

Passage d'arguments

La valeur d'un argument est recopiée dans l'environnement dans lequel est évaluée la fonction.

Le passage d'un argument par valeur souffre de deux défauts :

- une fonction ne peut pas modifier la valeur d'une donnée dont le nom lui a été passé en argument puisqu'elle n'a accès qu'à la copie de la valeur de cet argument;
- si l'argument est le nom d'une structure volumineuse, sa copie peut être coûteuse en temps.

Il est donc intéressant, dans ces deux cas, qu'une fonction puisse modifier une donnée définie dans le corps de la fonction appelante. Il suffit pour cela de lui passer en argument l'adresse de cette donnée au lieu de son nom. On dit alors que l'argument est passé **par adresse**. Lors d'un appel d'une fonction, c'est l'adresse de chaque paramètre effectif qui est passé à la fonction, et non pas sa valeur. C'est l'appel **par référence.**

4.4. Typage

Le typage est une discipline permettant d'associer des spécifications, ou *types*, aux programmes. Un type est typiquement un terme de petite taille. Le type attribué à un fragment de programme n'est fonction que des types attribués aux sous fragments qui le composent. Le type d'une fonction est déterminé par le type de son (ou ses) argument(s) et celui des valeurs qu'elle renvoie.

La notion clé dans tout langage fonctionnel est celle d'expression. Toute expression possède une valeur et un type. La notion de type permet de regrouper les objets en classe ayant des caractéristiques communes et caractérise les opérations primitives qui leurs sont applicables. Ecrire un programme consiste à définir une expression dont la valeur est la solution au problème posé. Parmi les types de base on trouve notamment : int, float, bool et string.

L'intérêt du typage est multiple. Le typage permet de vérifier si un argument passé à une fonction est bien du type du paramètre formel de la fonction. On peut faire cette vérification à l'exécution du programme. On appellera alors cette vérification, comme citée ci-dessus, le typage dynamique. En cas d'erreur sur les types le programme s'arrêtera dans un état cohérent.

Cette vérification peut aussi se faire avant l'exécution du programme, c'est-à-dire au moment de la compilation. On appelle cette vérification a priori le typage statique. Ce mode de typage garantit qu'un programme bien typé ne peut pas planter, et son exécution ne peut pas échouer inopinément, d'autre part, il fournit à l'utilisateur une garantie partielle de robustesse, ainsi qu'une propriété de *sécurité* certes limitée, mais sans laquelle aucune garantie plus poussée ne peut être fournie.

Enfin, de par son caractère compositionnel, et de par l'exploitation des notions duales de *polymorphisme* et d'*abstraction* de types, le typage favorise la *modularité*, c'est-à-dire le découpage des programmes en unités indépendantes et complémentaires.

Un langage est dit fortement typé si toute expression possède exactement un type qui définit précisément quelles opérations on peut lui appliquer.

On dit que les types des programmes sont **synthétisés [PRT01]** (ou encore **inférés**) lorsque le compilateur déduit de lui-même tout ou partie des types qui interviennent dans le programme, ce qui libère le programmeur de l'obligation de spécifier les types que ses programmes manipulent.

Un type est dit **polymorphe [CPF02]** lorsque certaines de ses composantes sont des variables qui représentent toute une famille de types (obtenus par **instanciation** de ces variables, c'est-à-dire remplacement de ces variables par d'autres types).

Si l'ensemble de départ d'une fonction "Min" par exemple est *entier* × *entier* et son ensemble d'arrivée *entier*, nous dirons que son type est *entier* × *entier* → *entier*. Dans le cas général, l'ensemble de départ est un produit cartésien et on appelle *arité* la dimension de ce produit cartésien. La fonction "Min" est d'arité 2; on dira qu'elle prend deux *paramètres* (ou arguments) et qu'elle délivre un *résultat*. Le type d'une fonction est la donnée du type de ses paramètres (son ensemble de départ, en général un produit cartésien) et du type de son résultat (son ensemble d'arrivée). Il se note par: D1 x … x Dn →A

Une expression représente toujours une seule valeur mais chaque valeur peut être représentée par plusieurs expressions différentes. Chaque valeur appartient à un type.

Une façon de créer de nouveaux types est de grouper des types existants :

Soient A et B deux types, le type (A, B) représente l'ensemble de pairs d'un élément de A et d'un élément de B. Ainsi, le type A –> B représente l'ensemble de fonctions de A vers B. On observe donc qu'une fonction est considérée comme une valeur.

La notion de type apporte une sécurité de programmation indispensable dans la réalisation de gros systèmes. De plus, elle permet de rendre l'exécution des programmes plus efficace.

4.5. La composition de fonctions

Le paradigme fonctionnel n'utilise pas de machine d'états pour décrire un programme, mais un emboîtement de fonctions que l'on peut voir comme des « boîtes noires » que l'on peut imbriquer les unes dans les autres. Chaque boîte possédant plusieurs paramètres en entrée mais une seule sortie, elle ne peut sortir qu'une seule valeur possible pour chaque tuple de valeurs présentées en entrée.

Une fonction regroupe une ou plusieurs expressions qui permettent de réaliser une tâche précise. Elle possède un nom qui permet de l'appeler. Les fonctions favorisent la modularité des programmes. Cela signifie qu'un programme se compose de plusieurs fonctions qui s'appellent entre elles. Ce mode de fonctionnement facilite la maintenance d'un programme dans la mesure où l'application n'est pas symbolisée par un bloc monolithique, mais par un ensemble de petites unités de code sur lesquelles il est facile d'agir. Ces fonctions peuvent constituer les éléments d'une boîte à outils qui pourront être composés et réutilisés pour les nouveaux développements.

La programmation fonctionnelle consiste à utiliser des langages dont la structure et la syntaxe reposent sur les concepts mathématiques de fonction et de composition de fonctions. La composition est bien entendu l'une des opérations de base d'un langage fonctionnel. On peut définir la *composition de fonctions*, vulgairement appelée une fonction de fonction. En fait, c'est une fonction dont la variable est elle-même une fonction. Par exemple, le successeur du successeur de x, noté :

$$\mathrm{Succ}(\mathrm{Succ}(x))$$

Représentons la composition de deux fonctions f et g par un point associatif à droite, et définissons :

$$(f \,.\, g)\; x = f\;(g\;\; x).$$

La composition est définie tant que ses arguments de gauche et de droite sont des fonctions, et le type des arguments de son argument de gauche est le même que le type des résultats de son argument de droite.

Considérons le langage fonctionnel minimaliste suivant :

1- la constante 0.
2- la fonction successeur S : x --> x + 1.
3- la composition de fonctions. Cela consiste en une combinaison de fonctions emboîtées et de variables substituées, permutées ou rajoutées. Par exemple :

0, (x,y) --> f(x,y), et (x,y) --> g(x,y) peuvent donner naissance par composition à la fonction (x,y,z,t) --> g(f(y,0),f(g(x,t),y)). C'est simplement la possibilité de programmer de nouvelles fonctions à partir des constantes et fonctions programmées précédemment en les combinant sans utiliser ni condition ni boucle répétitive.

4.6. Conclusion

On a mis l'accent dans cette partie sur les concepts essentiels offerts par le paradigme fonctionnel. Nous avons abordé un style de programmation qui utilise comme concepts essentiels la définition et l'application de fonctions. Dans cette approche [COU95], les variables jouent le même rôle qu'en mathématiques: elles sont une représentation symbolique des valeurs et servent en particulier à noter les paramètres des fonctions.

La programmation fonctionnelle supprime la notion de variable, et décrit un calcul par une fonction qui s'applique sur des données d'entrées et fournit comme résultat les données en sortie. De fait, ce type de programmation est plus abstrait car il faut décrire l'algorithme indépendamment des données.

La programmation fonctionnelle se veut un outil de haut niveau de description utilisant des concepts évolués (modularité, Abstraction, Fonctions d'ordre supérieur, etc.) et indépendant de toute implémentation.

La partie suivante est consacrée au système FS4SWC que nous proposons pour composer de manière automatique les services web, où nous introduirons plus en détail la solution à base du formalisme fonctionnel ainsi que la formulation de l'ensemble des services qui peuvent être rendus dans une composition à partir d'un ensemble existant.

Chapitre 5
Proposition d'une approche pour la composition automatique des services Web Approche par les fonctions

5.1. Introduction

L'étude des travaux existants a montré qu'à ce jour différentes dimensions de composition des services web font l'objet d'étude. Cependant, ce problème de composition automatique de services Web est par nature très difficile. En effet les données sont instables, le Web étant dynamique, de plus, comme l'a résume Wielinga et al dans **[THW 04]** : "Le problème de création d'un nouveau Web service composite est en principe égal au problème classique de programmation automatique généralisée. Ce problème est insolvable de manière notoire en général par aucune technique connue. Il n'y a pas de raison de croire que la version Web service de ce problème soit moins résistante à une solution générale." [1], et non seulement il n'y a pas une structure globale représentant l'ensemble des services disponibles.

Notre principale contribution est de proposer un système complet pour la composition, car nous considérons qu'une composition de services web doit s'engager dès la découverte de services jusqu'à l'interaction avec les utilisateurs. Un deuxième point sur le système proposé est que la composition peut être faite d'une façon transparente. Du point de vue de l'utilisateur, une fois que la requête est définie, le système s'engage à composer les services nécessaires, et lui proposer à la fin les compositions trouvées. Afin d'atteindre ces objectifs, nous proposons **FS4WSC** *(Functional System For Web Service Composition)*, un système pour composer automatiquement les services web sémantiques (SWS) basé sur une description d'un modèle dans le contexte de la composition des services web.

1 Ce qui est la traduction du texte suivant :"This problem of creation of new composite Web service is in principle equal to the old problem of generalized automatic programming. This problem is notoriously unsolved in general by any known techniques. There is no reason to believe that the Web service version of this problem will be any less resistant to a general solution.".

Dans cette partie, nous soulignons tout d'abord que notre travail se base sur une structure générale représentant l'ensemble des services Web disponibles, modélisés sous forme de fonction. Dans un second temps, nous précisons que la base de notre travail est les services web découverts issus de la requête de client et leurs compositions afin de satisfaire les besoins de client.

Nous allons tout d'abord définir plus précisément l'architecture de notre système ainsi que les modules qui la constitue. Ensuite, nous définissons un formalisme pour représenter les SWS puis nous formaliserons notre problème à l'aide de ce formalisme dans le but de proposer deux algorithmes qui répondent à nos motivations. L'objectif de ces algorithmes est de sélectionner l'ensemble des services Web sémantiques qui permettent de rendre le service souhaité par l'utilisateur (appelé requête) à partir de compositions de services existants. Notons qu'il s'agit en fait de comparer la requête avec chaque service existant puis retourner les services simples et/ou composites qui peuvent satisfaire la demande du client.

5.2. Principe

Les services Web sont considérés sous l'angle de leurs entrées et sorties (résultats). Ces entrées et ces sorties donnant la sémantique des services Web. L'accent est mis sur l'interaction avec un client. L'interaction entre un client et un service Web est le fait qu'un client déclenche une fonctionnalité du service Web et utilise le résultat pour continuer (but à atteindre et/ou information). Un service Web possède un comportement avec un client qui peut être modélisé techniquement par des graphes, des diagrammes d'activité UML, des machines à états fini (FSM), des fonctions…etc. Les fonctions sont retenues pour modéliser les services Web car elles permettent de décrire de manière compacte toutes les exécutions possibles.

A partir des descriptions de comportement d'un ensemble de services Web, il s'agit de trouver une composition satisfaisant un but (lequel modélise les conditions nécessaires d'interaction avec un client). Cependant, la tâche de composition automatique de services Web nécessite une modélisation précise des services Web. Le but est décrit aussi à l'aide d'une fonction. Il s'agit de composer les fonctions selon un algorithme, de manière à ce que la fonction obtenue puisse avoir les mêmes exécutions que la fonction but.

Avant de modéliser techniquement les services Web, nous présenterons d'abord l'architecture proposée.

5.3. Proposition d'architecture

Notre architecture (*voir **figure 10***) se compose de trois sous systèmes :

1. **L'architecture de référence qui est basée sur les trois entités** : Le *fournisseur* construit le service Web, il le publie dans le *registre* de services et le *demandeur* de service Web recherche dans ce registre les services répondant à ses besoins. Pour implémenter cette architecture et garantir l'interopérabilité des trois opérations précédentes (publication, recherche et lien), des propositions de standards ont été élaborées pour chaque type d'interactions comme nous l'avons déjà mentionné dans la *Première partie*.

2. **Système d'annotation sémantique** : notre contribution réside dans l'ajout, à cette architecture, un composant dédié à l'annotation des services Web. Nous expliquons ci-dessous son rôle et son fonctionnement en décrivant les différentes étapes composant le processus de publication et de recherche de services Web sémantiques. Ces étapes sont identifiées dans la **figure 10** par des numéros encerclés auxquels nous faisons référence par la suite pour décrire ce processus.

3. **Module de mise en correspondance et le moteur de composition** : ce sous système est défini pour permettre la recherche de services Web sémantiques adéquat à la requête. Le module de mise en correspondance effectue les correspondances entre la description d'un SWS et la demande de ce dernier en renvoyant au demandeur un fichier XML en résultat si la requête est satisfaite par un seul service Web. Le processus de correspondance s'appuie sur des règles qui exploitent l'ontologie, les profils des services et la requête du client. Dans le cas où la requête ne peut pas être satisfaite par un seul service, une composition des SWS retournés par le module de mise en correspondance est faite à l'aide du moteur de composition.

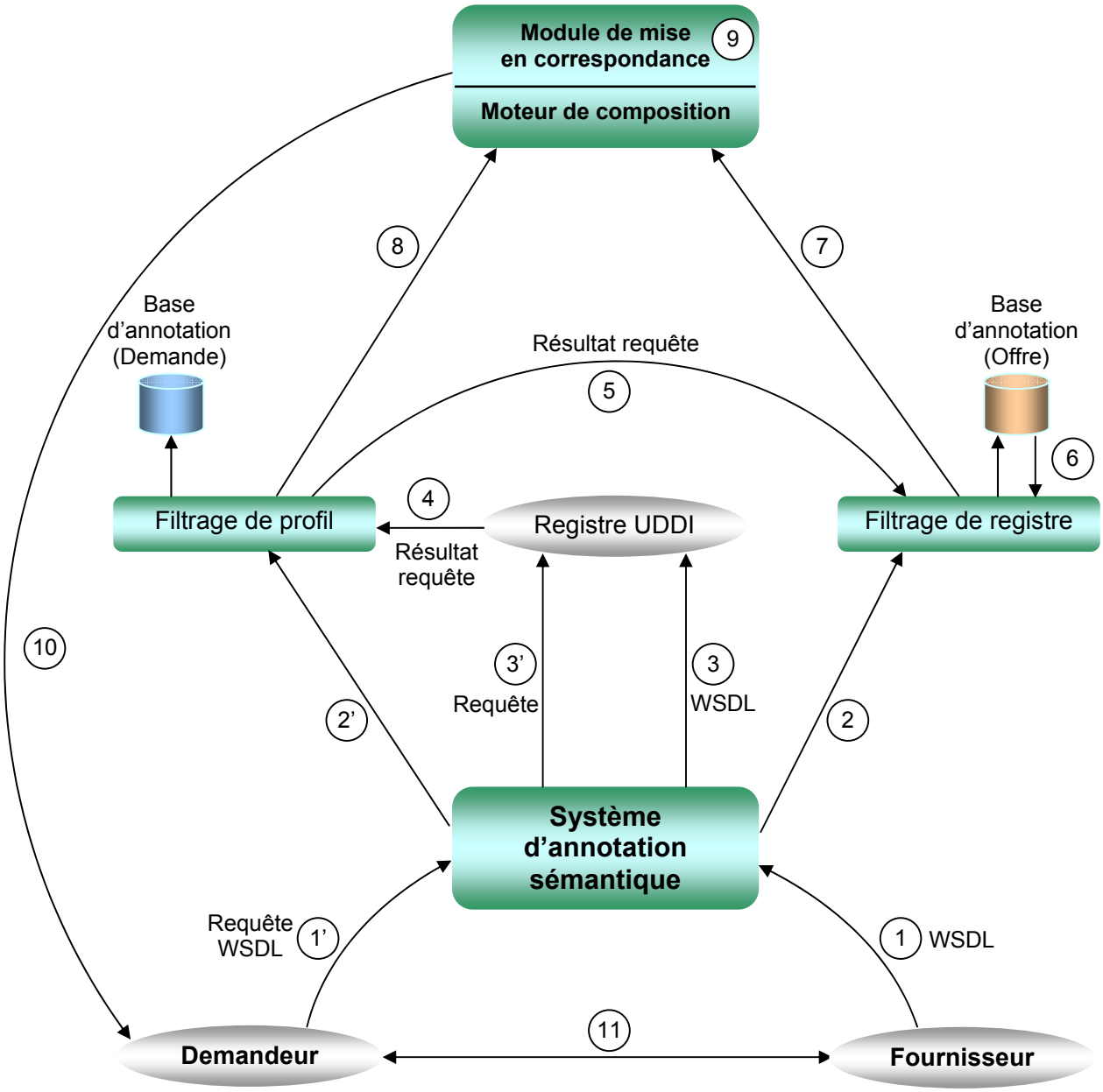


Figure 10 - Proposition d'architecture -

L'utilisation de l'architecture classique des services Web et l'intégration du formalisme de la sémantique ne modifie pas le système existant et facilite la mise en œuvre de cette architecture.

① Le fournisseur transmet la description du service Web sémantique. Un système d'annotation annote sémantiquement les noms et les entrées/sorties des demandes et des offres de services par des concepts ayant une description formelle afin d'être interprétables par une machine. Les concepts sémantiques sont utilisés entre autres pour caractériser les mots voisins (synonymes) des noms et des variables d'entrées/sorties des services Web, les différentes variantes (unités de mesure, types de données abstraits) dans lesquelles sont exprimées les valeurs des entrées/sorties des services Web ainsi que leurs types. ②Les services Web annotés sont envoyés à un composant de filtrage qui extrait la description sémantique des services et il les stocke dans une base d'annotation *(Offre)*.③ La description classique WSDL est ensuite transmise au registre UDDI, comme dans l'architecture de référence.①'

Du côté du demandeur, de la même manière que nous avons expliqué dans ① et ②, l'utilisateur soumet sa requête de service au même système d'annotation qui annote la requête puis ②' la soumet à un second composant de filtrage. Celui-ci stocke la description sémantique du service attendu dans une autre base d'annotation *(Demande)*. ③'La requête du service, sans la description sémantique, est ensuite transmise de manière classique au registre UDDI qui procède à l'exécution de la requête. ④Le résultat de la requête (la liste des services répondant à la demande de l'utilisateur) est intercepté par le module de *filtrage de profil* qui génère un fichier XML comportant les descriptions attendues.

⑤La liste des services est communiquée au module de *filtrage de registre* afin qu'il recherche, si elles existent, leurs descriptions sémantiques. ⑥Le module de *filtrage de registre* accède à la base d'annotation *(Offre)* pour rechercher les services Web sémantique SWS adéquats *(voir le fonctionnement dans **5.3.1)*** puis génère un fichier XML comportant les descriptions des SWS. Ce fichier XML d'une part⑦, et le fichier XML des demandes de l'utilisateur d'autre part ⑧, sont transmis au module d'appariement.

⑨L'appariement de services dans ce module s'effectue à deux niveaux : d'abord, la découverte d'un ensemble de services susceptibles de répondre à la requête du client, puis la sélection du service qui puisse, effectivement, y répondre si la requête peut être satisfaite seulement par un seul service. Dans le cas contraire le moteur de composition accède à la liste des services retournée par le module d'appariement afin de composer les services qui peuvent répondre à la requête. Plusieurs compositions peuvent avoir lieu. Nous pensons que même si les services ont été trouvés de manière automatique (invocation et composition comprises), il est nécessaire de proposer aux utilisateurs des compositions optimales, cette phase n'est pas présente dans la liste de nos motivations. Elle concerne l'optimisation des compositions. Des auteurs ont proposé des modèles de qualité de services et des algorithmes d'optimisation pour la composition **[ARD04] [ZEN et al03] [ZEN et al04].** Dans notre travail on s'intéressera à la première composition trouvée.

⑩Le résultat, sous la forme d'un fichier XML, comporte la liste des services répondant aux attentes de l'utilisateur est envoyé au demandeur de services. ⑪ La communication entre le demandeur de service et le fournisseur se fait ensuite de manière traditionnelle par l'intermédiaire du protocole SOAP.

L'explication détaillée de chaque module (système d'annotation, module d'appariement, moteur de composition) est ci-après.

5.3.1 Le système d'annotation sémantique

5.3.1.1. Description et correspondance sémantique

Dans le domaine des services Web, la sélection automatique d'un service pouvant fournir une fonctionnalité recherchée soulève le problème de la standardisation de la description. En effet, alors que de nombreux systèmes peuvent se contenter d'un vocabulaire spécifique pour décrire les fonctionnalités, les services Web s'intéressent à un environnement ouvert dans lequel un demandeur de service peut choisir entre des fonctionnalités fournies par des opérateurs variés, qui n'utilisent pas nécessairement le même vocabulaire.

Actuellement, les architectures orientées services souffrent d'une critique récurrente **[PER et al]** qui concerne l'aspect uniquement syntaxique des concepts de composition, coordination et surveillance c'est à dire les couches hautes de l'architecture, ce qui rend leur mise en oeuvre très complexe .

Dans une telle architecture, les échanges de messages jouent un rôle primordial, et l'objectif est alors d'assurer des interactions et des échanges de messages fiables afin de mieux répondre aux besoins des utilisateurs. Les échanges de messages doivent donc être exécutés dans un ordre donné pour atteindre le résultat visé.

Pour orchestrer les messages, ceux-ci peuvent respecter un procédé. WSDL fournit une description concrète mais de bas niveau d'un service Web, en termes de sa localisation, des opérations disponibles et des messages associés ainsi que des types de données et formats de leurs paramètres d'entrée ou de sortie. Comme WSDL **[SW5] [SW6]** ne permet pas de définir un tel procédé, différents langages de programmation ou de composition de services web ont vu le jour pour supporter cette modélisation : WSBPEL **[SW7]**, WS-CDL **[SW8]**. Tous ces langages de définition de procédés proposent des mécanismes permettant de composer et de coordonner des fonctions offertes par des services variés. La composition ou orchestration de services **[GUS et al; 04]** s'intéresse à la définition de services composés à partir de services plus simples. Ils sont décrits sous la forme de *workflows*, qui indiquent précisément les enchaînements d'invocations de services et les informations qui sont transmises de l'un à l'autre. La coordination ou chorégraphie de services définit des protocoles d'interaction entre services sous la forme de conversation **[GUS et al; 04]**. Ces deux aspects permettent la définition d'application mettant en jeu plusieurs services.

Cependant, ces techniques et standards fournissent peu ou pas d'appui pour la sémantique des services participants, de leurs messages et interactions. En plus, ces descriptions sont insuffisantes pour qu'un agent logiciel puisse interpréter la signification réelle des opérations WSDL.

Pour pallier à cette hétérogénéité et ces difficultés, les services Web sémantiques (SWS) ne se contentent pas de définitions syntaxiques, mais apportent des informations permettant de composer, de coordonner, et de surveiller plusieurs services Web.

L'approche des services Web sémantiques propose d'utiliser les techniques de représentation de connaissances issues du domaine du Web sémantique pour permettre une plus grande interopérabilité entre les services Web. Ces techniques exploitent des ontologies, qui visent à représenter formellement des connaissances pour permettre l'interprétation de leur sens par des machines. L'utilisation d'ontologies permet ainsi de fournir des descriptions dans lesquelles ce n'est pas uniquement la syntaxe, mais aussi le sens des termes utilisés qui est exploité pour reconnaître les fonctionnalités proposées. Plusieurs formalismes de descriptions ont été proposés. OWL-S **[ANU et al02]** s'appuie sur le langage officiel de descriptions d'ontologies du Web sémantique, OWL *(Web Ontology Language)* le but est de trouver une collection de services qui peut être combinée, en utilisant des constructeurs du OWL-S **[NAR02 b]**, pour former un service composé qui accomplit la tâche donnée pour obtenir le résultat escompté.

WSMO *(Web Service Modeling Ontology)* **[CHR 02]** est issu d'une initiative européenne pour la standardisation des langages de services Web sémantique et propose une architecture pour le développement de tels services. METEOR-S **[ABH et al04]** propose un système basé sur l'annotation sémantique de descriptions WSDL, de manière à enrichir des services existants avec des informations sémantiques. En utilisant des descriptions sémantiques, il devient possible de sélectionner des fonctionnalités de manière plus flexible.

Dans ce contexte, il est nécessaire de proposer des méthodes (une approche) qui permettent de combiner des outils d'annotation sémantique de services et des algorithmes de mise en correspondance de descriptions de services Web sémantiques de manière à pouvoir composer automatiquement des services en vue d'atteindre des fonctionnalités prédéfinies.

5.3.1.2. Fonctionnement du système d'annotation

Comme nous avons expliqués précédemment, l'annotation sémantique des services web repose sur le même principe utilisé dans les travaux cités dans la *section 2.2.2*.

Le système d'annotation identifie d'abord automatiquement dans un service les éléments pertinents à annoter, puis il exploite une ontologie, qui représente les concepts du domaine et les relations entre les concepts dans un langage de représentation des

connaissances, pour déterminer quels sont les concepts les plus spécifiques possibles de l'ontologie dont l'instance sera utilisée pour annoter chacun de ces éléments.

Le système d'annotation génère alors des annotations sémantiques qui sont des métadonnées sur les éléments d'un service liées à une ontologie.

5.3.1.3. Intégration de la sémantique dans le standard WSDL

La structure du service Web qui est décrite en XML est destinée à être intégré dans WSDL, le standard de description des services web, et utilisée à la fois par les demandeurs et par les fournisseurs de services.

Elle est composée de deux sous-systèmes : Le premier représente la partie d'un service qui sert à sa découverte. Il est défini par le triplet **(C, I, O)**. C est le contexte de la spécification, il est défini par un mot-clé ayant trait au domaine du service spécifié. **I** est la description des variables d'entrée et de leurs types de données abstraits dans l'offre ou dans la demande de services. **O** est la description des variables de sortie et de leurs types de données abstraits dans l'offre ou la demande de services.

Les mots-clés du triplet **(C, I, O)** peuvent être annotés par des concepts formels définis dans une ontologie partagée entre les utilisateurs de services du domaine. Les concepts sémantiques peuvent être utilisés pour la désignation des concepts qui sont proches sémantiquement (synonymes) des mots-clés du triplet, ainsi que les types des entrées/sorties des servies Web. Les types des entrées/sorties ne signifient pas les types de données abstraits (Integer, Real, etc), mais les unités de mesure utilisées pour exprimer les valeurs des entées/sorties dans l'ontologie du domaine.

Exemple :

Soit un service de vente de voitures qui est décrit par son nom *"Voiture"* qui a par exemple comme entrée l'attribut *"Modèle"* et comme sortie les attributs *("Prix","Genre")*, ces derniers peuvent être annotés par les concepts *"Price"* et *"Catégorie"* respectivement. L'unité de mesure de la sortie *Prix* est définie par le concept *Price* dans l'ontologie du domaine. Ce dernier correspond à un ensemble de monnaies, en l'occurrence l'Euro (EUR), le Dollar(USD), le Yen(YEN)…etc.

Le type de données abstraits de ces unités de mesure est :

Type {Euro and Dollar and Yen} : *Real*;

Le fait que les concepts du service Web contiennent plusieurs unités de mesure peut sembler inconsistant, et cela par rapport au fait qu'une valeur ne peut avoir qu'une seule unité de mesure à la fois. Cependant, l'annotation avec ce type de concepts (ensembles d'unités de mesure) sert seulement à un appariement partiel qui détermine les services *susceptibles* de

satisfaire la requête. Il y'a une deuxième phase d'appariement où seuls les services pouvant *certainement* répondre seront sélectionnés. Dans le cas où ces derniers n'existent pas dans l'annuaire, c'est un des services sélectionnés pendant la première phase qui sera transformé afin de pouvoir s'apparier avec la requête.

Le deuxième sous système permet de vérifier la consistance des services découverts dans la première phase pendant le processus d'appariement (l'appariement des *contraintes de typage*). En d'autre termes, la sélection des services qui puissent certainement satisfaire la requête du client, cela en vérifiant les contraintes sur les valeurs des entrées/sorties et les contraintes sur leurs types dans l'ontologie du domaine d'une offre et d'une demande de services afin de conclure qu'il y a similarité de services.

5.3.2. Le processus d'appariement de services Web

Le processus d'appariement est mis en œuvre à l'aide d'un module de mise en correspondance qui a pour but de trouver, à partir d'une requête émise par un demandeur de service Web, celui ou ceux qui correspondent le mieux au profil qui est décrit en WSDL étendue par la sémantique.

L'appariement entre une requête et un service se déroule en deux étapes : Durant la première phase, l'annuaire compare les spécifications des demandes et des offres de services. Il compare syntaxiquement les mots-clés décrivant, respectivement, le triplet **(C, I, O)** de la requête, et les triplets **(C', I', O')** de chaque offre de services disponible dans l'annuaire, et sémantiquement les concepts qui leur sont éventuellement rattachés. Le résultat de cette phase est un ensemble de services susceptibles de satisfaire la requête du client.

La deuxième phase concerne la comparaison des contraintes sémantiques de la demande de services avec les contraintes sémantiques des offres de services sélectionnées dans l'étape précédente. Cette phase comporte également deux étapes : d'abord la comparaison des contraintes de typage des entrées/sorties, ensuite celle des contraintes sur leurs valeurs. Le processus d'appariement des contraintes de typage des entrées/sorties de services concerne la demande de services et l'ensemble des services sélectionnés durant la première phase. Si toutes les entrées/sorties d'une demande et d'une offre de services sont de types similaires, alors il n'y a pas de conflit et le processus d'appariement continue avec leurs contraintes sur les valeurs. Si au contraire, au moins une entrée/sortie possède deux unités de mesures différentes dans la demande et dans l'offre de services, alors un conflit de typage apparaît. Pour l'appariement des contraintes sur les valeurs des entrées/sorties, la résolution peut s'effectuer par des algorithmes de satisfaction de contraintes **[LIU 04]**.

5.3.3. Le moteur de composition

En ce qui concerne les phases de FS4SWC, la phase de découverte a pour but de rechercher des services web candidats qui feront partie de la composition. Ensuite nous avons positionné la phase de composition qui s'occupe de la détermination et de l'organisation des tâches (ce qu'il faut faire) qui seront exécutées par les services candidats, chacun pouvant exécuter une ou plusieurs tâches, après la phase d'appariement. Le moteur de composition propose la composition et l'interopérabilité automatique lorsqu'une tâche est requise comme un objectif, les services web doivent être sélectionnés pour accomplir cette tâche. Un mécanisme de classement peut aider le moteur de composition à sélectionner les services appropriés en attribuant des poids à chaque service qui sont calculés à l'aide des formules mathématiques **[KHK06]**.

Les informations nécessaires pour sélectionner et composer des services seront expliquées par la suite. Le processus d'appariement et de composition de services sera expliqué en détail dans la *section 6.1.1*.

Ainsi et pour soutenir le processus de composition automatique de services web répondant à la requête, nous proposons un formalisme pour représenter les services Web atomiques et composites ainsi que deux algorithmes qui consistent à découvrir une combinaison de services satisfaisant la requête de client et à vérifier l'existence d'une composition de services Web.

5.4. Objectif

L'objectif de ces algorithmes est de donner une exécution cohérente d'une requête d'un client qui ne peut pas être satisfaite par un seul service Web, mais possible par un ensemble de services Web disponibles découverts basant sur la requête de client et les concepts mathématiques de fonction représentant les services web disponibles ainsi que les règles de typage et la composition de fonctions. L'idée derrière ces algorithmes est d'exploiter les règles de typage pour découvrir une combinaison de services satisfaisant la requête de client, ainsi les composer en définissant l'ordre d'exécution.

5.5. Un formalisme pour représenter les services Web

Dans cette partie, nous utilisons l'approche fonctionnelle pour modéliser des services Web, qui sont vus comme des fonctions. Pour représenter et raisonner sur des services Web complexes, il est nécessaire de disposer d'un formalisme permettant de :

- Décrire les services Web atomiques,
- Décrire la composition de services Web.

Le premier point consiste à décrire, comme nous l'avons évoqué au premier chapitre la sémantique des services Web atomiques. Cependant, les spécifications utilisées sont très

riches, nous devons circonscrire d'abord ce qui sera extrait de celles-ci, c'est-à-dire ce que nous voulons représenter.

D'autre part nous voulons disposer d'un langage permettant de composer des services Web. Pour cela nous aurons besoin des constructeurs habituels pour la séquence, le choix et le parallélisme.

Le formalisme recherché doit permettre d'exprimer ces deux aspects : la sémantique des services Web atomiques et la composition des services Web.

Le formalisme que nous présenterons dans la *section 5.6*, a pour but de représenter certaines propriétés des services Web atomiques et composites.

Pour montrer l'utilité et l'applicabilité de notre proposition, nous avons besoin de quelques définitions préalables.

5.5.1 Définitions de base

Définition 5.5.1.1. ***(Service Web)*** un service Web S est défini comme triplet (C, I, O) où C le contexte de la spécification de service, $I = \{ Ii / i > 0 \}$ un ensemble d'entrées de service et $O = \{ Oj / j > 0 \}$ l'ensemble de sorties de service.

Définition 5.5.1.2. ***(Composition sémantique)*** deux services Si et Sj pouvant être composés sémantiquement ensemble et dénoté par $S_i \triangleright S_j$ ssi $\exists I_k \in I_{Si} \; \exists O_l \in O_{Sj} \mid I_k \subseteq O_l$ où *ISi* et *OSj* sont les ensembles d'entrées et sorties de service respectivement, et $\subseteq$ dénote un symbole de matching (appariement) sémantique.

La définition signifie que deux services peuvent se composer si et seulement si les sorties (pas nécessairement tous) d'un service s'apparient sémantiquement avec les entrées de l'autre.

Définition 5.5.1.3. ***(Services Web Atomiques)*** un service Si est atomique :

Si il n'existe pas $S_n \triangleright S_m = S_i$.

Ce qui suit présente la syntaxe qui comprend un vocabulaire et des règles pour former les fonctions. Le vocabulaire est le suivant :

On dispose de :

$\sum F = \{f, h. . .\}$, un ensemble fini de symboles de fonctions,
$\sum C = \{A, B, C, \ldots\}$,
un ensemble fini de symboles de constantes pour représenter les types des fonctions, et V un ensemble infini dénombrable de variables.

On dispose aussi de CE = {→ , × , (,) , : , . , = , ⊗ , o , | | } l'ensemble des constructeurs des expressions de fonctions. Comme nous avons des symboles de fonctions et de variables, nous avons classiquement des fonctions.

Définition 5.5.1.4. ***(Fonction)*** La définition classique d'une fonction reste valable :

- si x, y deux variables appartenant à V, alors une fonction $f(x) = y$ est la fonction qui prend en entrée la valeur de x comme argument et retourne en sortie la valeur de y comme résultat,

- si f est une fonction d'arité n et $x1, x2,.....xn$, y des variables alors $f(x1, x2,.....xn) = y$ est une fonction qui prend en entrée les valeurs de $x1, x2,.....xn$ comme arguments et retourne en sortie la valeur de y comme résultat. En particulier cette valeur peut être une fonction.

Pour former des fonctions composées, nous devons utiliser des fonctions atomiques et/ou fonctions composées.

Définition 5.5.1.5. ***(Composition de fonctions)*** Comme nous l'avons vu précédemment *(chapitre 2 section 2.6)* la composition de deux fonctions est définie comme suit :

- si x est une variable appartenant à V, et f, g et h des fonctions alors la composition des deux fonctions f et g est la fonction h tel que $h\,x = (f\,.\,g)\,x = f\,(g\,x)$. h est la fonction de fonction c'est-à-dire la fonction g est l'argument de la fonction f.

La composition fonctionnelle est dénotée par l'opérateur (.) tel que son type est donné comme suit :

$$(\,.\,) :: (\beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma)$$

La composition fonctionnelle prend la fonction f de type $(\alpha \rightarrow \beta)$, la fonction g de type $(\beta \rightarrow \gamma)$, et retourne la fonction h de type $(\alpha \rightarrow \gamma)$

La composition est définie tant que ses arguments de gauche et de droite sont des fonctions, et le type des arguments de son argument de gauche est le même que le type des résultats de son argument de droite.

- la composition fonctionnelle est une opération associative. Nous avons :

$$(f\,.\,g)\,.\,h = f\,.\,(g\,.\,h)$$

Notre contribution s'articule autour des points suivants :

- Comment établir une structure générale représentant l'ensemble des services disponibles.
- Comment rechercher un service web répondant à un ensemble de critères.
- Comment composer l'ensemble des services découverts en définissant l'ordre de leurs exécutions pour répondre à la requête de client.

5.6. Représentation des SWS comme une fonction

Nous allons présenter dans cette partie une modélisation plus précise des services Web sémantiques afin de représenter les services Web atomiques ainsi que leurs compositions. Nous utiliserons à cet effet l'approche fonctionnelle car elle décrit les interfaces, c'est-à-dire les fonctionnalités des services Web. Cette approche modélise des interfaces simples, proches de la signature de fonction, c'est pour cette raison que nous utilisons l'appellation fonctionnelle. Nous stockons les informations concernant chaque requête et chaque service web sous forme d'une fonction, et en utilisant des règles de typage des données, on peut montrer la compatibilité de deux interfaces (fonctions).

Comme nous l'avons expliqués au précédent chapitre, l'annotation sémantique de descriptions WSDL permet d'enrichir des services existants avec des informations sémantiques ce qui devient possible de sélectionner des fonctionnalités de manière plus flexible. Les demandes et les offres de services sont décrites dans notre formalisme par des fonctions, où chaque élément représente un : nom et entrées/sorties, dans la signature du service. De plus, les noms des services et leurs entrées/sorties peuvent être annotés sémantiquement par des concepts ayant une description formelle afin d'être interprétables par une machine.

Cette section introduit, successivement, une nouvelle structure pour les services Web sémantiques et deux algorithmes qui supportent le processus de découverte et de composition de SWS.

La spécification d'une offre ou d'une demande de services avec un langage fonctionnel est illustrée par la figure 11. La signification de chaque élément est ci-après :

Context	ContNam . AnnotC
Inputs	InputNam . AnnotIn : Type
Outputs	OutputNam . AnnotOut : Type

Figure 11 - Spécification d'une offre et d'une demande de services

Context : Contexte de la spécification d'une demande ou d'une offre de services, il représente un mot-clé ***ContNam*** décrivant ce que fait le service c-à-d le nom du service, dans notre approche il représente le nom d'une fonction qui a comme argument l'élément "Inputs" qui peut être une ou plusieurs variables, et comme résultat l'élément "Outputs" qui peut être aussi une ou plusieurs variables.

Inputs/Outputs : Déclaration des variables d'entrée/sortie d'une spécification de services. ***InputNam/OutputNam*** sont des mots-clés décrivant la liste des variables d'entrées/sorties des services (fonctions dans notre formalisme), respectivement.

Le nom et les entrées sorties du service peuvent être annotés par des concepts sémantiques stockés dans les attributs ***AnnotC, AnnotIn, AnnotOut*** respectivement

AnnotC : Description formelle des concepts servant à l'annotation sémantique des noms des demandes et des offres de services.

AnnotIn, AnnotOut : Description formelle des concepts servant à l'annotation sémantique des entrées/sorties des demandes et des offres de services.

Le rattachement d'un concept C à un mot w se fait sous la forme *w.C*, et qui signifie que le concept C est la description formelle du mot *w*.

Type : Définition des types de données abstraits utilisés dans les paramètres d'entrées et de sorties.

Maintenant que la spécification d'une offre et d'une demande de services est définie, il faut représenter chaque service et ses paramètres d'entrée/sortie avec la sémantique qui leur sont associée utilisant le formalisme fonctionnel. Formellement un service Web peut être représenté sous forme d'une fonction *fsi = (Fsi ; Isi ; Osi)*. *Fsi* représente l'ensemble de fonctions (opérations) du service. L'ensemble Isi des variables représente les paramètres d'entrée de service et *Osi* son ensemble de paramètres de sortie.

A partir de service profile dans la description OWL-S (de chaque service) on retire les paramètre d'entrées et sorties in/out de chaque opération car les caractéristiques des services les plus faciles à obtenir sont le nombre et le type des entrées et sorties. Pour chaque opération, si un tel paramètre d'entrée i doit être fourni pour avoir telle sortie o ceci signifie qu'il existe une dépendance entre *i* et *o*, et pour laquelle une fonction *f(i) = o* est définie dans notre système.

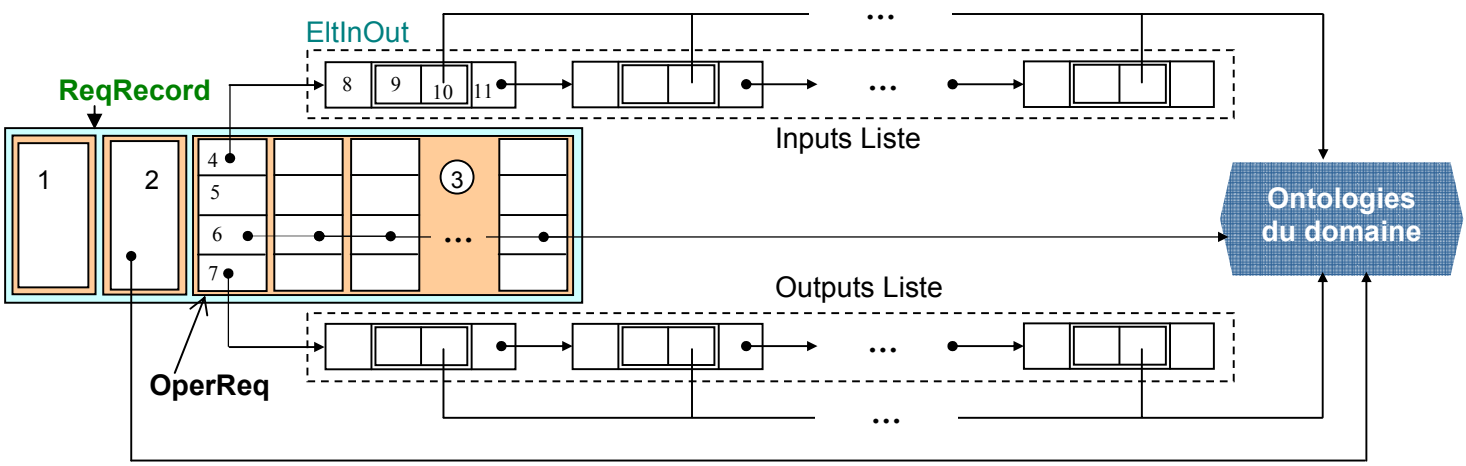
La sémantique fonctionnelle d'une opération est donnée par la sémantique de ses paramètres d'inputs/outputs, c'est à dire les types de données (simple ou complexe). La sélection de services Web sémantiques est un aspect crucial dans le processus de composition.

Une fonction composite *fsi o gsj* est définie entre *fsi et gsj*, si et seulement si il y a une similarité de type entre I_k *et* O_l c-à-d $I_k \subseteq O_l$ ($I_k \in Is_i$; $O_l \in Os_j$), où Is_i et Os_j sont les ensembles d'entrées et sorties de service *Si* et *Sj* respectivement.

5.6.1. Les structures de données sémantiques

Une opération (fonction) soit celle d'une requête ou bien d'un service, peut posséder une ou plusieurs variables d'Inputs/Outputs, pour cela on utilise la structure de donnée dynamique ***"Liste"***. Les ***"listes"*** spécifient la structure et la sémantique des informations qui sont communiquées comme paramètres de service dans la composition. Les informations qui sont communiquées à travers les messages doivent être conformes à ces structures. Comme chaque variable a son nom et annotation, donc chaque élément de la liste est représenté à l'aide d'une autre structure de donnée statique qui est de type ***"Record"***, donc on aura une ***liste d'enregistrements.***

Cette partie ***(figure 12, 13, 14)*** présente les structures de données sémantiques qui permettent de représenter chaque service et chaque requête avec leurs opérations.



1. ContextReq
2. Annot Req
3. OperReq
4. PtrIn
5. ContextOper
6. PtrContOper
7. PtrOut
8. Nom
9. TypeVar
10. PtrOntologie
11. Next

Figure 12 - Structure de donnée de la requête -

La structure de données illustrée dans la **Figure 12** représente la structuration d'une requête composée de plusieurs opérations. Cette structure est composée des éléments suivants:

ReqRecord: est un enregistrement composé de trois éléments:

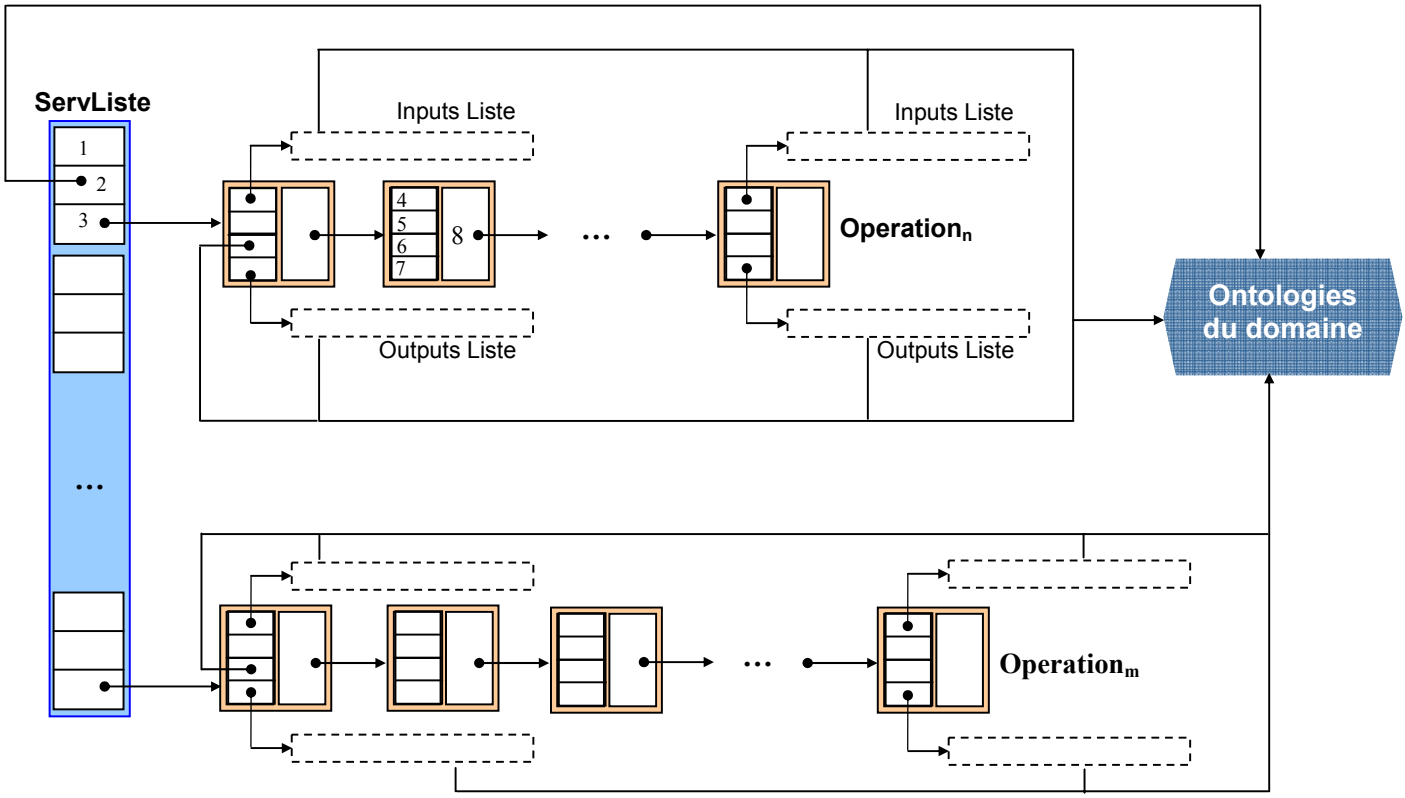
1. **ContextReq**: représente le nom de la requête.
2. **AnnotReq**: Pointeur vers une ontologie qui sert à l'annotation sémantique de la requête.
3. **OperReq:** est un tableau qui contient plusieurs enregistrements, chacun est composé de quatre champs, dont trois sont de type pointeur: ($Opération_i$ correspond à la $i^{ème}$ entrée du tableau).

- **PtrContOper:** Pointeur vers une ontologie qui sert à l'annotation sémantique des opérations de la requête.
- **ContextOper:** représente le nom de l'opération.
- **PtrIn, PtrOut:** représentent les têtes de listes des éléments d'Input et Output respectivement de l'opération. Ces listes contiennent des éléments définit comme suite :

EltInOut : c'est l'élément de base de chaque liste, il est de type enregistrement, composé

- ***Nom*:** Il est de type *String*, il représente le nom de chaque paramètre d'input/output.
- ***Annotation*:** c'est un enregistrement (*RECORD*) composé de deux champs qui contiennent des informations qui ajoute la sémantique aux paramètres d'inputs/outputs:
  - **TypeVar:** qui est de type *String*, il représente le vrai type des paramètres d'entrées/sorties.
  - **PtrOntologie:** représente un pointeur vers une ontologie du domaine ou un dictionnaire de mots qui peuvent contenir des mots synonymes à l'élément ***Nom***.

- ***Next*:** c'est un pointeur vers le prochain élément de la liste.



1. ContextServ
2. AnnotServ
3. PtrListOper

4,5,6,7. Définis comme **(4,5,6,7) Figure 12**
8. PtrNextOper

Figure 13 - Structure de donnée de la liste de services -

La structure de donnée illustrée dans la **Figure 13** représente la structuration d'une liste de services disponibles, chaque service peut être composé de plusieurs opérations. Cette structure est composée des éléments suivants:

1. **ServListe:** est un tableau dont les éléments sont de type Enregistrement :
   - **PtrListOper** : pointeur vers la liste des opérations pouvant être satisfaites par le même service.
   - **ContextServ** : représente le nom du service.
   - **Annot :** Pointeur vers une ontologie qui sert à l'annotation sémantique des services.

2. **Operation:** C'est l'élément de base de chaque liste des opérations, il est de type enregistrement (RECORD), composé de cinq champs, dont quatre sont de type pointeur :
   - **PtrIn :** représente la tête de listes des éléments d'Input de l'opération.
   - **PtrOut :** représente la tête de listes des éléments d'Output de l'opération.
   - **PtrNextOper :** pointeur vers la prochaine opération de la liste.
   - **PtrContOper :** Pointeur vers une ontologie qui sert à l'annotation sémantique des opérations du service.
   - **ContextOper :** représente le nom de l'opération.
3. **InputListe et OutputListe** : représente les listes des inputs et outputs de chaque opération. Les éléments de ces listes sont de même type que ceux définis auparavant pour les opérations de la requête. (*Elément **Figure 12***).

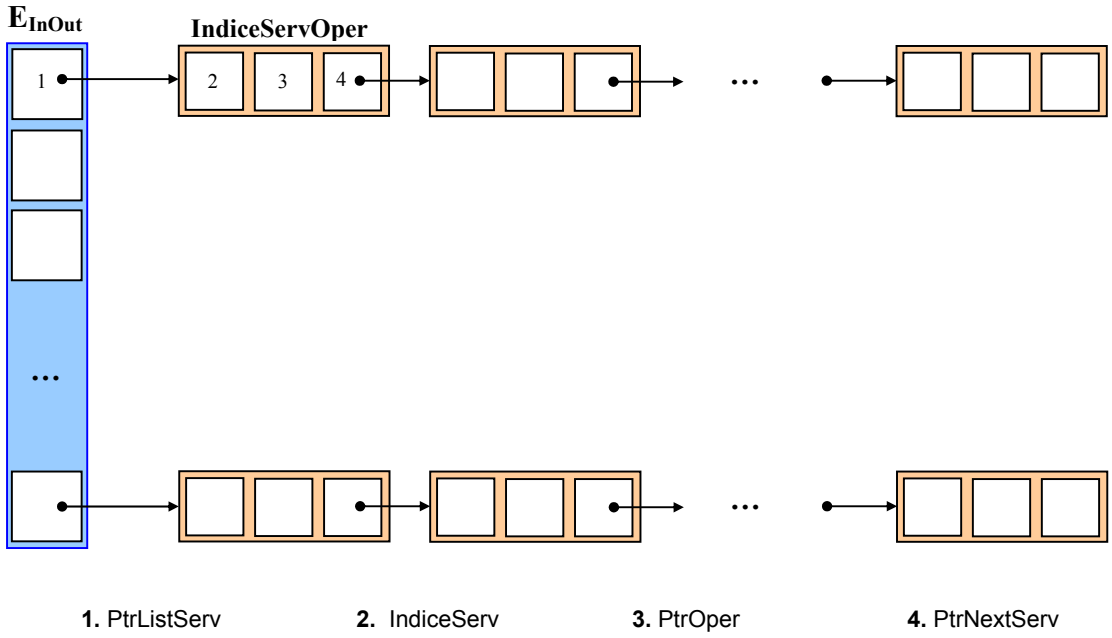


Figure 14 - Structure de liste de services satisfaisant chaque opération de la requête -

La structure de donnée illustrée dans la **figure 14** représente un tableau E_{InOut} qui contient des pointeurs vers les listes de services qui peuvent répondre à toutes les opérations de la requête.

Elle est composée des éléments suivants :

1. **E_{InOut} :** est un tableau dont les éléments sont de type Pointeur :

   - **PtrListServ :** pointeur vers la liste des services pouvant satisfaire l'opération « i » correspondant à la $i^{ème}$ entrée du tableau.

2. **IndiceServOper :** C'est l'élément de base de chaque liste de services, il est de type enregistrement (RECORD), composé de trois champs de type pointeur :

- **IndiceServ** : de type entier, représentant l'indice de l'entrée du tableau des services « **ServListe** » pouvant satisfaire l'opération demandée.
- **PtrOper** : pointeur vers l'opération du service qui match sémantiquement l'opération de la requête.
- **PtrNextServ :** pointeur vers le prochain élément de la liste de services.

Chapitre 6

Mise en œuvre

6.1. Algorithmes proposés

Les structures proposées dans notre travail ont une importance dans la description des capacités des paramètres de services, donc elles forment le fondement pour l'appariement sémantique ainsi que pour la composition.

Cette partie couvre les algorithmes nécessaires pour la mise en œuvre (l'implémentation) de la découverte, l'appariement et la composition de Services Web Sémantique. La première phase de notre système FS4WSC concerne la découverte de SWS: pour cela, nous avons proposé *l'Algorithme 1* de découverte des SWS qui peuvent potentiellement participer dans la composition. La deuxième phase de FS4WSC concerne la composition automatique où nous avons utilisé *l'Algorithme 2* de composition pour composer les services découverts dans la première phase s'ils sont composables. La troisième phase de FS4WSC qui n'est pas présente dans la liste de motivations correspond aux invocations automatiques de services en faisant appel aux services tout en fournissant les entrées et en obtenant les sorties.

Avant de construire nos algorithmes, on doit poser quelques hypothèses :

Hypothèse 1 : Le nombre de services Web disponibles est limité noté par Si. Soit n le nombre de fonctions correspondant à chaque service Web qui est représenté comme suit :

Pour chaque Si, on lui correspond une fonction = $Fi\ (I, O)$

- Fi : L'ensemble des fonctions, correspond aux opérations du service Web Si,
- I : L'ensemble des inputs de Fi, correspond aux inputs des opérations du service Web Si,
- O : L'ensemble des outputs de Fi, correspond aux outputs des opérations du service Web Si.

Hypothèse 2 : Toutes les opérations de chaque service Web sont connues ainsi l'ordre de leurs exécutions.

Hypothèse 3 : La requête du client est considérée comme une suite d'opérations.

Hypothèse 4 : Au plus un seul service peut exécuter une opération.

Corollaire de l'algorithme : L'algorithme retourne une réponse positive s'il existe une composition cohérente qui satisfait la requête (donné une exécution à la requête du client tout en respectant l'ordre de présentation de ces opérations), ainsi que l'ensemble des services Web impliqués dans cette composition, négative sinon.

6.1.1. Algorithme de découverte

6.1.1.1. Principe de fonctionnement de l'algorithme

Rappelons que notre problème consiste, dans un domaine particulier, à trouver tous les services et compositions de services qui permettent de répondre à une requête. Pour cela, nous nous appuyons sur les annotations associées aux fonctions et à leurs paramètres d'entrées/sorties. L'algorithme développé dans cette partie consiste à comparer les paramètres d'une demande du service avec les paramètres des services disponibles en deux étapes comme nous l'avons expliqué dans la section 5.3.2. L'originalité de l'approche que nous proposons consiste donc à utiliser une sémantique précise. Les services sont annotés avec les types de données et d'autres informations. Les types de données sont considérés comme des paramètres potentiels dans l'algorithme de découverte de services. Basant sur ces critères, des services pertinents assurant chaque opération de la requête seront sélectionnés afin d'être composés.

Le client présente la suite d'opération qu'il désire exécuter, l'algorithme parcourt l'ensemble des services Web qui peuvent exécuter l'opération en cours, si un service existe, l'algorithme met à jour la liste des services découvert, puis il passe à l'opération suivante jusqu'à ce qu'il épuise la suite d'opérations présentée.

Soit $Es = (S_1 ; S_2 ;\ldots; S_N)$ un ensemble de services (fonctions) et $S_Q = (fS_Q ; IS_Q ; OS_Q)$ la requête de l'utilisateur, où IS_Q l'ensemble d'entrées de la requête, OS_Q son ensemble de sorties et fS_Q l'ensemble de fonctions entre les paramètres *in/out* de la requête, autrement dit pour chaque opération de la requête on lui associe une fonction *f(ins)=outs*. Le problème, en terme de fonctions est comme suit:

Y'a-t-il un sous-ensemble de fonctions de l'ensemble ci-dessus ayant des inputs et des outputs qui **match** sémantiquement tous les inputs (IS_Q) et tous les outputs (OS_Q) de la requête S_Q respectivement? (*a*)

Alors, pour chaque fonction appartenant à fS_Q :

- Chercher l'ensemble des services vérifiant la contrainte (*a*) basant sur *l'Algorithme 1*.

En faisant l'appariement entre la liste des inputs (ReqInputList) et des outputs (ReqOutputList) de chaque opération de la requête avec la liste des inputs (ServInputList) et des outputs (ServOutputList) de chaque opération des services disponibles S_j paramètre par

paramètre jusqu'à l'épuisement de tous les paramètres de chaque opération de la requête par itération sur les structures de données sémantiques qui leurs sont associés, à l'aide de la méthode *MatchInputsOutputs (ReqInputList, ServInputList, ReqOutputList, ServOutputList)*. Elle retourne plusieurs listes de services pouvant satisfaire chacune des opérations de la requête qui seront stockées dans le tableau $\mathbf{E_{InOut}}$, dont chaque entrée i correspond à la même entrée i du tableau **OperInOut**. L'entrée i du tableau $\mathbf{E_{InOut}}$ contient un pointeur vers une liste de services qui peuvent satisfaire l'opération i du tableau **OperInOut**. Chaque service ainsi que l'opération$_k$ qui lui correspond et qui satisfait l'opération$_i$ de la requête sont identifiés par l'indice du service **IndiceServ** dans le tableau **ServListe** et le pointeur vers l'opération du même service **PtrOper.** Cette méthode retourne un boolean satisfait = **true** si toutes les entrées du tableau $\mathbf{E_{InOut}}$ ne sont pas vide, c-à-d il existe au moins un service qui peut répondre à chaque opération de la requête, satisfait = **false** sinon, qui va être transmis comme paramètre d'entrée à l'algorithme de découverte *FindSWS.*

6.1.1.2. Présentation de l'algorithme

Boolean MatchInputsOutputs (ReqInputList, ServInputList, ReqOutputList, ServOutputList) :

Entrée: La liste des paramètres in/out des opérations de la requête et des opérations de chaque service
Sortie: Booléen qui indique l'existence ou pas des services satisfaisants toutes les opérations de la requête.

Boolean Satisfait;

Begin
Satisfait := **true**;
for each oper$_i$ ∈ OperReq
E_{InOut} [i] := ∅;
for each S_j ∈ S
for each oper$_k$ ∈ OperServ
if (ReqInputList = ServInputList) and (ReqOutputList = ServOutputList) **then**
begin
Add IndiceService and OperServIndice to E_{InOut} ;
{E_{InOut} = E_{InOut} U {indiceS_j and indiceOs_j} ; ajouter S_j qui répond à I_i et O_i à la liste du tableau E_{InOut}[i].}
end;
end for
end for
if E_{InOut}[i] = ∅ **then**
Satisfait := **false;**
end if
end for

La requête ne peut être satisfaite {Une des entrées du tableau est vide}. Fin du parcours de la boucle des opérations c-à-d on ne vérifie pas l'opération suivante. {No service that match Input and Output of opération$_i$ found};

return satisfait **;**
End.

Après avoir retourner les listes de services correspondantes à chacune des opérations de la requête (les services satisfaisants chaque opération), On fait l'intersection entre chaque liste *(voir algorithme 1)*, on aura les cas suivants:

- un service en commun, ce service peut répondre à toutes les opérations donc à la requête intégrale (on peut avoir plusieurs services en commun mais on prend le premier trouvé).
- Si ce n'est pas le cas, on fait appel à l'algorithme 2 de composition ***SWSComposition***

Algorithme 1 :

FindSWS (Boolean a);

Entrée (s) : liste de services découverts qui satisfont chacune des opérations.
Sortie (s) : service simple/composite qui satisfait la requête.

```
Boolean b;
Begin
    b: = MatchInputsOutputs (ReqInputList, ServInputList, ReqOutputList, ServOutputList);
     if (b = true) then
            begin
              for each liste_i ∈ E_InOut do
                  if ∩ (liste_i) = S_j then
                      begin
                      Cette requête peut être satisfaite par un seul service S_j sans
                      composition;
                      end
                  else
                          SWSComposition (Array P);
               end for
             end if

 End.
```

6.1.2. Algorithme de composition

6.1.2.1. Principe de fonctionnement de l'algorithme

L'Algorithme 2 développé dans cette partie consiste à sélectionner une combinaison de services pouvant participer dans la composition à partir des listes de services retournés par l'Algorithme 1 en cherchant à fournir en sortie une fonction composite entre chaque deux services. Cette phase détermine donc les fonctions nécessaires à la composition de services et ensuite propose un ordre d'exécution de ces fonctions (tâches). Cet ordre est transmis comme paramètre à la phase suivante de FS4WSC qui est la phase d'Exécution de services, afin d'obtenir les valeurs pour les services candidats.

Avant de faire la composition de services, l'algorithme vérifie si les opérations de la requête s'exécutent séquentiellement et/ou parallèlement, en faisant la comparaison entre la liste des Outputs de la i^{ème} opération ($Oper_i$OutputList) et la liste des Inputs de la i^{ème} +1 opération ($Oper_{i+1}$InputList).

- Si les opérations s'exécutent en séquence, on fait la composition séquentielle de services correspondant à chacune de ces opérations.

  $(S_{j/j\in\{1..n\}} \in E_{inOut}[i+1])$ **o** $(S_{j/j\in\{1..m\}} \in E_{inOut}[i])$
- Si les opérations s'exécutent en parallèle, on fait la composition parallèle de services correspondant à chacune de ces opérations.(dans ce cas l'ordre n'est important)

  $(S_{j/j\in\{1..n\}} \in E_{inOut}[i])$ **//** $(S_{j/j\in\{1..m\}} \in E_{inOut}[i+1])$

6.1.2.2. Présentation de l'algorithme

Algorithme 2 :

SWSComposition (Array OperInOut):

Entrée: liste des opérations de la requête ;
Sortie: service(s) composé(s) qui satisfait la requête ;

begin
  for each $Oper_i \in$ OperInOut **do**
    if $Oper_i$OutputList = $Oper_{i+1}$InputList **then**
      SéqCompo $(S_{j/j\in\{1..n\}} \in E_{inOut}[i+1]$, $S_{j/j\in\{1..m\}} \in E_{inOut}[i])$
    else
      ParCompo $(S_{j/j\in\{1..n\}} \in E_{inOut}[i]$, $S_{j/j\in\{1..m\}} \in E_{inOut}[i+1])$;
  end for
end.

Remarque : Notre algorithme peut retourner en résultat toutes les compositions possibles, on peut sélectionner la meilleure en utilisant une autre méthode *"findBestComposition"* par exemple qui prend en entrée les compositions et donne en sortie la composition qui satisfait au mieux la demande du client. Dans notre travail, comme on l'a déjà mentionné, on choisit la première composition.

6.2. Etude de cas : l'agence de voyages

6.2.1. Choix du langage fonctionnel CAML

Le but principal du choix du langage typé CAML est d'apporter un moyen simple et naturel pour la définition de nouveaux types de données en privilégiant l'aspect sémantique des services Web (sémantique associée aux noms des services Web, aux opérations, aux entrées/sorties). L'utilisateur exprime, via ce langage, sa perception sémantique des différents objets qu'il souhaite implémenter et non pas les détails techniques de mis en œuvre.

Notre choix s'est porté sur ce langage car CAML est un langage : fortement typé, à typage polymorphe, à **inférence de type**, efficace, sûr, ramasse miettes (récupération automatique de la mémoire) …etc **[BEA 04], [HIC 06]**. L'une des grandes forces de ce langage réside dans la puissance des structures de données définissables et la simplicité de leur manipulation.

Les différents types que le langage CAML fournit et qu'on va utiliser pour modéliser notre exemple sont :

- **Les types de bases :** *int, float, char, string, bool,* et *unit.* ***unit*** dénote la valeur notée "**()**" qui indique un résultat sans importance (équivalent à *void* en ***C/C++***).

- **Un type fonctionnel :** défini par le constructeur "→" associatif à droite. Soit **t1** et **t2** deux types quelconques, **t1 → t2** est un type fonctionnel. Une valeur de type fonctionnel est appelée *fermeture.*

- **Un type produit :** défini par le constructeur de type "*" non associatif. Ce type est construit à partir du produit cartésien de n types, $n \geq 1$. On en distingue particulièrement :

  - ***Les n-uplets :*** Soit n ($n > 1$) types quelconques : ***t1, t2, …, tn.*** Le type ***t1 * t2 *… * tn*** est le type des n-uplets dont la $i^{ème}$ composante ($i \in [1,n]$) est de type ***ti***. Cas particulier : ***n=2***, le type ainsi défini est une ***paire***.
  - ***Les records ou enregistrements :*** défini à l'aide de champs nommés et typés explicitement, et mis entre accolades {}.
  - ***Les vecteurs vect :*** c'est un ensemble fini d'éléments de même type.
  - ***Les listes :*** c'est un ensemble a priori infini d'éléments de même type.

- **Un constructeur de type** : c'est un opérateur défini sur les types. Il peut être soit prédéfini comme ***list***, →, …, soit introduit par l'utilisateur. A chaque constructeur de types correspond un ou plusieurs constructeurs de valeurs qui appartiennent au type construit. Par exemple "**,**" est le constructeur de valeur de type n-uplets.

Afin d'implémenter les annotations sémantiques associées aux services Web de notre exemple, on va intégrer de nouveaux types de données sémantiques dans le langage CAML après avoir décrit nos services, (*voir section 6.2.2.*) on se basant sur les types définis ci-dessus.

Nous avons utilisé le langage CAML pour implémenter les algorithmes proposés décrit dans la *section 6.1* et les appliquer au domaine de réservation de voyages en ligne. Ce simple exemple de composition permet d'illustrer concrètement notre problématique. Cependant, il est important de noter que les perspectives d'application de notre travail sont beaucoup plus vastes et concernent tous les domaines dans lesquels la composition de services Web peut être envisagée.

6.2.2. Exemple d'application

Une agence de voyages *"e-TravelAgency"* fournit typiquement les services pour : consultation, réservation, paiement et annulation de billets d'avion, de chambres d'hôtel et de locations de voiture. Afin de fournir ces services à ses clients, l'agence de voyages doit établir des liens avec d'autres entreprises : compagnies aériennes, compagnies de location de voitures et réseaux hôteliers. Une institution financière (une banque) est également nécessaire pour faciliter les transactions financières entre les clients et l'agence de voyages, ou entre l'agence de voyages et les autres partenaires.

Le principe général du système peut se résumer ainsi :

• La compagnie aérienne, la compagnie de location de voitures et le réseau hôtelier doivent fournir des services permettant à une agence de voyages de consulter, réserver, payer et d'annuler les vols, les voitures et les chambres disponibles respectivement.

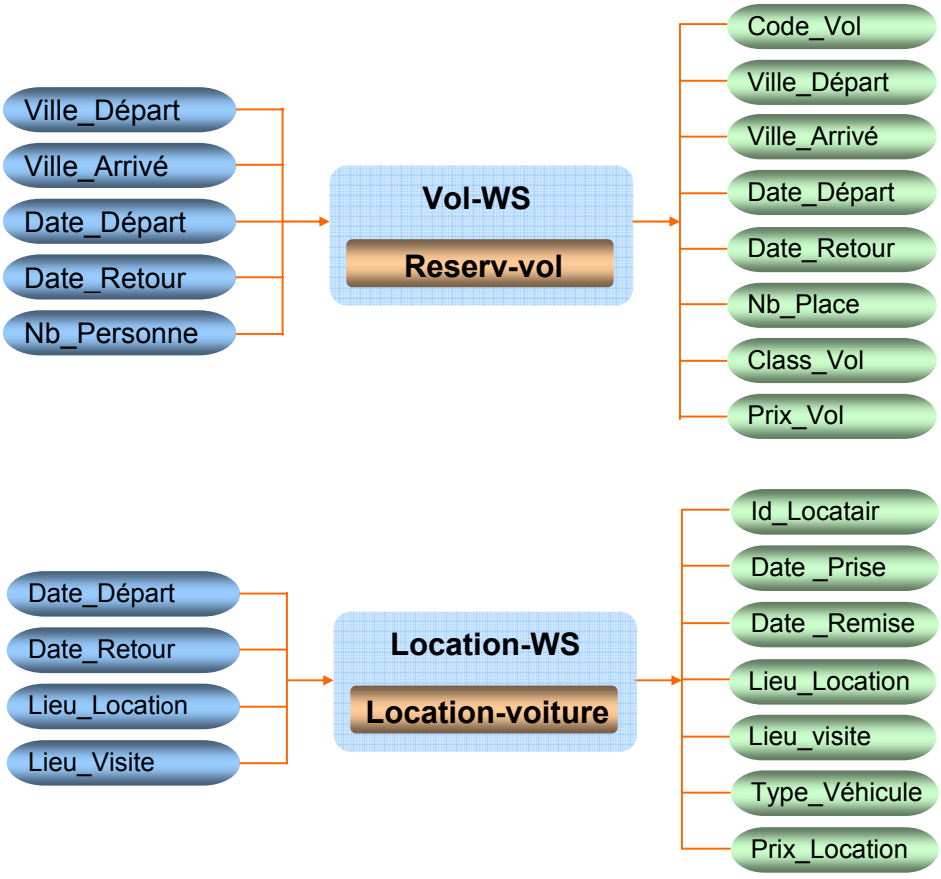
• La banque doit fournir un service permettant à une entreprise d'effectuer une transaction financière à partir de données de payement (identités du débiteur et du créancier).

A cet effet sept services web peuvent être proposés :

- ***Vol-WS*** : retourne les informations concernant les vols programmés selon la date sélectionnée et la ville de départ et d'arrivée.
- ***Location-WS***: retourne la disponibilité et les informations d'un modèle de voiture au près des agences de location de voiture selon la date sélectionnée et de la ville sélectionné.

- ***Hôtel-WS***: permet d'afficher touts les noms, l'adresse, le nombre d'étoiles, le prix de la chambre…etc. des hôtels de la ville sélectionnés ainsi que la disponibilité de chambres vide selon la date précisée et le nombre de places.
- ***Activité-WS***: retourne les activités disponibles d'une ville donnée.
- ***Bank-WS*** : permet au client de payer les différentes réservations qu'il sera amené à faire, puis additionne les prix générés par les services participant à la composition.
- ***Vol+Car-WS*** : offre les fonctionnalités du service *Reserv-Vol-WS* et du service *Rent-a-Car-WS*.
- ***Hôtel+Activité-WS*** : offre les fonctionnalités du service *Reserv-Hôtel-WS* et du service *Activité-WS*.

La figure ci-après (***figure 15***) montre les services fournit par l'agence de notre exemple ainsi que les inputs/outputs correspond à chaque services.



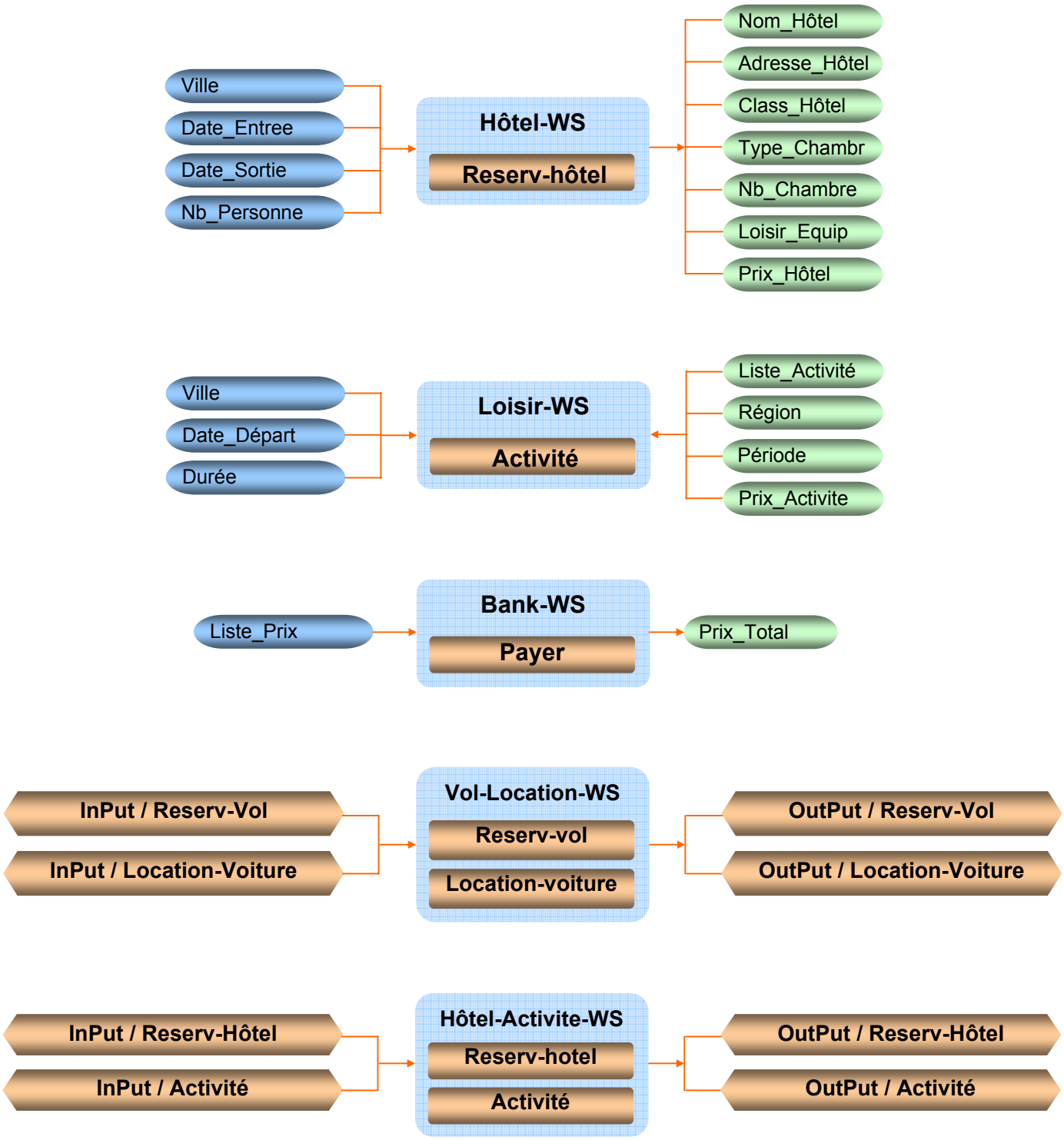


Figure 15 - *Les services Web offert par l'agence "e-TravelAgency"*

Un enseignant chercheur, habite à "*Alger*", doit se rendre à "*Constantine*" *Samedi* pour assister à un colloque. Il décide d'organiser son voyage par Internet en faisant appel à différents services Web de l'agence "*e-TravelAgency*". Sa requête comprend la réservation de billet d'avion et la location d'un véhicule pour la durée du séjour, ainsi que le payement de ces derniers.

La ***figure 16*** montre la suite d'opérations ainsi que la liste d'Inputs/Outputs de la requête du client.

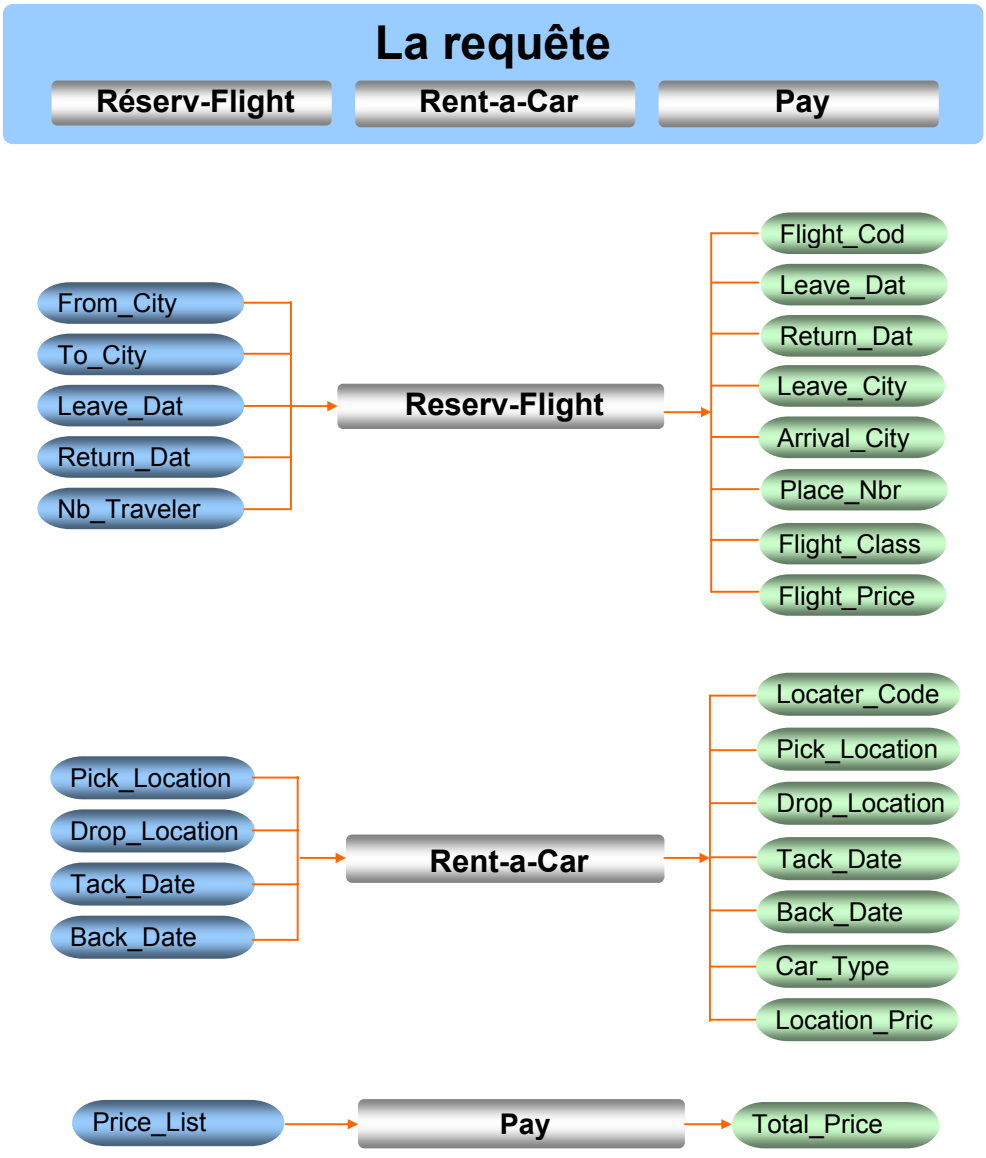


Figure 16 - La liste des opérations de la requête

Afin de satisfaire la demande du client, on propose la composition des services web montrée dans la **figure 17**.

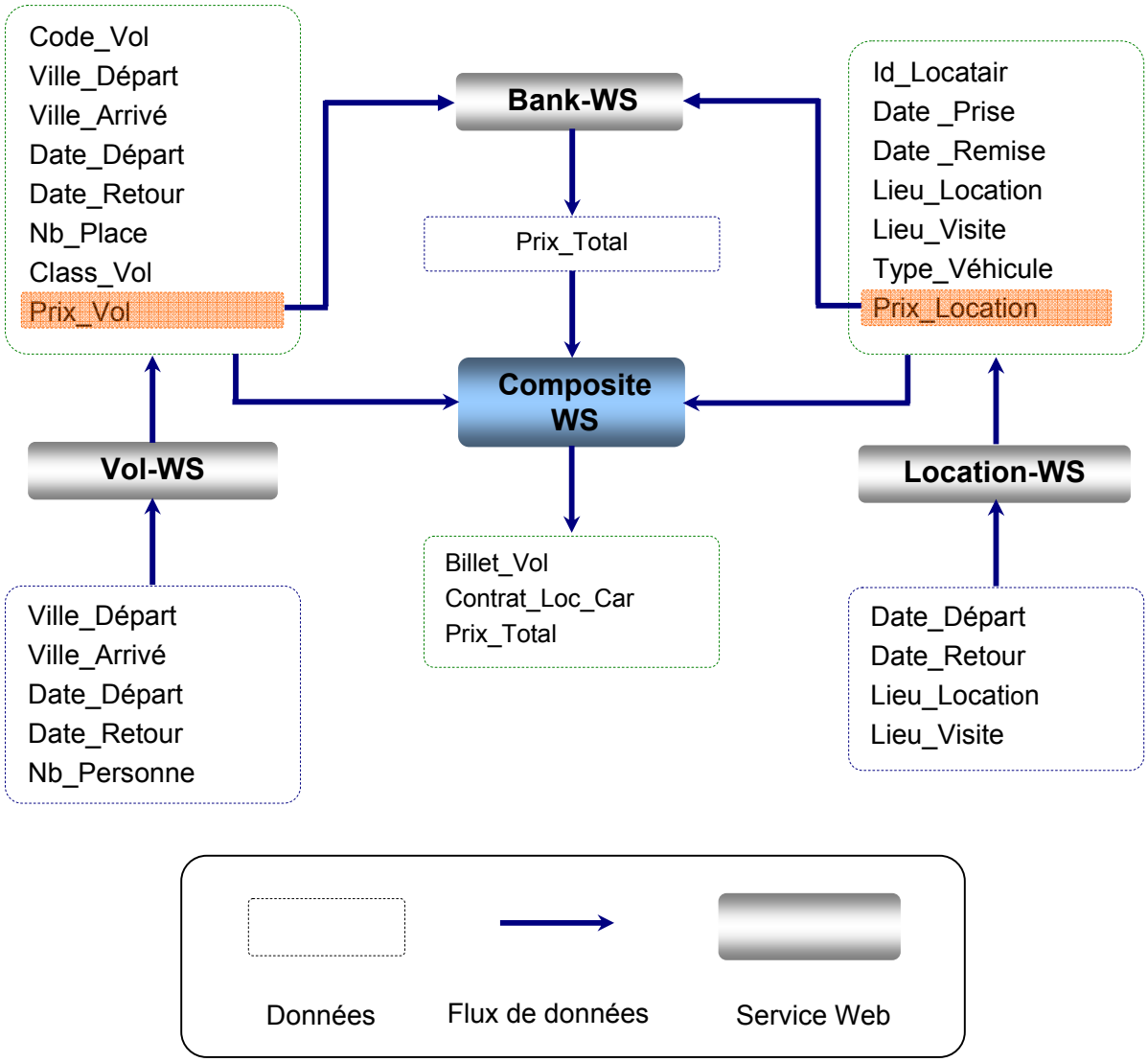


figure 17- *Composition de Services Web : Vol-WS, Location-WS et Bank-WS*

6.2.3. Implantation en CAML

6.2.3.1. Contraintes sémantiques

- La représentation de date diffère d'un service à un autre. Le service Web de *"Location de voiture"* utilise un format (jj.mm.aaaa), alors que le service Web "Rent-a-Car" adopte une notation anglophone (mm.jj.aaaa). Cet exemple montre qu'une description des données établie par le langage WSDL ne remplit pas les exigences de l'échange sémantique. Afin de prendre en considération ces deux formats un type *date* est défini à l'aide des constructeurs *App* et *Fun* (voir *section 6.2.3.2*).

- Le service "*Bank-SW*" possède une fonction *Payer* qui prend en entrée les paramètres "prix" correspondant aux différentes réservations : vol, hôtel, activité, location de voiture, et retourne en sortie le prix total des différentes réservations. Ce service utilise la devise "*Euro*" du prix total, tandis que le prix utilisé dans les services "*Reserv-Vol-SW*" et "*Rent-a-car-SW*" est le "*DA*". Malgré une compatibilité des types de données (type *"float"*) et des concepts sémantiques utilisés (concept *"prix"* et concept *"price"*), les données doivent être interprétées différemment, à cause de l'incompatibilité des unités de mesure. Pour éliminer cette incohérence, il faut convertir le prix des services à additionner dans la même devise du service "*Bank-SW* ". A cet effet, notre système offre des fonctions de conversion du ***DA*** vers l'***Euro***, du ***Yen*** vers l'***Euro*** et de l'***USD*** vers l'***Euro*** …etc. car la banque utilise seulement la monnaie qui est mesurée en *Euro* dans notre exemple. Ces fonctions sont définies comme suite : "*DA_To_Euro*", "*USD_To_Euro*" etc comme montré dans la *section 6.2.3.2*. Pour faire cette conversion il suffit seulement de définir des formules qui permettent de multiplier par un facteur multiplicateur qui est un nombre utilisé pour la mise à l'échelle, selon la monnaie de chaque pays qui est définie dans l'ontologie.

Cependant, l'intérêt de la description sémantique est de spécifier les paramètres du service en leur donnant une signification.

6.2.3.2. Implantation de notre exemple

```
# type monnaie = {nom : string; valeur : float};;
# let prix_vol = {nom= "euro"; valeur= 999.5};;

# type  date = | App of jour_semaine * int * nom_mois * int
               | Num of jour_semaine * int * int * int ;;
               | Fun of nom_mois * jour_semaine * int;;
# let dA_To_EURO x = x *. 0.01;;
val dA_To_EURO : float -> float = <fun>
```

Les paramètres d'Inputs/Outputs des opérations des services et de la requête sont déclarés comme suit :

```
# type eltInOut = {nom:string ; vrai_typ:string ; annot:string list};;
type eltInOut = { nom : string; vrai_typ : string; annot : string list}
```

*eltInOut est un "record " contenant les éléments de chaque Inputs liste /Outputs liste (**figure3**).*

1/ Remplissage des différents champs de chaque élément Input/Output des opérations

1.1. Les enregistrements correspondant aux opérations de la requête

Indication: **operInij** : L'input n° j de l'opération i. **operOutij** : L'output n° j de l'opération i.

a. Reserv_Flight

```
# let operIn11 = {nom = "From_city" ; vrai_typ = "String" ; annot = ["Ville_depar " ; "Departure_city"]};;
val operIn11 : eltInOut =
  {nom = "From_city" ; vrai_typ = "String" ;  annot = ["Ville_depar "; " Departure_city "]}

# let operIn12 = {nom = "To_city" ; vrai_typ = "String" ; annot = ["Ville_arrive "; "Arrival_city"]};;
# let operIn13 = {nom = "Leav_dat" ; vrai_typ = "date" ; annot = ["Date_depar"]};;
# let operIn14 = {nom = "Return_dat" ; vrai_typ = "date" ; annot = ["Date_retour"]};;
# let operIn15 = {nom = "Nb_traveler" ; vrai_typ = "int" ; annot = ["Nb_personne"]};;
# let  operOut11 = {nom = "Flight_code" ; vrai_typ = "string"; annot = ["Code_vol"]};;
val operOut11 : eltInOut =
  {nom = "Flight_code"; vrai_typ = "string"; annot = ["Code_vol"]}

# let operOut12 = {nom = "Leav_date" ; vrai_typ = "date"; annot = [ "Date_depar"]};;
# let operOut13 = {nom = "Return_dat" ; vrai_typ = "date" ; annot = ["Date_retour"]};;
# let operOut14 = {nom = "Departure_city" ; vrai_typ = "String" ; annot = ["Ville_depar "; "From_city"]};;
# let operOut15 = {nom = "Arrival_city" ; vrai_typ = "String" ; annot = ["Ville_arrive";"To_city"]};;
# let operOut16 = {nom = "Place_nbr" ; vrai_typ = "int" ; annot = ["Nb_place"]};;
# let operOut17 = {nom = "Flight_class" ; vrai_typ = "String" ; annot = ["Class_vol"]};;
# let operOut18 = {nom = "Flight_price" ; vrai_typ = "monnaie" ; annot = ["Prix_vol"]};;
```

b. Rent_a_Car

```
# let operIn21 = {nom = "Pick_location" ; vrai_typ = "string" ; annot = ["Lieu_location"; "Departure_city"]};;
# let operIn22 = {nom = "Drop_location" ; vrai_typ = "string" ; annot = ["Lieu_visite"; "Arrival_city"]};;
# let operIn23 = {nom = "Tack_date" ; vrai_typ = "date" ; annot = ["Date_ prise"; "Leav_date"]};;
# let operIn24 = {nom = "Back_date" ; vrai_typ = "date" ; annot = ["Date_ remise" ;"Return_date"]};;

# let operOut21 = {nom = "Locatair_cod" ; vrai_typ = "string" ; annot = ["Id_locatair"]};;
# let operOut22 = {nom = "Pick_location" ; vrai_typ = "string" ;annot = [" Lieu_location";
"Departure_city"]};;
# let operOut23 = {nom = "Drop_location" ; vrai_typ = "string" ; annot = ["Lieu_visite"; "Arrival_city"]};;
# let operOut24 = {nom = "Tack_date" ; vrai_typ = "date" ; annot = ["Date_prise" ;"Leav_date"]};;
```

```
# let operOut25 = {nom = "Back_date" ; vrai_typ = "date" ; annot = ["Date_remise"; "Return_date"]};;
# let operOut26 = {nom = "Car_typ" ; vrai_typ = "string" ; annot = ["type_vehicule“]};;
# let operOut27 = {nom = "Location_price" ; vrai_typ = "monnaie" ; annot = ["Prix_location"]};;
```

c. Pay

```
# let operIn31 = {nom = "Flight_price" ; vrai_typ = "monnaie" ; annot= [“Prix_vol@Prix_transport"]};;
# let operIn32 = {nom = "Rent_price" ; vrai_typ = "monnaie" ; annot= [“Prix_location"]};;
# let operOut31 = {nom = "Total_price" ; vrai_typ = "monnaie" ; annot = ["Prix_total"]};;
```

1.2. Les enregistrements correspondant aux opérations des services

a. Reserv- vol

```
# let operIn41 = {nom = "Ville_depar" ; vrai_typ = "string" ; annot = ["From_city" ;"Departure_city"]};;
# let operIn42 = {nom = "Ville_arrive" ; vrai_typ = "string" ; annot = ["To_city"; "Arrival_city"]};;
# let operIn43 = {nom = "Date_depar" ; vrai_typ = "date" ; annot = ["Leav_dat"]};;
# let operIn44 = {nom = "Date_retour" ; vrai_typ = "date" ; annot = ["Return_dat"]};;
# let operIn45 = {nom = "Nb_personne" ; vrai_typ = "int" ; annot = ["Nb_traveler"]};;

# let operOut41 = {nom = "Code_vol" ; vrai_typ = "string"; annot = ["Flight_code"]};;
# let operOut42 = {nom = "Date_depar" ; vrai_typ = "date"; annot = ["Leav_date"]};;
# let operOut43 = {nom = "Date_retour" ; vrai_typ = "date" ; annot = ["Return_date"]};;
# let operOut44 = {nom = "Ville_depar " ; vrai_typ = "String" ; annot = ["Departure_city" ; "From_city"]};;
# let operOut45 = {nom = "Ville_arrive" ; vrai_typ = "String" ; annot = ["Arrival_city" ;"To_city"]};;
# let operOut46 = {nom = "Nb_place" ; vrai_typ = "int" ; annot = ["Place_nbr"]};;
# let operOut47 = {nom = "Class_vol" ; vrai_typ = "String" ; annot = ["Flight_class"]};;
# let operOut48 = {nom = "Prix_vol" ; vrai_typ = "monnaie" ; annot = ["Flight_price"]};;
```

b. Location- voiture

```
# let operIn51 = {nom = "Lieu_location" ; vrai_typ = "string"; annot = ["Pick_location"; "Departure_city"]};;
# let operIn52 = {nom = "Lieu_visite" ; vrai_typ = "string" ; annot = [" Drop_location" ; "Arrival_city"]};;
# let operIn53 = {nom = "Date_prise" ; vrai_typ = "date" ; annot = ["Tack_date"; "Leav_date"]};;
# let operIn54 = {nom = "Date_remise" ; vrai_typ = "date" ; annot = ["Back_date" ;"Return_date"]};;

# let operOut51 = {nom = "Id_locatair" ; vrai_typ = "string" ; annot = ["Locatair_cod"]};;
# let operOut52 = {nom = "Lieu_location" ; vrai_typ = "string"; annot =["Pick_location";"Departure_city"]};;
# let operOut53 = {nom = "Lieu_visite" ; vrai_typ = "string" ; annot = ["Drop_location"; "Arrival_city"]};;
# let operOut54 = {nom = "Date_prise" ; vrai_typ = "date" ; annot = ["Tack_date"; "Leav_date"]};;
# let operOut55 = {nom = "Date_remise" ; vrai_typ = "date" ; annot = ["Back_date" ; "Return_date"]};;
# let operOut56 = {nom = "type_vehicule" ; vrai_typ = "string" ; annot = ["Car_typ"]};;
# let operOut57 = {nom = "Prix_location" ; vrai_typ = "monnaie" ; annot = ["Location_price"]};;
```

c. Payer

```
# let operIn61 = {nom = "Prix_vol" ; vrai_typ = "monnaie" ; annot= [“Flight_price"; "Prix_transport"]};;
# let operIn62 = {nom = "Prix_location" ; vrai_typ = "monnaie" ; annot= [“Rent_price"]};;
# let operOut61 = {nom = "Prix_total" ; vrai_typ = "monnaie" ; annot = ["Total_price"]};;
```

2/ Construction des Inputs liste et Outputs liste associées à chacune des opérations

2.1. Les Inputs liste/ Outputs liste correspond aux opérations de la requête

Indication: rOperlistIni : La liste d'inputs de l'opération n°i de la requête.
sOperlistOutj : La liste d'outputs de l'opération n°j du service.

a. Reserv- Flight

```
# let rOperlistIn1 = [operIn11; operIn12; operIn13; operIn14; operIn15];;
val rOperlistIn1 : eltInOut list =
  [{nom = "From_city"; vrai_typ = "String"; annot = ["Ville_depar"; "Departure_city"]};
   {nom = "To_city"; vrai_typ = "String"; annot = ["Ville_arrive"; "Arrival_city"]};
   {nom = "Leav_dat"; vrai_typ = "date"; annot = ["Date_depar"]};
   {nom = "Return_dat"; vrai_typ = "date"; annot = ["Date_retour"]};
   {nom = "Nb_traveler"; vrai_typ = "int"; annot = ["Nb_personne"]}];;
```

"*rOperlistIn1*" ***est une liste d'enregistrements de type*** **"eltInOut list".**

```
# let rOperlistOut1 = [operOut11; operOut12; operOut13; operOut14; operOut15; operOut16; operOut17;
operOut18];;
```

b. Rent- a-Car

```
# let rOperlistIn2 = [operIn21; operIn22; operIn23; operIn24];;
# let rOperlistOut2 = [operOut21; operOut22; operOut23; operOut24; operOut25; operOut26; operOut27];;
```

c. Pay

```
# let rOperlistIn3 = [operIn31; operIn32];;
# let rOperlistOut3 = [operOut31] ;;
```

2.2. Les Inputs liste/ Outputs liste correspond aux opérations des services

a. Reserv- vol

```
# let sOperlistIn1 = [operIn41; operIn42; operIn43; operIn44; operIn45];;
# let sOperlistOut1 = [operOut41; operOut42; operOut43; operOut44; operOut45; operOut46; operOut47;
operOut48];;
```

b. Location- voiture

```
# let sOperlistIn2 = [operIn51; operIn52; operIn53; operIn54];;
# let sOperlistOut2 = [operOut51; operOut52; operOut53; operOut54; operOut55; operOut56; operOut57];;
```

c. Payer

```
# let sOperlistIn3 = [operIn61; operIn62];;
# let sOperlistOut3 = [operOut61] ;;
```

3/ Construction du tableau "OperReq" *qui contient un ensemble d'enregistrements (opérations de la requête) comme montrés dans la figure 16 dont chaque enregistrement est constitué de quatre éléments :* ***ContextOper, AnnotOper, ListInputs et ListOutputs***

3.1. Construction des enregistrements (opération$_i$) du tableau "OperReq"

```
# type eltOperReq = {context_Oper: string; annot_Oper : string; rOperlistIn: eltInOut list; rOperlistOut: eltInOut list};;
type eltOperReq = {
        context_Oper : string;
        annot_Oper : string;
        rOperlistIn : eltInOut list;
        rOperlistOut : eltInOut list;}
```

"eltOperReq est un enregistrement contenant les informations de chaque opération de la requête".

- **Remplissage des différents champs de chaque opération (record)**

a. Reserv-flight

```
# let operReq1 = { context_Oper = "Reserv-flight"; annot_Oper = "Reserv-vol"; rOperlistIn = rOperlistIn1;
  rOperlistOut = rOperlistOut1};;
  val operReq1 : eltOperReq =
        {context_Oper = "Reserv_flight"; annot_Oper = "Reserv_vol";
         rOperlistIn = [{nom = "From_city"; vrai_typ = "String"; annot = ["Ville_depar" ; "Departure_city"]};
                        {nom = "To_city"; vrai_typ = "String"; annot = ["Ville_arrive" ; "Arrival_city"]};
                        {nom = "Leav_dat"; vrai_typ = "date"; annot = ["Date_depar"]};
                        {nom = "Return_dat"; vrai_typ = "date"; annot = ["Date_retour"]};
                        {nom = "Nb_traveler"; vrai_typ = "int"; annot = ["Nb_personne"]}];
         rOperlistOut = [{nom = "Flight_code"; vrai_typ = "string"; annot = ["Code_vol"]};
                         {nom = "Leav_date"; vrai_typ = "date"; annot = ["Date_depar"]};
                         {nom = "Return_dat"; vrai_typ = "date"; annot = ["Date_retour"]};
                         {nom = "Departure_city"; vrai_typ = "String"; annot = ["Ville_depar"; "From_city"]};
                         {nom = "Arrival_city"; vrai_typ = "String"; annot = ["Ville_arrive"; "To_city"]};
                         {nom = "Place_nbr"; vrai_typ = "int"; annot = ["Nb_place"]};
                         {nom = "Flight_class"; vrai_typ = "String"; annot = ["Class_vol"]};
                         {nom = "Flight_price"; vrai_typ = "monnaie"; annot = ["Prix_vol"]}]}
```

operReq1 correspond à la première operation (ou enregistrement) "Reserv-flight" de la requête.

b. Rent-a-car

```
# let operReq2 = { context_Oper = "Rent-a-car"; annot_Oper = "Location-voiture";
  rOperlistIn = rOperlistIn2 ; rOperlistOut = rOperlistOut2};;
```

c.Pay

```
# let operReq3 = { context_Oper = "Payment"; annot_Oper = "Versement@Payement";
  rOperlistIn = rOperlistIn3;  rOperlistOut = rOperlistOut3};;
```

3.2. Construction du tableau "OperReq"

```
# let operReqTab = [|operReq1; operReq2; operReq3|];;
val operReqTab : eltOperReq array =
 [|{context_Oper = "Reserv-flight"; annot_Oper = "Reserv-vol";
            rOperlistIn = [{nom = "From_city"; vrai_typ = "String"; annot = ["Ville_depar" ; "Departure_city"]};
                           {nom = "To_city"; vrai_typ = "String"; annot = ["Ville_arrive" ; "Arrival_city"]};
                           {nom = "Leav_dat"; vrai_typ = "date"; annot = ["Date_depar"]};
                           {nom = "Return_dat"; vrai_typ = "date"; annot = ["Date_retour"]};
                           {nom = "Nb_traveler"; vrai_typ = "int"; annot = ["Nb_personne"]}];
       rOperlistOut = [{nom = "Flight_code"; vrai_typ = "string"; annot = ["Code_vol"]};
                       {nom = "Leav_date"; vrai_typ = "date"; annot = ["Date_depar"]};
                       {nom = "Return_dat"; vrai_typ = "date"; annot = ["Date_retour"]};
                       {nom = "Departure_city"; vrai_typ = "String"; annot = ["Ville_depar"; "From_city"]};
                       {nom = "Arrival_city"; vrai_typ = "String"; annot = ["Ville_arrive"; "To_city"]};
                       {nom = "Place_nbr"; vrai_typ = "int"; annot = ["Nb_place"]};
                       {nom = "Flight_class"; vrai_typ = "String"; annot = ["Class_vol"]};
                  {nom = "Flight_price"; vrai_typ = "monnaie"; annot = ["Prix_vol"]}]};
  {context_Oper = "Rent-a-car"; annot_Oper = "Location-voiture";
   rOperlistIn = [{nom = "Pick_location"; vrai_typ = "string"; annot = ["Lieu_location";"Departure_city"]};
                  {nom = "Drop_location"; vrai_typ = "string"; annot = ["Lieu_visite";"Arrival_city"]};
                  {nom = "Tack_date"; vrai_typ = "date"; annot = ["Date_ prise"; "Leav_date"]};
                  {nom = "Back_date"; vrai_typ = "date"; annot = [“Date_ remise” ;”Return_date”]}];
   rOperlistOut = [{nom = “Locatair_cod”; vrai_typ = “string”; annot = ["Id_locatair"]};
                   {nom = "Pick_location"; vrai_typ = "string"; annot = [" Lieu_location";"Departure_city"]};
                   {nom = "Drop_location"; vrai_typ = "string"; annot = ["Lieu_visite";"Arrival_city"]};
                   {nom = "Tack_date"; vrai_typ = "date"; annot = ["Date_prise";"Leav_date"]};
                   {nom = "Back_date"; vrai_typ = "date"; annot = ["Date_remise";"Return_date"]};
                   {nom = "Car_typ"; vrai_typ = "string"; annot = ["type_vehicule"]};
                   {nom = "Location_price"; vrai_typ = "monnaie"; annot = ["Prix_location"]}]};
    {context_Oper = "Payment"; annot_Oper = "Versement@Payement";
     rOperlistIn = [{nom = "Flight_price" ; vrai_typ = "monnaie" ; annot= [“Prix_vol"; "Prix_transport"]};
                    {nom = "Rent_price" ; vrai_typ = "monnaie" ; annot= [“Prix_location"]}];
     rOperlistOut = [{nom = "Total_price" ; vrai_typ = "monnaie" ; annot = ["Prix_total"]}]}|]
```

4. **Construction de l'enregistrement "ReqRecord" qui est composé de trois éléments contenant les informations de la requête dont : ContextReq, AnnotReq et OperReq**

type reqRecord = {context_Req: string; annot_Req: string; tabOperReq : eltOperReq array };;
type reqRecord = { context_Req : string ; annot_Req : string; tabOperReq : eltOperReq array}

```
# let request1 = {context_Req = "Voyage"; annot_Req = "Travel@Vol@Location@Rent@Flight
@Versement@Déboursement@ Frais_Voyage"; tabOperReq = operReqTab};;
val request1 : reqRecord =
 {context_Req = "Voyage"; annot_Req = "Travel@Vol@Location@Rent@Flight @Versement@Déboursement@
Frais_Voyage";
  tabOperReq =
[|{context_Oper = "Reserv-flight"; annot_Oper = "Reserv-vol";
            rOperlistIn = [{nom = "From_city"; vrai_typ = "String"; annot = ["Ville_depar" ; "Departure_city"]};
                           {nom = "To_city"; vrai_typ = "String"; annot = ["Ville_arrive" ; "Arrival_city"]};
                           {nom = "Leav_dat"; vrai_typ = "date"; annot = ["Date_depar"]};
                           {nom = "Return_dat"; vrai_typ = "date"; annot = ["Date_retour"]};
                           {nom = "Nb_traveler"; vrai_typ = "int"; annot = ["Nb_personne"]}];
        rOperlistOut = [{nom = "Flight_code"; vrai_typ = "string"; annot = ["Code_vol"]};
                        {nom = "Leav_date"; vrai_typ = "date"; annot = ["Date_depar"]};
                        {nom = "Return_dat"; vrai_typ = "date"; annot = ["Date_retour"]};
                        {nom = "Departure_city"; vrai_typ = "String"; annot = ["Ville_depar"; "From_city"]};
                        {nom = "Arrival_city"; vrai_typ = "String"; annot = ["Ville_arrive"; "To_city"]};
                        {nom = "Place_nbr"; vrai_typ = "int"; annot = ["Nb_place"]};
                        {nom = "Flight_class"; vrai_typ = "String"; annot = ["Class_vol"]};
                    {nom = "Flight_price"; vrai_typ = "monnaie"; annot = ["Prix_vol"]}]};
  {context_Oper = "Rent-a-car"; annot_Oper = "Location-voiture";
   rOperlistIn = [{nom = "Pick_location"; vrai_typ = "string"; annot = ["Lieu_location";"Departure_city"]};
                  {nom = "Drop_location"; vrai_typ = "string"; annot = ["Lieu_visite";"Arrival_city"]};
                  {nom = "Tack_date"; vrai_typ = "date"; annot = ["Date_ prise"; "Leav_date"]};
                  {nom = "Back_date"; vrai_typ = "date"; annot = [“Date_ remise” ;”Return_date”]}];
   rOperlistOut = [{nom = “Locatair_cod”; vrai_typ = “string”; annot = ["Id_locatair"]};
                   {nom = "Pick_location"; vrai_typ = "string"; annot = [" Lieu_location";"Departure_city"]};
                   {nom = "Drop_location"; vrai_typ = "string"; annot = ["Lieu_visite";"Arrival_city"]};
                   {nom = "Tack_date"; vrai_typ = "date"; annot = ["Date_prise";"Leav_date"]};
                   {nom = "Back_date"; vrai_typ = "date"; annot = ["Date_remise";"Return_date"]};
                   {nom = "Car_typ"; vrai_typ = "string"; annot = ["type_vehicule"]};
                    {nom = "Location_price"; vrai_typ = "monnaie"; annot = ["Prix_location"]}]};
    {context_Oper = "Payment"; annot_Oper = "Versement@Payement";
     rOperlistIn = [{nom = "Flight_price" ; vrai_typ = "monnaie" ; annot= [“Prix_vol"; "Prix_transport"]};
                    {nom = "Rent_price" ; vrai_typ = "monnaie" ; annot= [“Prix_location"]}];
          rOperlistOut = [{nom = "Total_price" ; vrai_typ = "monnaie" ; annot = ["Prix_total"]}]}|]}
```

5. Construction des enregistrements associés à chacune des opérations des services

```
# type eltOperServ = {context_Oper: string; annot_Oper : string; sOperlistIn: eltInOut list; sOperlistOut: eltInOut list};;
```

eltOperServ est un enregistrement contenant les informations de chaque opération du service.

- **Remplissage des différents champs de chaque opération (record)**

a. Reserv-vol

```
# let operServ1 = { context_Oper = "Reserv-vol"; annot_Oper = "Reserv-flight"; sOperlistIn = sOperlistIn1;
                    sOperlistOut = sOperlistOut1};;
  val operServ1 : eltOperServ =
   {context_Oper = "Reserv-vol"; annot_Oper = "Reserv-flight";
    sOperlistIn = [{nom = "Ville_depar"; vrai_typ = "string"; annot = ["From_city"; "Departure_city"]};
                   {nom = "Ville_arrive"; vrai_typ = "string"; annot = ["To_city";"Arrival_city"]};
                   {nom = "Date_depar"; vrai_typ = "date"; annot = ["Leav_dat"]};
                   {nom = "Date_retour"; vrai_typ = "date"; annot = ["Return_dat"]};
                   {nom = "Nb_personne"; vrai_typ = "int"; annot = ["Nb_traveler"]};
                   {nom = "Class_vol"; vrai_typ = "string"; annot = ["Flight_class"]}];
    sOperlistOut = [{nom = "Code_vol"; vrai_typ = "string"; annot = ["Flight_code"]};
                    {nom = "Date_depar"; vrai_typ = "date"; annot = ["Leav_date"]};
                    {nom = "Date_retour"; vrai_typ = "date"; annot = ["Return_date"]};
                    {nom = "Ville_depar "; vrai_typ = "String"; annot = ["Departure_city";"From_city"]};
                    {nom = "Ville_arrive"; vrai_typ = "String"; annot = ["Arrival_city";"To_city"]};
                    {nom = "Nb_place"; vrai_typ = "int"; annot = ["Place_nbr"]};
                    {nom = "Class_vol"; vrai_typ = "String"; annot = ["Flight_class"]};
                    {nom = "Prix_vol"; vrai_typ = "monnaie"; annot = ["Flight_price"]}]}
```

operServ1 correspond à la première opération "Reserv-vol" du 1er service.

6.1. 1ère liste des opérations du service "Vol-WS"

b. Location-voiture

```
# let operServ2 = { context_Oper = "Location-voiture"; annot_Oper = "Rent-a-car";
                    sOperlistIn = sOperlistIn2; sOperlistOut = sOperlistOut2};;
```

c. Payer

```
# let operServ3 = { context_Oper = "Versement"; annot_Oper = "Payment"; sOperlistIn = sOperlistIn3;
                    sOperlistOut = sOperlistOut3};;
```

6. Construction des listes des opérations des services

Dans notre exemple, nous avons étudié le cas où un service Web contient une seule opération, donc on va construire une liste contenant une seule opération pour chaque service (voir figure 4). Pour construire la liste des opérations des services possédant plus d'une opération (par exemple Vol-Voiture-Ws montré dans la figure 17) il suffit d'utiliser le même principe décrit dans ***2.1.***

6.1. 1ère liste des opérations du service "Vol-WS"

```
# let servOperList1 = [operServ1];;
val servOperList1 : eltOperServ list =
      [{context_Oper = "Reserv-vol"; annot_Oper = "Reserv-flight";
       sOperlistIn = [{nom = "Ville_depar"; vrai_typ = "string"; annot = ["From_city"; "Departure_city"]};
                      {nom = "Ville_arrive"; vrai_typ = "string"; annot = ["To_city";"Arrival_city"]};
                      {nom = "Date_depar"; vrai_typ = "date"; annot = ["Leav_dat"]};
                      {nom = "Date_retour"; vrai_typ = "date"; annot = ["Return_dat"]};
                      {nom = "Nb_personne"; vrai_typ = "int"; annot = ["Nb_traveler"]};
                      {nom = "Class_vol"; vrai_typ = "string"; annot = ["Flight_class"]}];
       sOperlistOut = [{nom = "Code_vol"; vrai_typ = "string"; annot = ["Flight_code"]};
                       {nom = "Date_depar"; vrai_typ = "date"; annot = ["Leav_date"]};
                       {nom = "Date_retour"; vrai_typ = "date"; annot = ["Return_date"]};
                       {nom = "Ville_depar "; vrai_typ = "String"; annot = ["Departure_city";"From_city"]};
                       {nom = "Ville_arrive"; vrai_typ = "String"; annot = ["Arrival_city";"To_city"]};
                       {nom = "Nb_place"; vrai_typ = "int"; annot = ["Place_nbr"]};
                       {nom = "Class_vol"; vrai_typ = "String"; annot = ["Flight_class"]};
                       {nom = "Prix_vol"; vrai_typ = "monnaie"; annot = ["Flight_price"]}]}]
```

6.2. 2ème liste des opérations du service "Location-WS"

```
# let servOperList2 = [operServ2];;
```

6.3. 3ème liste des opérations du service "Bank-Ws"

```
# let servOperList3 = [operServ3];;
```

7. Construction du tableau "ServList" qui contient un ensemble d'enregistrements, chaque enregistrement correspond à un service qui est constitué des éléments suivants : ContextServ, AnnotServ et servOperList qui est une liste des opérations du service Web.

7.1. Construction des enregistrements du tableau "ServList"

```
# type eltServList = { context_Serv: string; annot_Serv:string; servOperList: eltOperServ list };;
```

- **Remplissage des différents champs de chaque service (record)**

a. Vol-WS

```
# let servList1 = { context_Serv = "Vol" ; annot_Serv = "Voyage@Travel" ; servOperList = servOperList1};;
val servList1 : eltServList =
                    { context_Serv = "Vol"; annot_Serv = "Voyage@Travel";
                     servOperList =
       [{context_Oper = "Reserv-vol"; annot_Oper = "Reserv-flight";
       sOperlistIn = [{nom = "Ville_depar"; vrai_typ = "string"; annot = ["From_city"; "Departure_city"]};
                      {nom = "Ville_arrive"; vrai_typ = "string"; annot = ["To_city";"Arrival_city"]};
                      {nom = "Date_depar"; vrai_typ = "date"; annot = ["Leav_dat"]};
                      {nom = "Date_retour"; vrai_typ = "date"; annot = ["Return_dat"]};
                      {nom = "Nb_personne"; vrai_typ = "int"; annot = ["Nb_traveler"]};
                      {nom = "Class_vol"; vrai_typ = "string"; annot = ["Flight_class"]}];
       sOperlistOut = [{nom = "Code_vol"; vrai_typ = "string"; annot = ["Flight_code"]};
                       {nom = "Date_depar"; vrai_typ = "date"; annot = ["Leav_date"]};
                       {nom = "Date_retour"; vrai_typ = "date"; annot = ["Return_date"]};
                       {nom = "Ville_depar "; vrai_typ = "String"; annot = ["Departure_city";"From_city"]};
                       {nom = "Ville_arrive"; vrai_typ = "String"; annot = ["Arrival_city";"To_city"]};
                       {nom = "Nb_place"; vrai_typ = "int"; annot = ["Place_nbr"]};
                       {nom = "Class_vol"; vrai_typ = "String"; annot = ["Flight_class"]};
                       {nom = "Prix_vol"; vrai_typ = "monnaie"; annot = ["Flight_price"]}]}]}
```

b. Location-WS

```
# let servList2 = {context_Serv = "Location"; annot_Serv = "Rent@Emprunt"; servOperList = servOperList2};;
```

c. Bank-WS

```
# let servList3 = {context_Serv = "Payement_reservation"; annot_Serv = "Versement@Déboursement@
Frais_Voyage" ; servOperList = servOperList3};;
```

7.2. Construction du tableau "ServList"

```
# let servList = [|servList1; servList2; servList3|];;
```

Le résultat est un tableau d'enregistrements de type "eltServList array".

Définissant maintenant les différentes fonctions associées à chaque opération de chaque service avec leurs arguments et les résultats qu'elles retournent.

```
# let reserv_Vol (sOperlistIn1: eltInOut list)  = sOperlistOut1;;
val reserv_Vol : eltInOut list -> eltInOut list = <fun>
val reserv_Vol : 'a -> eltInOut list = <fun>
```

Ce type est retourné dans le cas où on ne précise pas le type de la variable d'input "sOperlistIn1" , *sinon on peut mettre tout les elts de la liste d'input pour imposer le type.*

```
# let location-voiture (sOperlistIn2: eltInOut list)  = sOperlistOut2;;

# let versement (sOperlistIn3: eltInOut list)  = sOperlistOut3;;

if (request1.context_Req = servList.(0).context_Serv) || (request1.context_Req = "vol") then true else
false;;
```

On passe maintenant à la phase d'appariement.

Comme nous avons expliqués dans la *section 6.1.* , les étapes à suivre pour aboutir à une composition satisfaisant la requête sont récapitulées dans le schéma suivant :

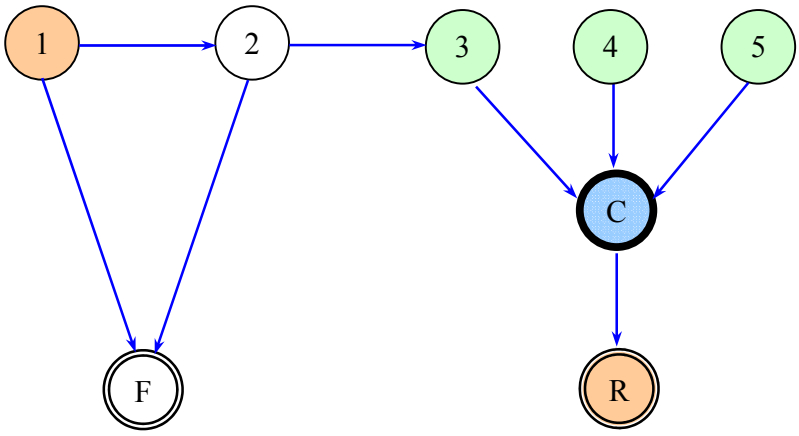


Figure 18- *Schéma de fonctionnement du système FS4SWC*

1. **Matching entre requête et services.**
    - Appariement entre contexte Req et Contexte Serv.
    - Appariement entre contexte Req et Annot Serv.
    - Appariement entre Annot Req et Annot Serv.

2. **Matching entre les opérations de la requête et les opérations de chaque service.**
    - Appariement entre contexte OperReq et Contexte OperServ.
    - Appariement entre contexte OperReq et Annot OperServ.
    - Appariement entre Annot OperReq et Annot OperServ.

3. **Matching entre les Inputs liste/ Outputs liste de la requête et Inputs liste/ Outputs liste des $Serv_k$ satisfaisant $OperReq_i$**
    - Appariement entre les champs "nom", "vrai_typ" et "annot" de chaque opération de la requête et les mêmes champs des opérations de chaque service.

4. **Ajouter indice $Serv_k$ et $OperServ_j$ à la liste de services satisfaisant $OperReq_i$**

5. **Matching entre les opérations de la requête**
    - Appariement entre la liste des Outputs de la $i^{ème}$ opération ($Oper_i$OutputList) et la liste des Inputs de la $i^{ème}$ +1 opération ($Oper_{i+1}$InputList).
    - Déduire le type de la composition des services (séquentielle et/ou parallèle).

C. Composition des services Web.

R. Résultat de la composition (Requête satisfaite).

F. Aucune correspondance trouvée / Fin du traitement (Requête non satisfaite).

Plus en détail (Algorithmiquement) :

```
Function Equivalent_RS (Request1, ServList[K]) :Boolean ;
    Var
      ExistAS : Boolean ;   EltAnnotServ, EltAnnotReq : String ;     EltCourantServ, EltCourantReq : Ptr ;
    Begin
        {======   Equivalence entre les context =====}
        Equivalent_RS :=False ;
        If Request1.ContextReq = ServList[K]. ContextServ then
            Equivalent_RS := True
        {======   Equivalence entre le contextReq et AnnotServ =====}
            Else
              Begin
                ExistAS := False ;
                EltCourantServ := ServList[K].AnnotServ ;
                    While (NeoList (AnnotServ)) and (Not ExistAS) do
                      Begin
                        EltAnnotServ := EltCourantServ.Valeur ;
                        If Request1.ContextReq = EltAnnotServ then
                            Begin
                                ExistAS := True ;
                                Equivalent_RS := True ;
                            End ;
                        EltCourantServ := Next ;
                      End ;
              End
        {======   Equivalence entre  AnnottReq et AnnotServ =====}
            Else
              Begin
                EltCourantReq := Request1.AnnotReq ;
                While (NeoList (AnnotReq)) do
                  Begin
                    EltAnnotReq := EltCourantReq.Valeur ;
                    ExistAS := False ;
                    EltCourantServ := ServList[K].ContextServ ;
                    While (NeoList (AnnotServ)) and (Not ExistAS) do
                      Begin
                        EltAnnotServ := EltCourantServ.Valeur ;
                        If EltAnnotReq = EltAnnotServ then
                            Begin
```

```
                                                    ExistAS := True ;
                                                    Equivalent_RS := True ;
                                                End;
                                            EltCourantServ := Next ;
                                        End ;
                                    EltCourantReq := Next ;
                                End ;
                            End ;
                END ;
```

==========

```
Function Equivalent_ORS (Request1.OperReq[j], ServList[k].PtrListOper^.EltOperServ) Boolean ;
Begin
{======  Equivalence entre les context =====}
        Equivalent_ORS :=False ;
        If Request1.OperReq[j].ContextOper = ServList[K].PtrListOper^.EltOperServ.ContextOper then
                Equivalent_ORS := True
{======  Equivalence entre le ContextOperReq et AnnotOperServ =====}
        Else
                Begin
                        ExistAOS := False ;
                        EltCourantOperServ := ServList[K]. PtrListOper^.EltOperServ.PtrContextOper ;
                        While (NeoList (AnnotOperServ)) and (Not ExistAOS) do
                                Begin
                                        EltAnnotOperServ := EltCourantOperServ.Valeur;
                                        If Request1.OperReq[j].ContextOperReq = EltAnnotOperServ then
                                                Begin
                                                    ExistAOS := True ;
                                                    Equivalent_ORS := True ;
                                                End ;
                                        EltCourantOperServ := Next ;
                                End ;
                End
```

==========

```
=========================================================================

{======  Equivalence entre le AnnottReq et AnnotServ =====}
        Else
           Begin
                EltCourantOperReq := Request1.OperReq[j].PtrContOper ;
                While (NeoList (AnnotOperReq)) do
                Begin
                     EltAnnotOperReq := EltCourantOperReq.Valeur ;
                     ExistAOS := False ;
                     EltCourantOperServ := ServList[K]. PtrListOper^.EltOperServ.PtrContextOper ;
                     While (NeoList (AnnotOperServ)) and (Not ExistAOS) do
                        Begin
                             EltAnnotOperServ := EltCourantOperServ.Valeur;
                             If EltAnnotOperReq = EltAnnotOperServ then
                                Begin
                                   ExistAOS := True ;
                                   Equivalent_ORS := True ;
                                End;
                             EltCourantOperServ := Next ;
                        End ;
                End ;
                EltCourantOperReq := Next ;
           End
End ;
END ;

=========================================================================
```

==========

```
Function Equivalent_InORS (Request1.OperReq[j].PtrIn, ServList[k].PtrListOper^.PtrIn) : Boolean ;
Var
        PtrOperReqIn, PtrOperServIn: Pointeur;
        Exist: Boolean;
Begin
        Equivalent_InORS := true;
        PtrOperReqIn := Request1.OperReq[j].PtrIn;
        PtrOperServIn := ServList[k].PtrListOper^.PtrIn;

        While (NeoList (OperReqIn)) and (Equivalent_InORS) do
           Begin
                Exist := false;
                While (NeoList (OperServIn)) and (not Exist) do
                   Begin
                      if (PtrOperReqIn^.nom = PtrOperServIn^.nom) Or (PtrOperReqIn^.ContextEltIn =
                                          PtrOperServIn^.ContextEltIn) then    Exist := true;
                      PtrOperServIn := Next;
                   End;
                if (not Exist) then
                     Equivalent_InORS:=False;
                PtrOperReqIn := Next ;
           End;
End;
```

==========

==========

Function Equivalent_OutORS (**Request1.OperReq[j].PtrOut**, **ServList[k].PtrListOper^.PtrOut**) : Boolean ;

```
Var
        PtrOperReqOut, PtrOperServOut: Pointeur;
        Exist: Boolean;
Begin
        Equivalent_OutORS := true;
        PtrOperReqOut := Request1.OperReq[j].PtrOut;
        PtrOperServOut := ServList[k].PtrListOper^.PtrOut;

        While (NeoList (OperReqOut)) and (Equivalent_OutORS) do
           Begin
              Exist: = false;
              While (NeoList (OperServOut)) and (not Exist) do
                 Begin
                    if (PtrOperReqOut^.nom = PtrOperServOut^.nom) Or (PtrOperReqOut^.ContextEltOut
                                        = PtrOperServOut^.ContextEltOut) then    Exist := true;
                    PtrOperServOut := Next;
                 End;
              if (not Exist) then
                   Equivalent_OutORS:=False;
              PtrOperReqOut := Next ;
           End;
End;
```

==========

6.3. Conclusion

A travers une vue pratique, cette partie est consacrée à une présentation de notre système ainsi que les algorithmes nécessaires pour la mise en œuvre de la découverte, l'appariement et la composition de Services Web Sémantique, et une présentation succincte du langage CAML qui sert de référence à l'implémentation des services et la sémantique qui leur sont associés ainsi que la composition de ces derniers.

6.4. Etude comparative

Dans le Chapitre 2*, nous avons présenté un panorama des principales approches qui peuvent être mises en oeuvre dans le cadre de la composition des services Web, notre choix s'est porté sur l'approche fonctionnelle qui est l'une des approches les plus efficaces basée à la fois sur le dynamisme, l'exactitude et la sémantique. Nous allons maintenant comparer notre approche avec les approches décrite précédemment, et notre architecture proposée avec l'architecture METEOR-S et StarWSCoP.*

Selon les quatre exigences décrites dans la section 2.7, on peut dire que l'approche basée sur les fonctions offre une connectivité et une composition automatique de services et une bonne scalabilité de la composition. Elle fournit de plus une spécification des propriétés de qualité de service non fonctionnelles par l'annotation sémantique des services, leurs opérations et leurs inputs/outputs comme expliquer précédemment, une vérification très forte de la conformité et la compatibilité de services à l'aide des types de données abstraits et sémantiques, et une preuve de la composition de services obtenue.

L'avantage de notre approche est de présenter les services comme des « boîtes noires » à l'aide du formalisme fonctionnel. En fait, les entrées/sorties d'un service sont gérées au sein des fonctions dont on connaît leurs types de données (sémantique) et leurs types de données abstraits, et qui sont sauvegardés dans des structures de données sémantiques. Ceci permet une meilleure découverte et composition de service. L'avantage du choix du formalisme fonctionnel est qu'il permet d'une part de remplacer les noms des services, noms des opérations, noms des paramètres d'inputs/outputs par d'autres noms qui ne sont pas identiques syntaxiquement mais qui s'apparient sémantiquement à l'aide de l'***α conversion***, et d'autre part de déduire automatiquement le type du service composite à l'aide de l'***isomorphisme de curry howard.***

Notre architecture a le même objectif avec METEOR-S (Managing End-To-End OpeRations for Semantic Web Services) et StarWSCoP (Star Web Services Composition Plateform). Cet objectif est de fournir des services web sémantiques en annotant les services Web existant, la découverte et la composition de ces services afin de répondre aux demandes des clients.

La différence réside dans le principe de fonctionnement ainsi que les phases qui constituent chacune des architectures.

La plateforme METEOR-S **[VER et al05]** s'est développée selon trois phases principales qui ont permis d'introduire les trois concepts importants de cette initiative :

- mise en place de l'infrastructure : il consiste à installer une infrastructure de découverte sémantique définie au dessus d'un registre UDDI et qui est appelée MWSDI (*METEOR-S Web Service Discovery Infrastructure*);

- annotation sémantique : il définit l'outil MWSAF (METEOR-S Web Service Annotation Framework) et qui permet d'enrichir sémantiquement les services web en utilisant une extension enrichie de WSDL appelée WSDL-S (WSDL Semantics). Le rôle de WSDL-S **[SW10]** est d'ajouter un niveau sémantique aux services web à travers l'enrichissement des fichiers WSDL mais également du registre enrichi UDDI;

- la composition et l'exécution de services : il a pour rôle d'assembler des services pour composer des services complexes en se basant sur BPEL4WS. L'outil se chargeant de cette tâche s'appelle MWSCF (METEOR-S *Web Service Composition Framework*).

Cette architecture comporte deux modules: le module front-end et le module back-end. Le module front-end est utilisé pour créer des services web sémantiques en procédant à l'annotation du code source avec des ontologies. Les fichiers WSDL-S sont ensuite publiés dans un registre UDDI enrichi. Le module de back-end offre principalement des fonctionnalités liées à la découverte de services, à la composition de services et à l'exécution et l'orchestration des services composés.

StarWSCoP (Star Web Services Composition Plateform) **[SUN et al03]** est l'une des plateformes de composition dynamique de services Web. Il inclut plusieurs modules: un système intelligent qui décompose les exigences de l'utilisateur en des descriptions abstraites du service, un registre de service qui est un entrepôt du service web; un moteur de découverte pour trouver des services adéquats qui satisferont les exigences de l'utilisateur dans le registre de service, un moteur de composition qui supervise les services composés afin qu'il soit exécuté dans le bon ordre, un wrapper (adaptateur) qui assure l' interopérabilité des services hétérogènes qui ont été développés séparément par des fournisseurs différents, une bibliothèque qui stocke les traces d'informations sur l'exécution des service composite, une

estimation en temps réel des QoS du service composite et un moniteur de l'événement qui contrôle les événements et notifie le moteur de composition.

Cette plateforme ajoute à l'UDDI une couche basée sur les ontologies pour définir une sémantique pour les services Web. Pour assurer la composition dynamique des services web basé sur la QoS, WSDL est étendu avec des attributs de QoS, comme le temps, le coût, ou la fiabilité.

Notre plateforme est composée de *l'architecture de référence* (*voir* ***figure2***), à laquelle nous avons ajouté un *système d'annotation sémantique* qui est un composant dédié à l'annotation sémantique des SW (les offres et les demandes) en obtenant des SWS qui seront par la suite envoyés à un *composant de filtrage* qui extrait la description sémantique et il les stocke dans une *base d'annotation*. L'exécution de la requête s'effectue d'abord au niveau de l'annuaire sans la description sémantique d'une manière classique, puis une autre phase de recherche s'effectue au niveau de la base d'annotation pour rechercher les services Web sémantiques adéquats, c-à-d rechercher les descriptions sémantiques correspond aux services retournés par la première étape. Un *module d'appariement* de services s'effectue à deux niveaux : d'abord, la découverte d'un ensemble de services susceptibles de répondre à la requête du client, puis la sélection du service qui puisse, effectivement, y répondre. Lorsqu'une tâche est requise comme un objectif et un ensemble de services web doivent être sélectionnés pour accomplir cette tâche, un *moteur de composition* propose la composition et l'interopérabilité automatique de ces services.

Conclusion Générale

Nous avons étudié l'architecture des services Web, les différentes problématiques de recherche sur les services Web et nous nous sommes particulièrement intéressés aux travaux sur la *composition dynamique de services Web*.

Pouvoir composer des services Web est un enjeu important pour le développement des activités des entreprises sur Internet. En effet, pour une entreprise, l'utilisation d'un service Web d'une autre entreprise permet de réduire ses coûts de la même façon qu'elle fait sous-traiter certains aspects de sa production ou de ses tâches administratives. L'entreprise à donc besoin d'outils afin de pouvoir composer les différents services Web qu'elle utilise. Notamment, la composition de services Web est considérée comme un point fort, qui permet de répondre à des requêtes complexes en combinant les fonctionnalités de plusieurs services au sein d'une même composition.

Afin de permettre une composition dynamique, nous nous sommes basés sur l'*approche fonctionnelle*. Nous avons donc proposé *un système de programmation fonctionnelle* qui permet de *composer les services Web* avec l'étude de la description sémantique de ces services, où nous avons défini un formalisme qui permet de raisonner sur des structures de données, garantir la sémantique de la requête et des services Web et est muni des constructeurs pour composer les services. Le principe d'enrichissement sémantique est basé principalement sur l'utilisation d'un système d'annotation sémantique commun pour les services d'offre et de demande qui identifie les éléments pertinents à annoter dans un service à savoir : le *contexte du service*, le *contexte des opérations* et le *contexte des Inputs/Outputs* de chaque opération, puis il exploite une ontologie pour déterminer quels sont les concepts les plus spécifiques possibles de l'ontologie dont l'instance sera utilisée pour annoter chacun de ces éléments. Le résultat de cet enrichissement est un fichier WSDL annoté sémantiquement, qui doit être passé par la suite par un module de mise en correspondance qui a pour but de trouver, à partir d'une requête émise par un demandeur de service Web, celui ou ceux qui correspondent le mieux au profil qui est décrit en WSDL étendue par la sémantique. Les services Web sélectionnés par ce module et qui accomplissent une tâche composite vont être composés par le moteur de composition.

L'objectif recherché à travers l'utilisation de la sémantique est de permettre aux machines d'interpréter les données traitées et de saisir leur signification de manière automatique. Cet objectif est concrètement atteint par l'annotation sémantique des services.

Nous avons montré comment les langages fonctionnels peuvent être utilisés pour résoudre le problème de la composition des services Web, et plus précisément nous avons utilisé le langage CAML pour l'implémentation de l'exemple *"Organisation d'un voyage"* en faisant appel aux différents services d'une agence de voyage.

L'approche que nous avons proposée permet de rendre explicite la représentation des services Web et la sémantique qui leur sont associés, ainsi que la composition des services grâce à l'utilisation des *fonctions* et la *composition de fonctions.* Elle permet aussi à l'utilisateur de profiter de la simplicité et la puissance d'expression que les langages fonctionnels offrent et de la richesse de la sémantique claire de ces langages. Elle offre des possibilités très étendues de définition et de manipulation de structures de données très riches.

En conclusion, les travaux proches du notre se préoccupent principalement de la sémantique de services. Nous jugeons pertinent d'utiliser l'architecture classique des services Web employée dans les organisations afin que la mise en œuvre de l'annotation sémantique ne modifie pas les systèmes existants.

Comme perspectives, ce travail peut être complété dans le court terme, par une implémentation de l'algorithme d'appariement pour la découverte et l'algorithme de composition que nous avons proposé, afin d'évaluer le degré d'efficacité de ces derniers. D'autre part, de nombreuses perspectives pour le long terme peuvent être envisagées. Par exemple, l'introduction d'une sémantique formelle pour la vérification de la validité d'une composition car il existe de nombreux travaux qui proposent des modèles formels pour la composition de services Web. L'objectif d'une telle modélisation est de permettre l'utilisation de la puissance des langages formels pour effectuer des vérifications sur la validité d'une composition, vérifier des propriétés sur le service composite (e.g. l'absence d'interblocage) ou encore faciliter le processus de composition en permettant d'évaluer par exemple la composabilité de deux services, ou la remplaçabilité d'un service par un autre.

Un autre aspect peut également être traité, il s'agit de développer un algorithme d'appariement (matching) qui fait des correspondances entre les pré-conditions d'une action avec les effets d'une autre, ainsi que la prise en compte des besoins non-fonctionnels de l'utilisateur, comme par exemple des besoins de qualité de service et de sécurité.

Références bibliographiques

[ABS03] Ambroszkiewicz, S. (2003) Benatallah, B. and Shaw, M.C. (Eds.): "*Entish: An Approach to Service Composition*", TES 2003, LNCS 2819, Springer-Verlag Berlin Heidelberg, pp.168–178.

[ALV et al03] Alvarez, P. Guelfi, N. *et al.* (Eds.): "*A Java Coordination Tool for Web-service Architectures: The Location-Based Service Context*", FIDJI, LNCS 2604, Springer-Verlag Berlin Heidelberg, pp.1–14. (2003)

[AND03] T. Andrews, F. Curbera, and et al. "*Business Process Execution Language for Web Services*", Version 1.1.*http://www.ibm.com/developerworks/library/ws-bpel/*, 2003.

[ANF01] L.F. Andrade et J.L. Fiadeiro. "*Coordination technologies for web-services*". Dans *Workshop on Object-Oriented Web Services (at OOPSLA)*, 2001.

[ANT03] Grigoris Antoniou and Frank van Harmelen. "*Web ontology language : Owl* ". In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*. Springer-Verlag, 2003

[ARD04] D. Ardagna and B. Pernici. "*Global and local qos constraints guarantee in web services Selection* ". *In Proceedings of 2005 IEEE International Conference on Web Services (ICWS 2005),* volume II, pages 805–806, 2004.

[ARR04] S. Arroyo and M. Stollberg. "*WSMO Primer. WSMO Deliverable D3.1*", DERIWorking Draft. Technical report, WSMO, 2004. http://www.wsmo.org/2004/d3/d3.1/.

[BAC78] J.BACKUS, "*Can Programming Be Liberated from the Von Neuman Style? A Functional Style and its Algebra of Programs*" Com. of the ACM, vol. 21, n°8, pp 613-641, 08/78.

[BAR03] Douglas K. Barry. "*Web Services and Service-Oriented Architectures : The Savvy Manager's Guide*". Morgan Kaufmann, 2003.

[BCR02] T. Bellwood, L. Clément, and C. von Riegen. "*Universal description, discovery and Integration. Technical report, OASIS UDDI Specification Technical Committee*", mar 2002. http ://www.oasis-open.org/cover/uddi.html.

[BEN et al04] Benatallah, B., Casati, F. and Toumani, F. (2004b) "*Web Service Conversation Modeling.A Cornerstone for E-business Automation*", IEEE Internet Computing, January– February 2004.

[BER et al05] D. Berardi, D. Calvanese, D. G. Giuseppe, R. Hull, and M. Mecella. "*Automatic Composition of Transition-based Semantic Web Services with Messaging*". *31st International Conference on Very Large Databases*, pp. 613–624, 2005.

[BHI05] Sami Bhiri; Thèse Doctorat "*Approche Transactionnelle pour Assurer des Compositions Fiables de Services Web* " université Henri Poincaré- Nancy 1. Octobre 2005.

[BOO03] D. Booth, H. Haas, F. McCabe, and E. Newcomer. "*Web services architecture* ", aout 2003. http ://www.w3.org/TR/2003/WD-ws-arch-20030808/.

[BRO96] AlanWBrown. "*Component-based software engineering* " : selected papers from the Software Engineering Institute. Los Alamitos, CA : IEEE Computer Society Press, 1996.

[BSD03] B. Benatallah, Q. Sheng, and M. Dumas. "*The Self-Serv Environment for Web Services Composition*". *IEEE Internet Computing* 7(1):40–48, 2003.

[BUL et al03] T. Bultan, X. Fu, R. Hull, and J. Su, "*Conversation Specification: A New Approach to Design and Analysis of E-Service Composition*", Proceedings of the 12th International World Wide Web Conference (WWW 2003), ACM, 2003, pp. 403–410.

[BUN et al03] Bunting, D., Hurley, M.C.O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J. and Swenson, K. (2003) "*Web Services Context (WS-Context) Ver1.0*", http://developers.sun.com/ techtopics/webservices/wscaf/wsctx.pdf, July 28.

[CAB02] Cabrera, F. *et al.* "*Web Services Coordination (WS-coordination)*", ftp://www6.software.ibm.com/software/developer/library/ws-coordination.pdf (2002).

[CAS01] F. Casati & M-C. Shan (2001) "*Models and languages for describing and discovering eservices*".In SIGMOD '01 : Proceedings of the 2001 ACM SIGMOD international conference on Management of data, page 626, New York, NY, USA, 2001. ACM Press.

[CHA02] Extrait de l'ouvrage "*Services Web avec SOAP, WSDL, UDDI, ebXML...* " de Jean-Marie Chauvet, © Eyrolles, 2002.

[CHI05] Yannis Chicha1 Marc Ga¨etano , "*Mathematical Services Composition*", Electronic Note in Theoretical Computer Science 114 (2005) 103–117 www.elsevier.com/locate/entcs.

[CHR02] Dieter Fensel and Christoph Bussler."*The Web Service Modeling Framework WSMF*". *Electronic Commerce : Research and Applications*, 1 :113–137, 2002.

[CKB04] Luis Felipe Cabrera, Cristopher Kurt, Don Box "*Extrait du livre Introduction à l'architecture de services Web et ses spécifications WS-*" Première publication: DotNetGuru.org.. Octobre 2004.

[CNW01] F. Curbera, W. Nagy, and W. Weerawarana. "*Web services : Why and how?*" Conference on Object-Oriented Programming, Systems, Languages and Application (OOPSLA), Workshop on Object-Oriented Web Services, 2001.

[CSW01] F. Curbera, I. Silva-Lepe, and S. Weerawarana. "*On the integration of heterogeneous web service partners* " , 2001.
http ://www.research.ibm.com/people/b/bth/OOWS2001/curbera.pdf.

[CNW01] F. Curbera, A. Nagy, et S. Weerawarana. "*Web-services : Why and how*". Dans *Workshop on Object-Oriented Web Services (in OOPSLA)*, Aout 2001.
http ://www.research.ibm.com/people/b/bth/OOWS2001.html.

[COL et al99] N. Collier, Hyun Seok Park, Norihiro Ogata, Yuka Tateisi, Chikashi Nobata, Takeshi Sekimizu, Hisao Imai and Jun'ichi Tsujii. (1999). The GENIA project: "*corpus-based knowledge acquisition and information extraction from genome research papers*". In *Proceedings of the European Association for Computational Linguistics* (EACL 1999).

[COU95] Guy Cousineau, Michel Mauny. *"APPROCHE FONCTIONNELLE DE LA PROGRAMMATION"* EDISCIENCE international Paris 1995.

[CPF02] *"Cours de programmation fonctionnelle et logique"*. Décembre 2002.

[DAN03] Jérôme Daniel. Extrait de livre "*SERVICES WEB Concepts, techniques et outils*" Edition vuibert Informatique 2003.

[DUS05] Schahram tdar and Wolfgang Schreiner, "*A survey on web services composition*", *Int. J. Web and Grid Services*, *Vol. 1*, *No. 1*, *2005.*

[ELF et al04] Amal El Fallah-Seghrouchni; Serge Haddad; Tarak Melitti; Alexandru Suna Rapport "*Interopérabilité des systèmes multi-agents à l'aide des servicesWeb*", 2004.

[END et al 04] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, Tony Newling - 2004 – "*Pattern: Service Oriented Architecture and Web Services*", *IBM INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION* , ISBN 073845317X

[FBM02] D. Fensel, C. Bussler, & A. Maedche (2002). "*Semantic Web Enabled Web Services*. *In International Semantic Web Conference*", Sardinia, Italy ,volume 2348, pages1-2 .Invited paper.

[FIL06] Jean-Christophe Filliâtre, Master Informatique M1 "*Initiation à la programmation fonctionnelle*" Université Paris Sud 2005_2006

[GER06] A. Gerevini and D. Long. "*Preferences and Soft Constraints in PDDL3*. *ICAPS Workshop on Preferences and Soft Constraints in Planning*", 2006.

[GEO et al02] Georgakopoulos, D, Schuster, H., Cichocki, A. and Baker, D. (2002) "*Process-based e-service composition for modeling and automating zero latency supply chains*", Information System Frontiers, Kluwer Academic Publishers, Vol. 4, No. 1, pp.33–54.

[GHM00] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. Nielsen. "*Simple object access protocol (soap) 1.1*". Technical report, World Wide Web Consortium, may 2000. http ://www.w3.org/TR/SOAP/.

[GOT et al02] K. D. Gottschalk, S. Graham, H. Kreger & J. Snell (2002). "*Introduction to Web services architecture". IBM Systems Journal 41(2):* 170-177.

[GRU00] T. Gruber. "*What is an ontology ?*" http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, 2000.

[GUE96] Yves GUERTE Thèse de de DOCTORAT "*Dérivation de programmes impératifs à partir de spécifications algébriques*" octobre 1996.

[HAM03] R. Hamadi and B. Benatallah. "*A Petri Net-based Model forWeb Service Composition*". 14th Australasian Database Conference, pp. 191–200. Australian Computer Society, Inc., 2003.

[HAN et al02] Hansen, M., Madnick, S. and Siegel, M. (2002) Bussler, C. *et al.* (Eds.): "*Process Aggregation Using Web Services*", WES, LNCS 2512, Springer-Verlag Berlin Heidelberg, pp.12–27.

[HAR93]. T.A.Hardin, V.DG.Viguié. *"Concepts et outils de programmation – Le style fonctionnel, le style impératif, avec CAML et ADA".* InterEditions. Deuxième tirage corrigé, 1993.

[HEA01] K. Heather. "*Web services conceptual architecture (wsca 1.0)*", may 2001. http ://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[HEN88] M. Hennessy. "*Algebraic Theory of Processes*". MIT Press, 1988.

[HEN80]. P.HENDERSON, "*Functional Programming – Application and implementation*" Prentice Hall International Series in Computer Science – 1980.

[HER05] Nathalie HERNANDEZ, Docteur de l'Université Paul Sabatier de Toulouse, *"Ontologies de domaine pour la modélisation du contexte en recherche d'information "*, Laboratoire IRIT – Pôle SIG-EVI, Décembre 2005.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn, "*Dynamic logic*", The MIT Press, 2000.

[HUG89]. J.HUGHES, *"Why functional programming maters"* the Computer Journal, vol. 32, n°2, pp 98-107, 04/89.

[HUM04] Humberto, Cervantes - 2004 - "*Vers un modèle à composants orienté services pour supporter la disponibilité dynamique*", Université Joseph Fourier - Grenoble 1.

[ISM05] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. *"Business process execution language for web services"* version 1.1. http://www 128.ibm.com/developerworks/library/specification/ws-bpel/, July 2005.

[KEL03] P. Kellert and F. Toumani. *"Les web services sémantiques"*. In Web sémantique, Action spéci_que 32 CNRS/STIC, October 2003.

[KHK06] A.Khelladi, S.Kouici. *"Mesure de similarité structurelle pour la classification des données binaires" COSI'06* P 37-47.

[KKB et al04] Keidl, M., Kemper, A. Bertino, E. *et al.* (Eds.): *"A Framework for Context-aware Adaptable Web-services"*, EDBT, LNCS 2992, Springer-Verlag Berlin Heidelberg, pp.826–829. (2004).

[KKM05] M. Klein, B. König-Ries, and M. Müssig. "*What is needed for semantic service descriptions - a proposal for suitable language constructs*". International Journal on Web and Grid Services (IJWGS), 1(3/4) :328–364, 2005.

[LIU04] LIU C., FOSTER I. T., "*A Constraint Language Approach to Matchmaking*", *RIDE*, 2004, p. 7-14.

[MAR03] D. Martin. " *Owl-s : Semantic markup for web services*". Novembre 2003, http ://www.daml.org/services/owl-s/1.0/owl-s.html.

[MAR et al04] D. L. Martin, M. Paolucci, S. A. McIlraith, M. H. Burstein, D. V. McDermott, D. L. McGuinness, B. Parsia, T. R. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. P. Sycara. "*Bringing Semantics to Web Services : The OWL-S Approach*". In J. Cardoso and A. P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42. Springer, 2004.

[MAX01] E. M. Maximilien & M.P. Singh (2001). "*Conceptual Model of Web Service Reputation*", SIGMOD Record 31(4): 36-41.

[MCD02] D. V. McDermott. "*Estimated-Regression Planning for Interactions with Web Services*". 6th Intl. Conference on Artificial Intelligence Planning Systems, pp. 204–211, 2002.

[MCI01] S. McIlraith, T.C. Son, & H. Zeng (2001). "*Semantic Web Services*". IEEE Intelligent Systems. Special Issue on the Semantic Web, 16(2):46–53.

[MCI02]: S. McIlraith and T.C. Son, Adapting Golog *"for Composition of Semantic Web Services"*, Proceedings of the 8th International Conferences on Principles of Knowledge Representation and Reasoning (KR 2002), Morgan Kaufmann, 2002, pp. 482 – 493.

[MIL82] R. Milner. "*A Calculus of Communicating Systems*". Springer-Verlag New York, Inc., 1982.

[MIL et al04] J. Miller, K. Verma, P. Rajasekaran, A. Sheth, R. Aggarwal, and K. Sivashanmugam. "*WSDL-S : Adding Semantics to WSDL - White Paper*". Technical report, Large Scale Distributed Information Systems, 2004. http://lsdis.cs.uga.edu/library/ download/wsdl-s.pdf.

[MIL04] Nikola Milanovic and Miroslaw Malek , *"Current Solutions forWeb Service Composition"*. Humboldt University, Berlin IEEE INTERNET COMPUTING 1089-7801/04/$20.00 © 2004 IEEE Published by the IEEE Computer Society NOVEMBER ,DECEMBER 2004.

[MOR06] MAXIME MORNEAU, maître ès sciences (M.Sc.), *"Recherche d'information sémantique et extraction automatique d'ontologie du domaine"* FACULTÉ DES SCIENCES ET DE GÉNIE UNIVERSITÉ LAVAL QUÉBEC, Août 2006.

[MSZ01]: S. McIlraith, T.C. Son, and H. Zeng, *"Semantic Web Services"*, IEEE Intelligent Systems 16 (2001), no. 2, 46 – 53.

[MVH03] McGuinness D.L. and van Harmelen F.; "*OWL Web Ontology Language Overview*"; http://www.w3.org/TR/owl-features/ , Août 2003, World Wide Web Consortium - Candidate Recommendation.

[NAR02 a] S. Narayanan. Simulation, "*verification and automated composition of web" services*. In *Proceedings of the Eleventh International World Wide Web conference*, pages 77–88, Honolulu, HI, 2002.

[NAR02 b] S. Narayanan and S. McIlraith. "*Simulation, verification automated composition of web services*". In Proc. Int.Wide Web Conf. (WWW), 2002.18]

[ORR et al03] Orriens, B., Yang, J. and Papazoglou, M.P. Orlowska, M.E. *et al.* (Eds.): "*Model Driven Service Composition*", ICSOC, LNCS 2910, Springer-Verlag Berlin Heidelberg, pp.75–90. (2003)

[ORR et al03 a] Orriens, B., Yang, J. and Papazoglou, M.P. (2003b) Jeusfeld, M.A. and Pastor, Ó. (Eds.): "*A Framework for Business Rule Driven Web Service Composition*", ER 2003 Workshops, LNCS 2814, Springer-Verlag Berlin Heidelberg 2003, pp.52–64.

[ORR et al03 b] Orriens, B., Yang, J. and Papazoglou, M.P. (2003c) Benatallah, B. and Shan, M-C. (Eds.): "*A Framework for Business Rule Driven Service Composition*", TES, LNCS 2819, Springer-Verlag Berlin Heidelberg, pp.14–27.

[PAO et al02] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. *"Importing the Semantic Web in UDDI".* In *Proceedings of E-Services and the Semantic Web Workshop*, 2002.

[PAP03] Papazoglou, M.P. (2003) "*Web Services and Business Transactions*", World Wide Web: Internet and Web Information Systems, Kluwer Academic Publishers, Vol. 6, pp.49–91.

[PBM02] P. F. Pires and M. Benevides and M. Mattoso. "*WEBTRANSACT: a Framework For Specifying And Coordinating Reliable Web Services Compositions*". Technical report, sherry.i_.unizh.ch/pires02webtransact.html, 2002.

[PEE05] J. Peer. *"Semantic Service Markup with SESMA".* In *Web Service Semantics Workshop (WSS'05) at the Fourteenth International World Wide Web Conference (WWW'05)*, 2005.

[PEY90] S.L.Peyton-Jones. "*Mise en œuvre des langages fonctionnels de programmation*". Manuels Informatiques Masson. Masson,1990.

[PFI96] Cuno Pfister and Clemens Szyperski. "*Why objects are not enough* ". In *Proceedings, International Component Users Conference*, Munich, Germany, 1996. SIGS.

[PIC et al03] Piccinelli, G. and Williams, S.C. (2003) van der Aalst, W.M.P. *et al.* (Eds.): "*Workflow: A Language for Composing Web Services*", BPM, LNCS 2678, Springer-Verlag Berlin Heidelberg, pp.13–24.

[PIE91] G.Pierra. DUNOD *"Les bases de la programmation et du génie logiciel",* Paris 1991.

[PIS et al05 a] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. *"Automated Composition of Web Services by Planning at the Knowledge Level".* 19th Intl. Joint Conferences on Artificial Intelligence, pp. 1252–1259, 2005.

[PIS et al05 b] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. *"Automated Synthesis of composite BPEL4WSWeb Services".* 3rd Intl. Conference on Web Services, pp. 293–301. IEEE Press, 2005.

[PRE02] Preuner, G. and Schrefl, M. "*Integration of web services into workflows through a multilevel schema architecture*", *Proceedings of the 4th IEEE Int'l Workshop on Advanced Issues of E-commerce and Web-based Information Systems (WECWIS 2002)*, IEEE. (2002)

[PRT01] Rapport d'activités *"Programmation typée, modularité et compilation"* Rocquencourt INRIA 2001.

[RAN03] Ran, S. (2003) "*A model for web services discovery with QoS*", ACM SIGecom Exchanges, Vol. 4, No. 1, pp.1–10.

[RIC04] Ricardo DE LA ROSA-ROSERO. ProjetMaster Mathématiques Informatique *"Découverte et Sélection de Services Web pour une application Mélusine".*Septembre 2004.

[RUO04] RUOYAN ZHANG, "*Ontology-Driven Web Services Composition Techniques*", thèse de Master en science ATHENS, GEORGIA University 2004.

[SBS04] G. Salaün, L. Bordeaux, and M. Schaerf. *"Describing and Reasoning on Web Services using Process Algebra". 2nd IEEE International Conference on Web Services*, pp. 43–50. IEEE Computer Society,2004.

[SHE et al02] Sheng, Q.Z., Benatallah, B., Dumas, M. and Mak, E.O-Y. (2002) *"SELF-SERV: a platform for rapid composition of web services in a peer-to-peer environment"*, Proceedings of the 28th VLDB Conference, Hong Kong, China.

[SHU03] Shuping Ran; "*A model for web services discovery with QdS*"; ACM SIGecom Exchanges; Volume 4 , Issue 1 Spring, 2003; ACM Press; http://doi.acm.org/10.1145/844357.844360.

[SOA00] SOAP, *" Simple Object Access Protocol (SOAP) 1.1"*, rapport, may 2000, World Wide Web Consortium, http ://www.w3.org/TR/SOAP/.

[STO77]. J.E.STOY *"Denotational Semantics : the Scott-Strachey approach to programming language theory".* M.I.T Press, 1977.

[SUN et al03] Sun, H., Wang, X., Zhou, B. and Zou, P. (2003) *"Research and Implementation of Dynamic Web Services Composition"*, APPT 2003, LNCS 2834, Springer-Verlag Berlin Heidelberg, pp.457–466.

[SYC et al03] Katia Sycara, Massimo Paolucci., Anupriya Ankolekar, Naveen Srinivasan *"Automated discovery, interaction and compositionof Semantic Web services",* Carnegie Mellon University, Pittsburgh, Pennsylvania, USA,18 July 2003.

[SZY97] Clemens Szyperski. "*Component software : beyond object-oriented programming*" New York : ACM Press Harlow, England Reading, Mass : Addison-Wesley, 1997.

[TEA01] Wrox Author Team. *"Service Web XML Professionnel"*. Wrax, Paris, December 2001.

[UDD02] UDDI, *"Universal Description, Discovery and Integration"*, rapport, mar 2002, OASIS UDDI Specification Technical Committee, http ://www.oasisopen.org/cover/uddi.html.

[VER et al05] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller *"METEOR–S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services",* Journal of Information Technology and Management, Special Issue on Universal Global Integration, Vol. 6, No. 1 (2005) pp. 17-39. Kluwer Academic Publishers. (pre-publication version).

[WSD01] WSDL, *" Web Services Description Language (WSDL) 1.1"*, rapport, mar 2001, World Wide Web Consortium, http ://www.w3.org/TR/wsdl.

[WVD02] P. Wohed, W. Van Der Aalst, and M. Dumas. *"Pattern based analysis of BPEL4WS"*. Technical Report FIT-TR-2002-04, Centre for Information Technology Innovation, Queensland University of Technology, Australia, April 2002.

[W3C06] W3C. Hypertext transfer protocol. http ://www.w3.org/Protocols/, 2006.

[YAN01] J. Yang and M. Papazoglou. "*Web components : A substrate for web service reuse and composition* ". In Proceedings of the 14th International Conference on Advanced Information Systems Engineering, pages 21–36, Toronto, Canada, 2001.

[YAN02] Jian Yang and Mike. P. Papazoglou "*Web Component: A Substrate for Web Service Reuse and Composition*" *Proc. 14th Conf. Advanced Information Systems Eng.* (CAiSE02) LNCS 2348, Springer-Verlag, 2002, pp. 21–36.

[ZEN et al03] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng. *" Quality driven web services composition". In Proceedings of Twelfth International Conference of WWW, Budapest, Hungary, 2003.*

[ZEN et al04] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and H. Chang. *"Qos-aware middleware for web services composition". In IEEE Trans. on Software Engineering, 2004.*

Sites web

[SW1] OASIS SOA Reference Model TC :
http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=soa-rm

[SW2] The OMG and Service Oriented Architecture :
http://www.omg.org/attachments/pdf/OMG-and-the-SOA.pdf

[SW3] Web Services Architecture, W3C,
http://www.w3.org/TR/ws-arch/

[SW4] http://www.w3.org/2002/ws/

[SW5] "*Web Services Desscription Language, WSDL*". http://www.w3.org/TR/wsdl.

[SW6] R. Chinnici, M. Gudgin, J.J. Moreau, J. Schlimmer, and S. Weerawarana. "*Web Services Description Language (WSDL)* " 2.0. Working draft, W3C. Available on line" : http://www.w3. org/TR/wsdl20.

[SW7] "*Web Services Business Process Execution Language, WSBPEL*". http://www.oasis-open.org.

[SW8] Web Services Choreography Description Language, WS-CDL. http://www.w3.org/2002/ws/chor/.

[SW9] " *SAWSDL Working Group. Semantic Annotations for WSDL*". W3c working draft, The World Wide Web Consortium (W3C), Sept. 2006. http://www.w3.org/TR/ 2006/WD-sawsdl-20060928/.

[SW10] "*WSDL-S, Web Service Semantics - WSDL-S, Version 1.0*". W3C Member Submission, [On Line] http://www.w3.org/Submission/WSDL-S/, 2005.